DM503

Forelæsning 5

Information

- Ingen forelæsning på torsdag
- Obligatorisk opgave
  - Spørgsmål?

Indhold

- Introduktion til mål for programmers køretid
- Sorteringsalgoritmer
  - Udvælgelsessortering
  - Boble-sortering
  - Indsættelsessortering

Sortering

Sortering

- Eksempel på måling af algoritmers køretid

Sortering

- Eksempel på måling af algoritmers køretid
- Sortering af et array A[0,...,n-1]

Sortering

- Eksempel på måling af algoritmers køretid
- Sortering af et array A[0,...,n-1]
  - Givet et array med n elementer

Sortering

- Eksempel på måling af algoritmers køretid
- Sortering af et array A[0,...,n-1]
  - Givet et array med n elementer

  - Arranger elementerne så de står i sorteret rækkefølge

Køretid

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"
- Vi vil gerne kunne sige at algoritme A er mere effektiv end algoritme B

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"
- Vi vil gerne kunne sige at algoritme A er mere effektiv end algoritme B
  - Bruge det til at afgøre hvilken algoritme vi vil bruge

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"
- Vi vil gerne kunne sige at algoritme A er mere effektiv end algoritme B
  - Bruge det til at afgøre hvilken algoritme vi vil bruge
- Hvis inputtet bliver dobbelt så stort, så tager algoritme A x gange så lang tid

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"
- Vi vil gerne kunne sige at algoritme A er mere effektiv end algoritme B
  - Bruge det til at afgøre hvilken algoritme vi vil bruge
- Hvis inputtet bliver dobbelt så stort, så tager algoritme A x gange så lang tid
- ...

Køretid

- Vi vil gerne kunne snakke om effektiviteten af en algoritme
  - "Hvor hurtig er den?"
- Vi vil gerne kunne sige at algoritme A er mere effektiv end algoritme B
  - Bruge det til at afgøre hvilken algoritme vi vil bruge
- Hvis inputtet bliver dobbelt så stort, så tager algoritme A x gange så lang tid
- ...
- Vi vil altså gerne have et mål/tal for effektiviteten af en algoritme

Køretid

Køretid

- 1. forsøg

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer
  - Kør algoritmen med input I

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer
  - Kør algoritmen med input I
  - Tag tid

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer
  - Kør algoritmen med input I
  - Tag tid
  - Målet er nu den tid det tog

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer
  - Kør algoritmen med input I
  - Tag tid
  - Målet er nu den tid det tog
- Godt eller skidt mål?

Køretid

- 1. forsøg
  - Implementer algoritmen i et sprog (fx Java)
  - Køb en computer
  - Kør algoritmen med input I
  - Tag tid
  - Målet er nu den tid det tog
- Godt eller skidt mål?
  - Kan vi nu svare på de spørgsmål vi gerne ville?

Køretid

Køretid

- Problemer med 1. forsøg

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I
  - Afhængigt af computeren

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I
  - Afhængigt af computeren
  - Afhængigt af implementeringen

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I
  - Afhængigt af computeren
  - Afhængigt af implementeringen
  - Afhængigt af det valgte sprog

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I
  - Afhængigt af computeren
  - Afhængigt af implementeringen
  - Afhængigt af det valgte sprog
  - ...

Køretid

- Problemer med 1. forsøg
  - Afhængigt af input instansen I
  - Afhængigt af computeren
  - Afhængigt af implementeringen
  - Afhængigt af det valgte sprog
  - ...
- Hvad gør vi så?

Køretid

Køretid

- Afhængigt af input instansen I

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n
  - Tag gennemsnit

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n
  - Tag gennemsnit
  - Find maksimum

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n
  - Tag gennemsnit
  - Find maksimum
- Problemet?

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n
  - Tag gennemsnit
  - Find maksimum
- Problemet?
  - Oftest er der uendeligt mange mulige instanser

Køretid

- Afhængigt af input instansen I
- Prøv med flere instanser I_1, I_2, ..., I_n
  - Tag gennemsnit
  - Find maksimum
- Problemet?
  - Oftest er der uendeligt mange mulige instanser
  - Der findes uendeligt mange lister man kan finde på at ville sortere

Køretid

Køretid

- Afhængigt af computeren

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet
  - Afhængigt af hvilke andre programmer der kører

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet
  - Afhængigt af hvilke andre programmer der kører
  - ...

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet
  - Afhængigt af hvilke andre programmer der kører
  - ...
- Afhængigt af sproget og implementering

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet
  - Afhængigt af hvilke andre programmer der kører
  - ...
- Afhængigt af sproget og implementering
  - Rigtigt mange forskellige sprog

Køretid

- Afhængigt af computeren
  - Afhængigt af hastighed (klokfrekvens)
  - Afhængigt af hukommelsen (RAM, cache, ...)
  - Afhængigt af operativsystemet
  - Afhængigt af hvilke andre programmer der kører
  - ...
- Afhængigt af sproget og implementering
  - Rigtigt mange forskellige sprog
  - Uendeligt mange måder at implementere samme algoritme på

Køretid

Køretid

- Konklusion

Køretid

- Konklusion
  - Vi kan ikke basere et mål for effektiviteten af en algoritme på målinger af hvor hurtigt en konkret implementering løser en række konkrete instanser

Køretid

- Konklusion
  - Vi kan ikke basere et mål for effektiviteten af en algoritme på målinger af hvor hurtigt en konkret implementering løser en række konkrete instanser
- Hvad gør vi så?!?

Køretid

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser
  - Afhængigt af computeren, implementationen og det valgte sprog

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser
  - Afhængigt af computeren, implementationen og det valgte sprog
    - Matematisk kan vi forestille os algoritmen (ikke en implementering) på en abstrakt computer

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser
  - Afhængigt af computeren, implementationen og det valgte sprog
    - Matematisk kan vi forestille os algoritmen (ikke en implementering) på en abstrakt computer
- Skaber nyt problem, hvilket?

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser
  - Afhængigt af computeren, implementationen og det valgte sprog
    - Matematisk kan vi forestille os algoritmen (ikke en implementering) på en abstrakt computer
- Skaber nyt problem, hvilket?
  - Får et teoretisk mål

Køretid

- Kig på algoritmens effektivitet fra et teoretisk synspunkt
- Løser alle ovenstående problemer
  - Afhængigt af input instansen I
    - Matematisk kan vi sagtens håndtere uendeligt mange instanser
  - Afhængigt af computeren, implementationen og det valgte sprog
    - Matematisk kan vi forestille os algoritmen (ikke en implementering) på en abstrakt computer
- Skaber nyt problem, hvilket?
  - Får et teoretisk mål
  - Skulle helst sige noget om virkeligheden

Køretid

Køretid

- Hvad man gør konkret

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet
  - Antallet af gange den udføres skal være proportional med det totale antal udførte operationer

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet
  - Antallet af gange den udføres skal være proportional med det totale antal udførte operationer
  - DM507

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet
  - Antallet af gange den udføres skal være proportional med det totale antal udførte operationer
  - DM507
- Tæl hvor mange gange operationen bliver udført som funktion af inputets størrelse

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet
  - Antallet af gange den udføres skal være proportional med det totale antal udførte operationer
  - DM507
- Tæl hvor mange gange operationen bliver udført som funktion af inputets størrelse
  - I værste tilfælde (worst-case analysis)

Køretid

- Hvad man gør konkret
- Kig på algoritmen (IKKE en implementering)
- Vælg en karakteristisk operation for algoritmen
- Operationen skal være fundamental for problemet
  - Antallet af gange den udføres skal være proportional med det totale antal udførte operationer
  - DM507
- Tæl hvor mange gange operationen bliver udført som funktion af inputets størrelse
  - I værste tilfælde (worst-case analysis)
  - I gennemsnit (average-case analysis)

Køretid

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	Dimensionerne af de to matricer

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	Dimensionerne af de to matricer
Sorter et array	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	Dimensionerne af de to matricer
Sorter et array	Antallet af elementer i arrayet

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	Dimensionerne af de to matricer
Sorter et array	Antallet af elementer i arrayet
Gennemløb af et træ	

Køretid

- Eksempler på karakteristisk operationer

Problem	Operation
Find x i array	Sammenligning af x og en indgang i arrayet
Gang to matricer sammen	Multiplikation af to indgange (eller multiplikation og addition)
Sorter et array	Sammenligning af to indgange i arrayet
Gennemløb af et træ	At følge én kant i træet

- Eksempler på inputets størrelse

Problem	Størrelsen af input
Find x i array	Antallet af elementer i arrayet
Gang to matricer sammen	Dimensionerne af de to matricer
Sorter et array	Antallet af elementer i arrayet
Gennemløb af et træ	Antallet af knuder i træet

Sortering

Sortering

- Tilbage til sortering

Sortering

- Tilbage til sortering
- Tre algoritmer
  - Udvælgelsessortering
  - Boble-sortering
  - Indsættelsessortering

Udvælgelsessortering

Udvælgelsessortering

- Idé

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...
- Hmm... lidt svært at afgøre om et element er "næst-mindst"

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...
- Hmm... lidt svært at afgøre om et element er "næst-mindst"
  - Hvis det mindste element er forrest (A[0])

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...
- Hmm... lidt svært at afgøre om et element er "næst-mindst"
  - Hvis det mindste element er forrest (A[0])
  - så er det næste-mindste element i arrayet A

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...
- Hmm... lidt svært at afgøre om et element er "næst-mindst"
  - Hvis det mindste element er forrest (A[0])
  - så er det næste-mindste element i arrayet A
  - det mindste elemente i del-arrayet A[1,...,n-1]

Udvælgelsessortering

- Idé
  - Find det mindste element i arrayet A
  - Flyt det forrest
  - Find det næst-mindste element i arrayet A
  - Flyt det næst-forrest
  - ...
- Hmm... lidt svært at afgøre om et element er "næst-mindst"
  - Hvis det mindste element er forrest (A[0])
  - så er det næste-mindste element i arrayet A
  - det mindste elemente i del-arrayet A[1,...,n-1]
- Dvs. skal bare kunne finde det mindste element i et array

Udvælgelsessortering

Udvælgelsessortering

- Find det mindste element i et array A[0,...,n-1]

Udvælgelsessortering

- Find det mindste element i et array A[0,...,n-1]
-
```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

Udvælgelsessortering

- Find det mindste element i et array A[0,...,n-1]
-
```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Find det mindste element i et del-array A[i,...,n-1]

Udvælgelsessortering

- Find det mindste element i et array A[0,...,n-1]
-
```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Find det mindste element i et del-array A[i,...,n-1]
-
```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

Udvælgelsessortering

Udvælgelsessortering

- Algoritme

Udvælgelsessortering

- Algoritme
-

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

Udvælgelsessortering

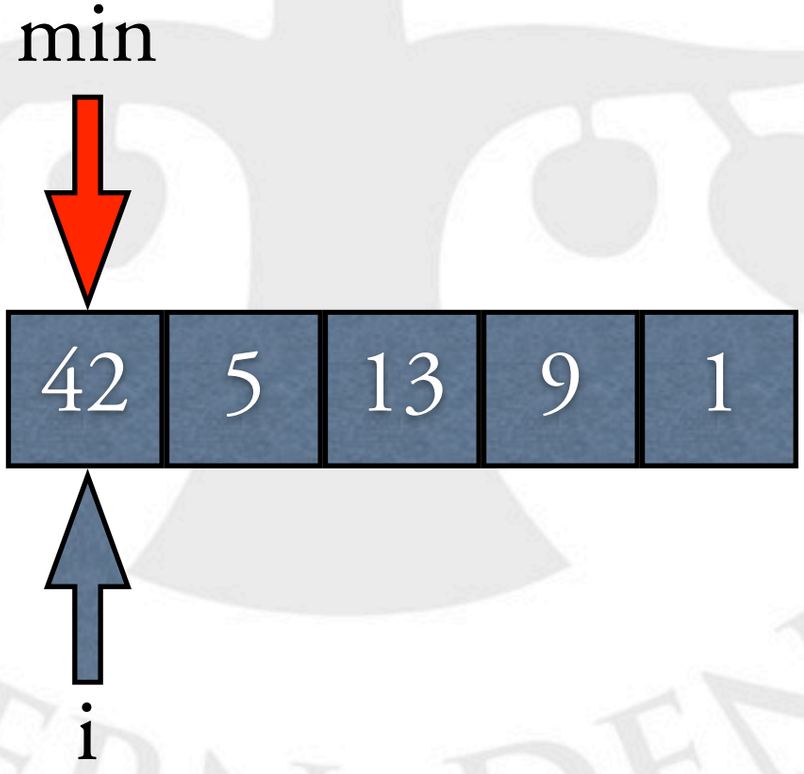
- Algoritme
-

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

Udvælgelsessortering

- Algoritme
-

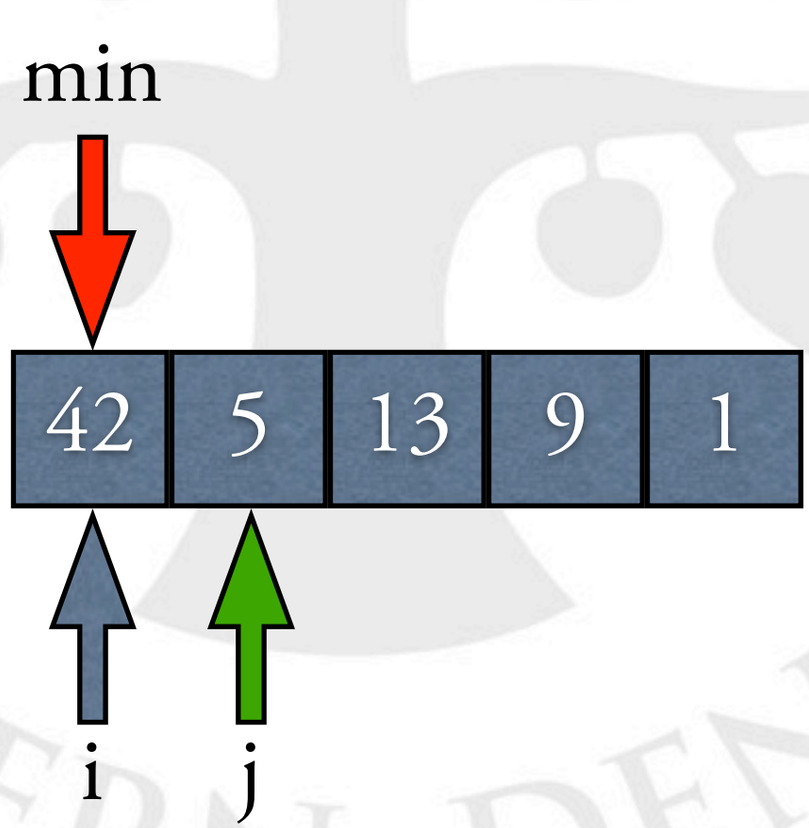
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

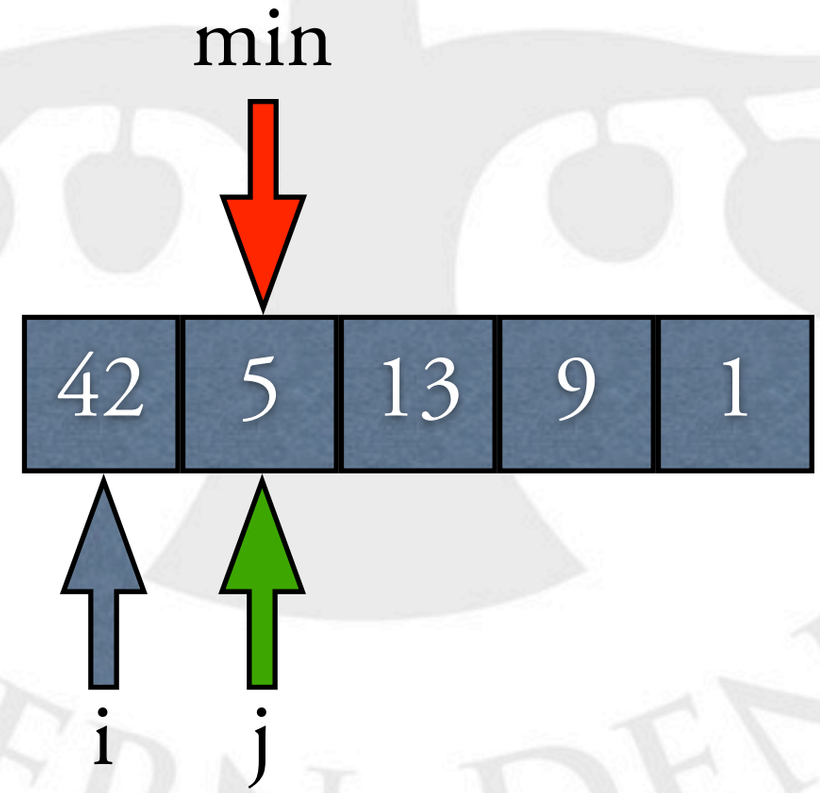
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

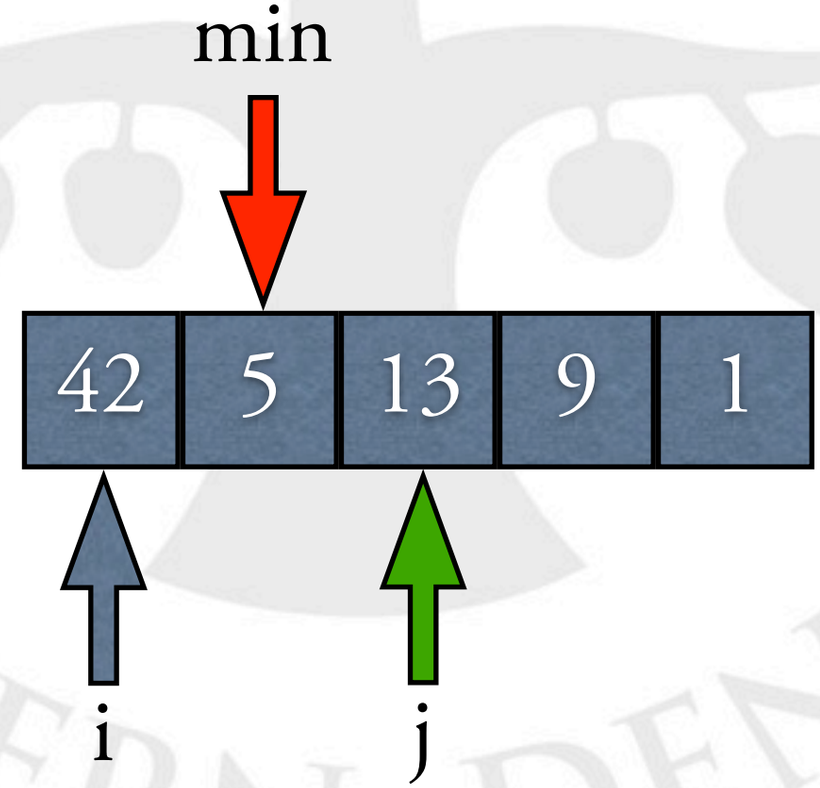
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

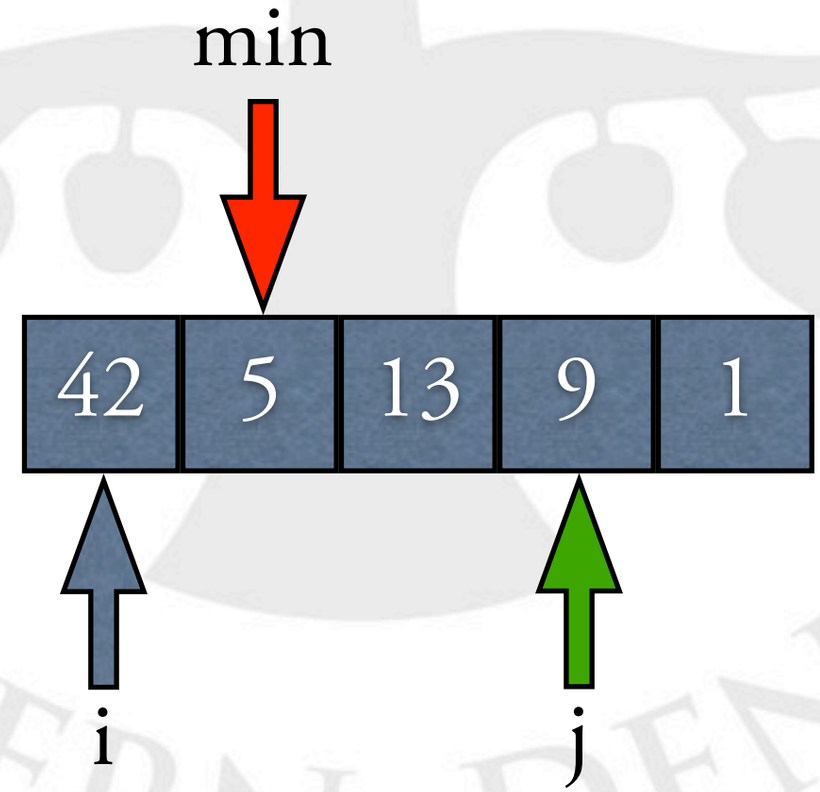
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

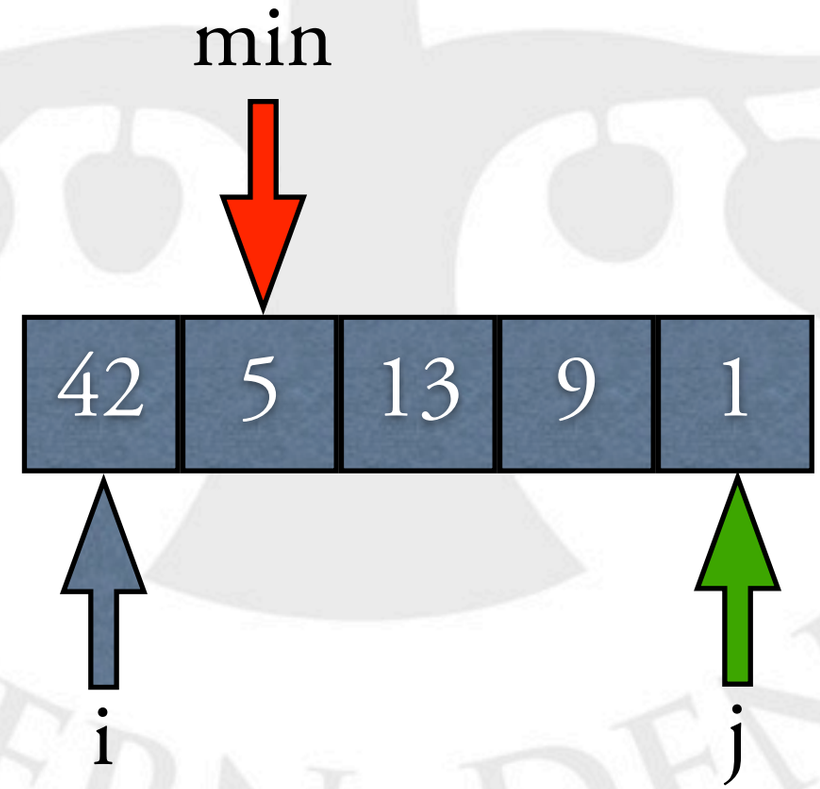
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

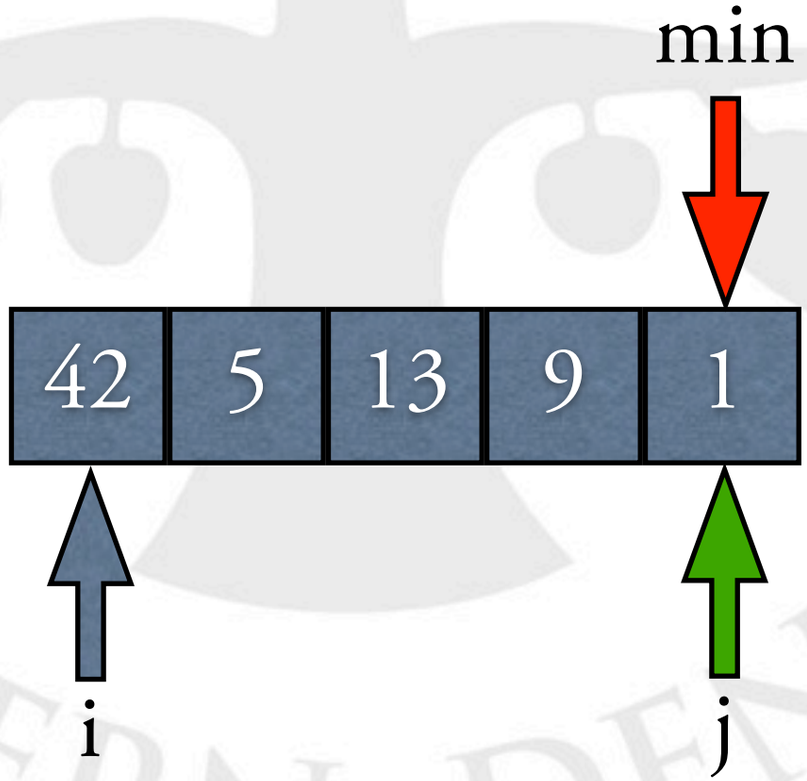
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

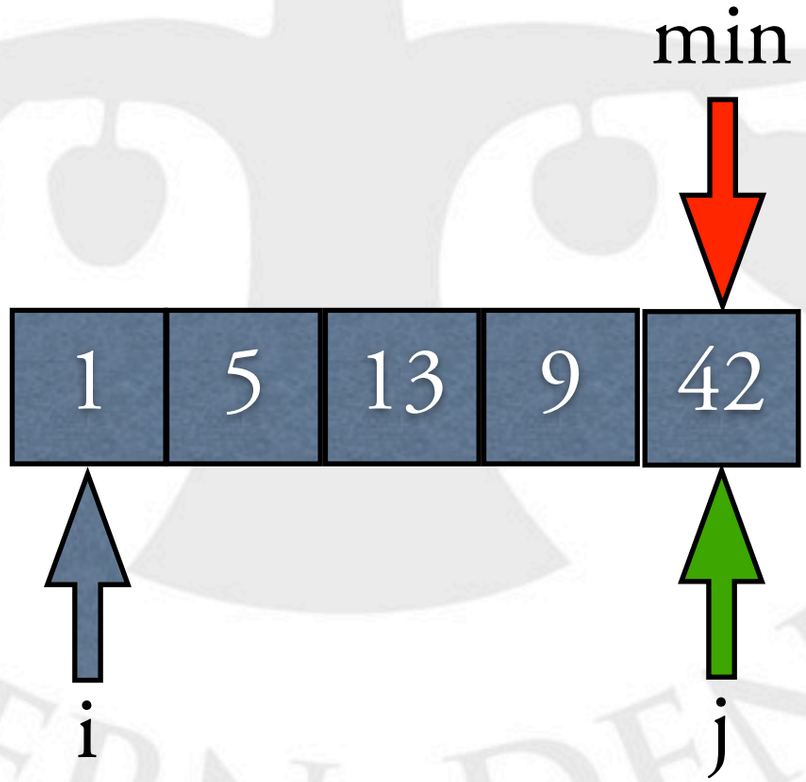
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

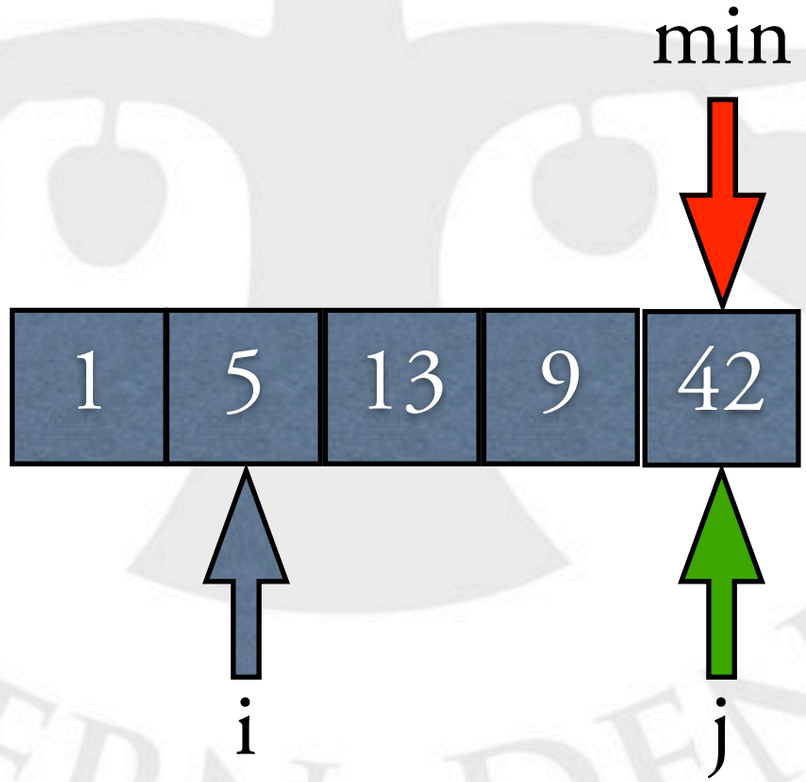
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

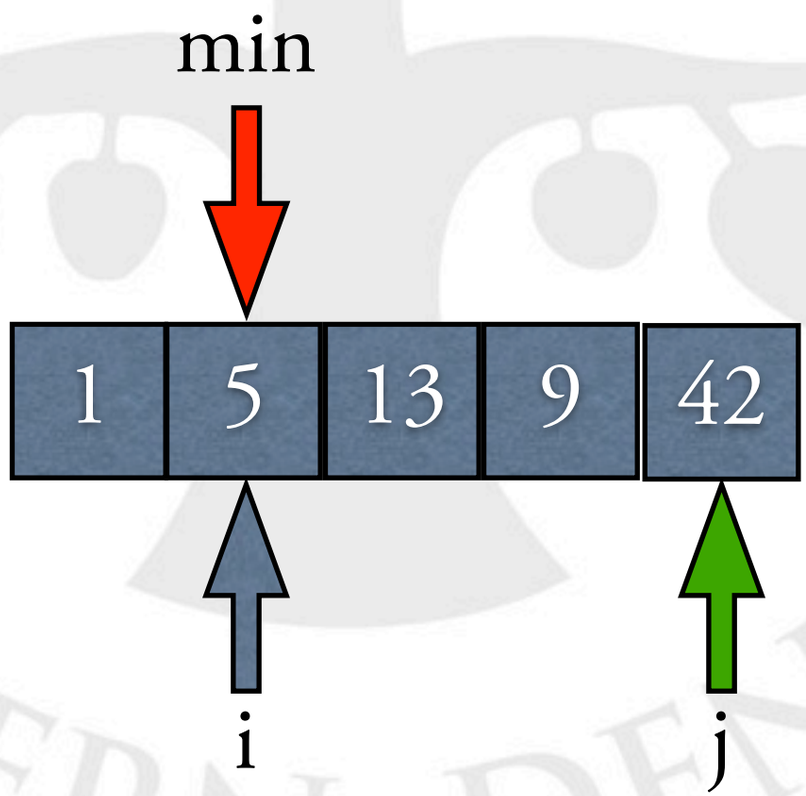
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

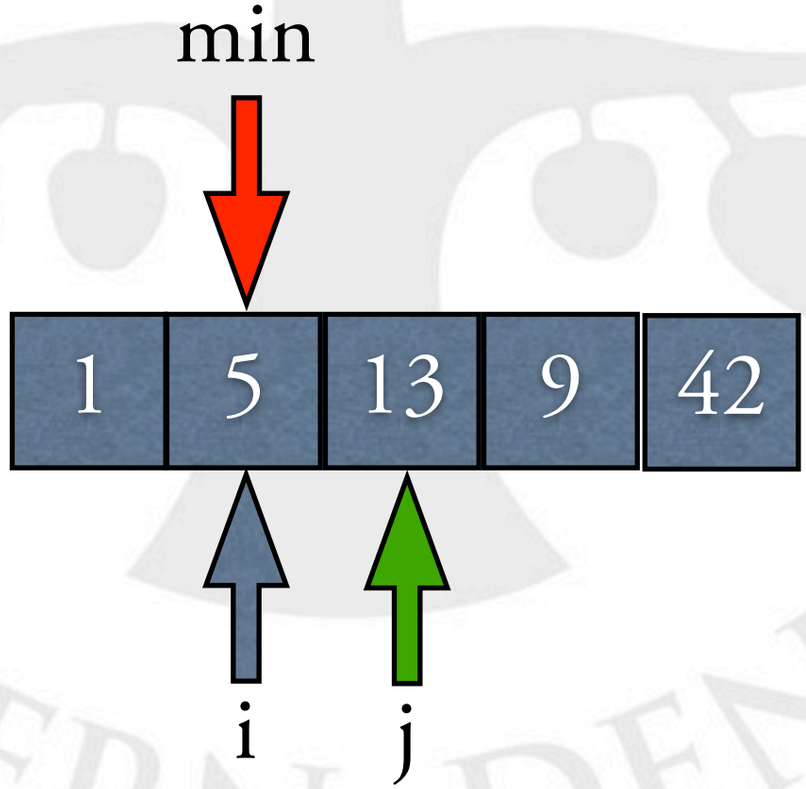
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

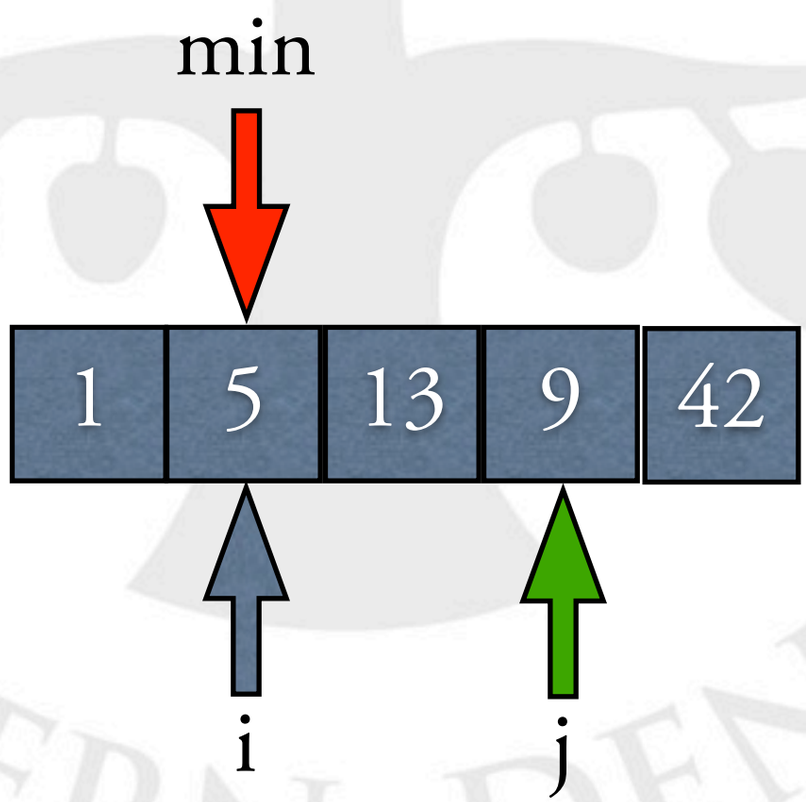
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

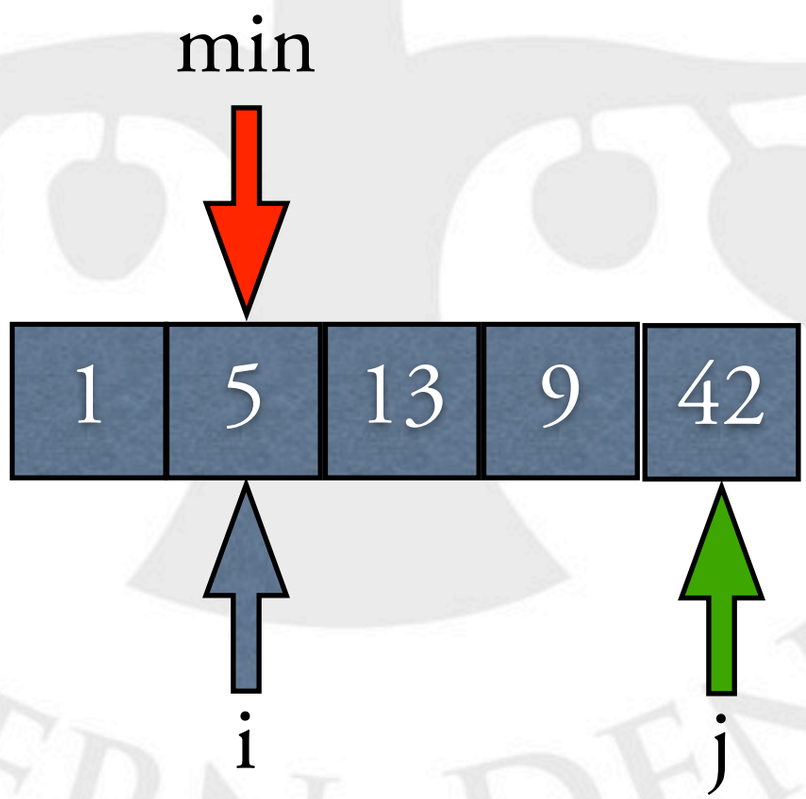
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

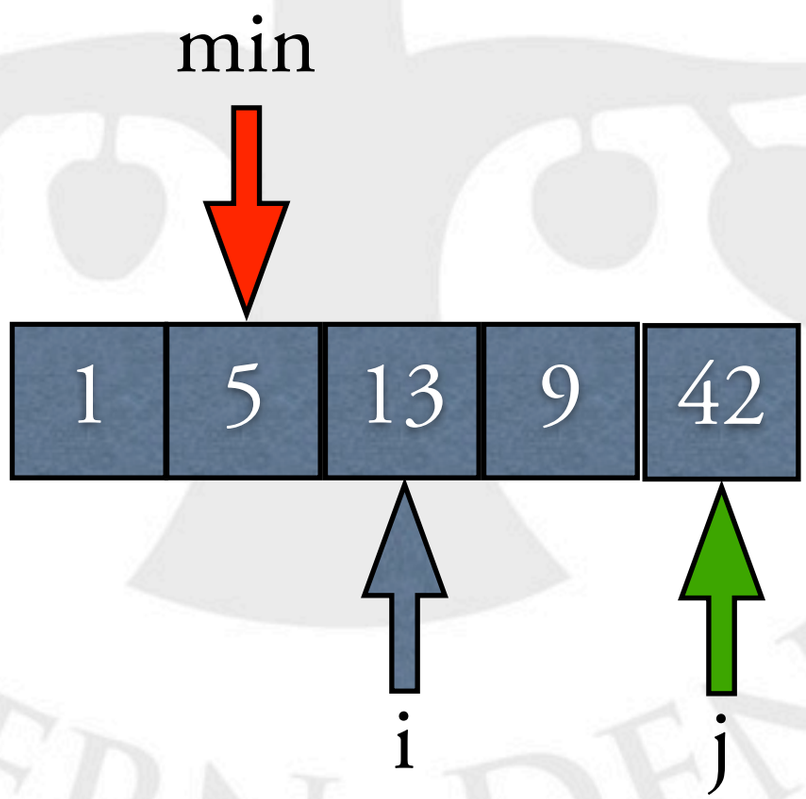
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

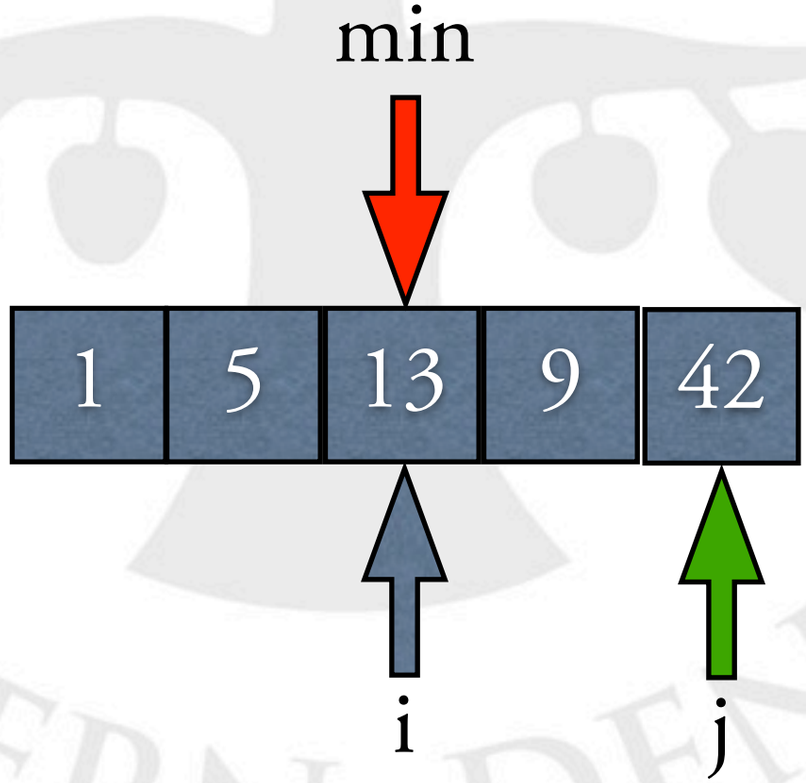
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

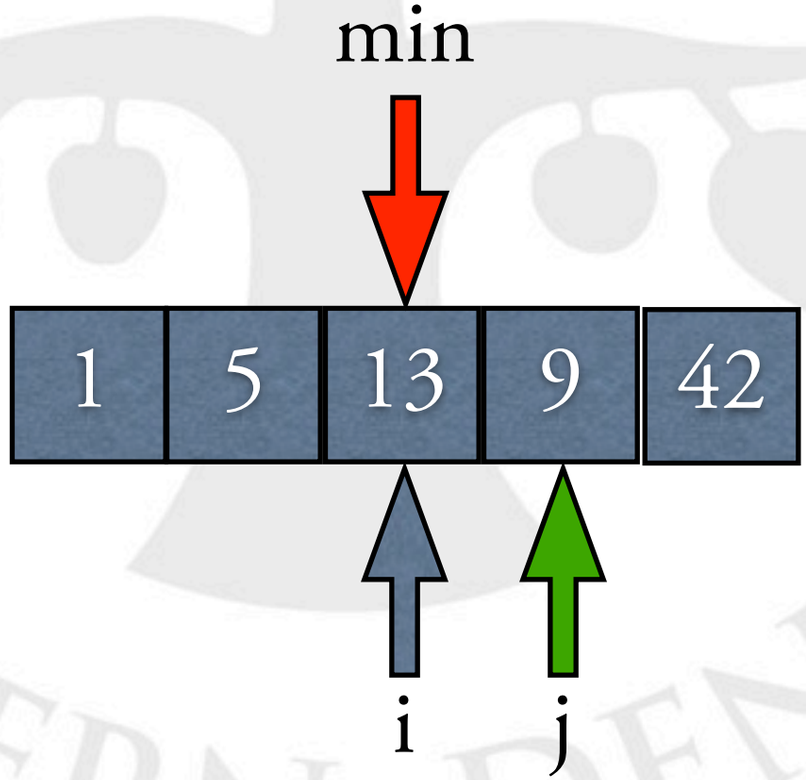
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

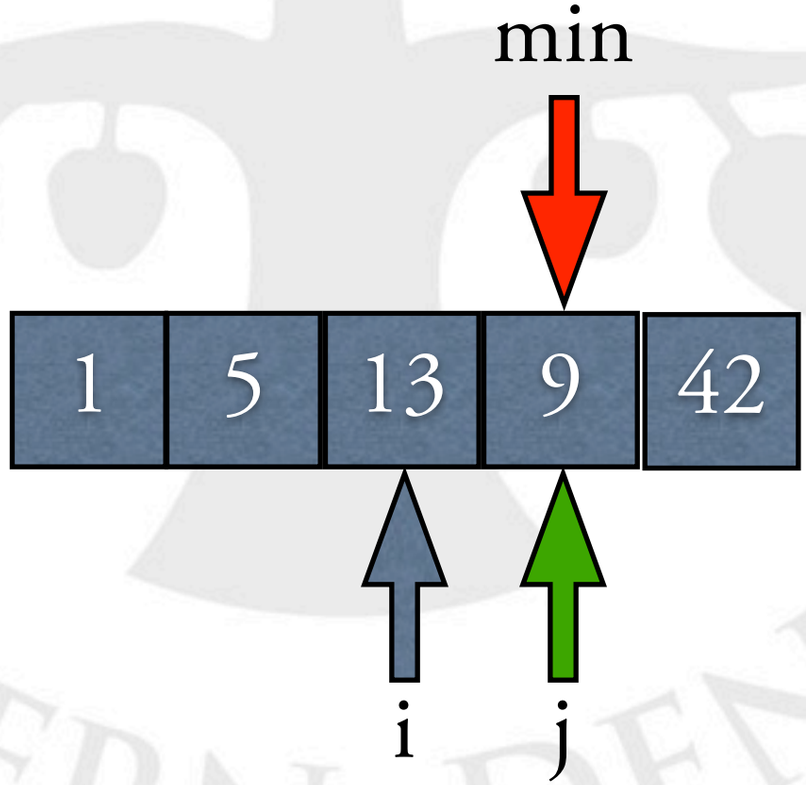
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

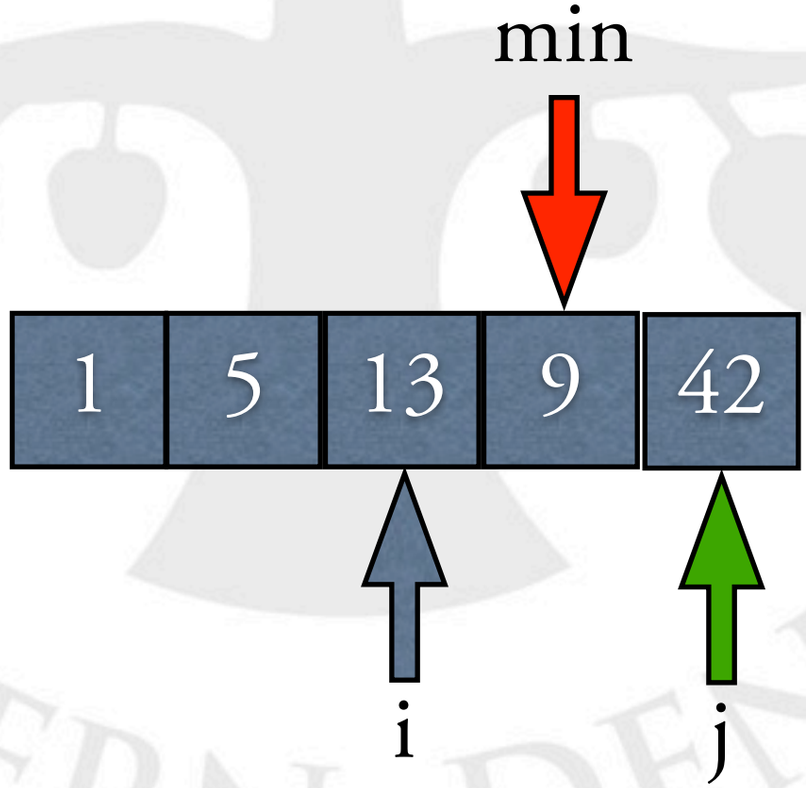
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

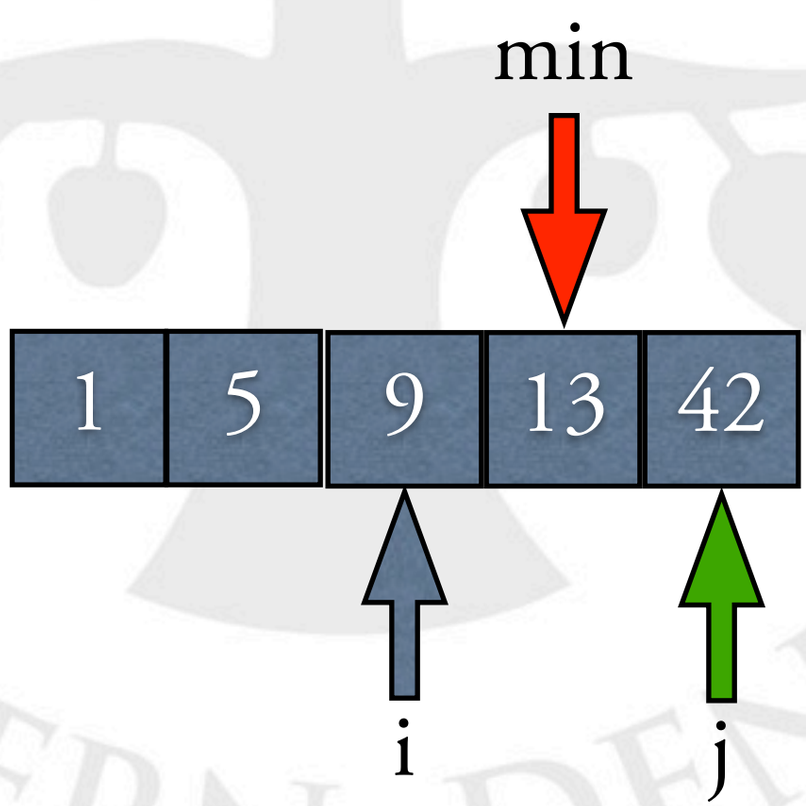
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

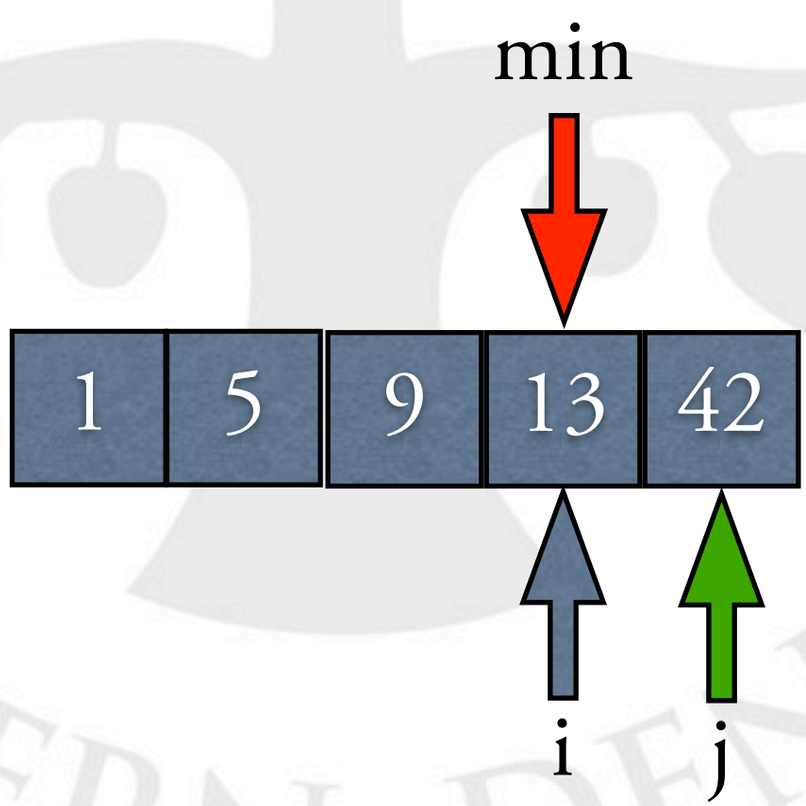
```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

- Algoritme
-

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```



Udvælgelsessortering

Udvælgelsessortering

- Analyse

Udvælgelsessortering

- Analyse
  - Karakteristisk opperation?

Udvælgelsessortering

- Analyse
  - Karakteristisk opperation?
    - Sammenligning af to indgange i arrayet

Udvælgelsessortering

- Analyse
  - Karakteristisk opperation?
    - Sammenligning af to indgange i arrayet
  - Størrelsen af input?

Udvælgelsessortering

- Analyse
  - Karakteristisk opperation?
    - Sammenligning af to indgange i arrayet
  - Størrelsen af input?
    - Antallet af elementer i arrayet, dvs. n

Udvælgelsessortering

Udvælgelsessortering

- Lad os starte helt simpelt

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?
  - A[1] < A[min]

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?
  - A[1] < A[min]
  - A[2] < A[min]

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?
  - A[1] < A[min]
  - A[2] < A[min]
  - ...

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?
  - A[1] < A[min]
  - A[2] < A[min]
  - ...
  - A[n-1] < A[min]

Udvælgelsessortering

- Lad os starte helt simpelt
- Find det mindste element i A[0,...,n-1]

```
min = 0
for j = 1 to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Kun et sted sammenlignes to elementer
- Hvor mange gange bliver den sammenligning udført?
  - A[1] < A[min]
  - A[2] < A[min]
  - ...
  - A[n-1] < A[min]
- I alt n-1 sammenligninger af to elementer

Udvælgelsessortering

Udvælgelsessortering

- Lidt mere kompliceret

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?
- A[i+1] < A[min]

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?
  - A[i+1] < A[min]
  - A[i+2] < A[min]

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?
  - A[i+1] < A[min]
  - A[i+2] < A[min]
  - ...

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?
  - A[i+1] < A[min]
  - A[i+2] < A[min]
  - ...
  - A[n-1] < A[min]

Udvælgelsessortering

- Lidt mere kompliceret
- Find det mindste element i A[i,...,n-1]

```
min = i
for j = (i + 1) to n-1 do:
    if A[j] < A[min]:
        min = j
```

- Hvor mange gange bliver den sammenligning udført?
  - A[i+1] < A[min]
  - A[i+2] < A[min]
  - ...
  - A[n-1] < A[min]
- I alt (n-1)-(i+1)+1 = n-i-1 sammenligninger

Udvælgelsessortering

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...

Udvælgelsessortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    min = i
    for j = (i + 1) to n-1 do:
        if A[j] < A[min]:
            min = j
    swap A[i] and A[min]
```

- Bemærk at "find det mindste element i A[i,...,n-1]" udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...
  - i = n-2: n-i-1 = 1 sammenligning

Udvælgelsessortering

Udvælgelsessortering

- Totale antal sammenligninger

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 =$

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 =$

$$\sum_{m=1}^{n-1} m = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 = \sum_{m=1}^{n-1} m = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$
  - Det er vores resultat

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 = \sum_{m=1}^{n-1} m = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$
  - Det er vores resultat
  - Kan sammenlignes med funktioner for andre algoritmer

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 = \sum_{m=1}^{n-1} m = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$
  - Det er vores resultat
  - Kan sammenlignes med funktioner for andre algoritmer
  - Kan også sige noget direkte om køretiden af en implementering

Udvælgelsessortering

- Totale antal sammenligninger
  - Summen af alle sammenligningerne ovenfor
  - $1 + 2 + 3 + \cdots + n - 1 = \sum_{m=1}^{n-1} m = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$
  - Det er vores resultat
  - Kan sammenlignes med funktioner for andre algoritmer
  - Kan også sige noget direkte om køretiden af en implementering
    - En fordobling af antallet af elementer vil ca. give en firdobling af køretiden

Boble-sortering

Boble-sortering

- Idé

Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

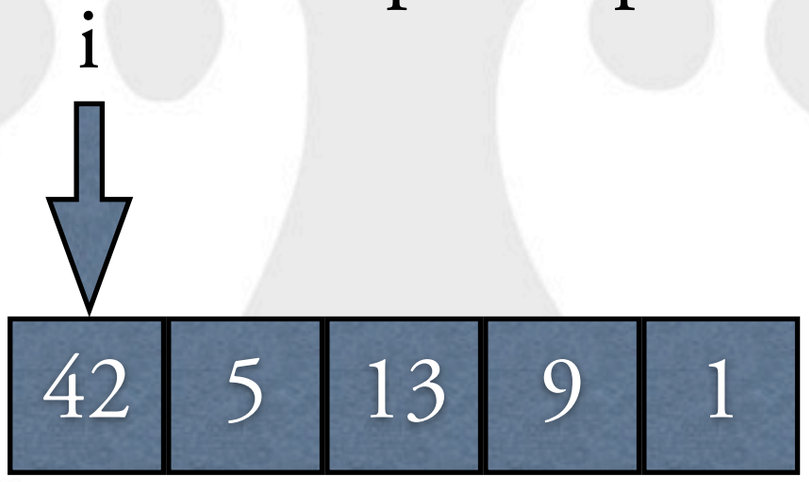
  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

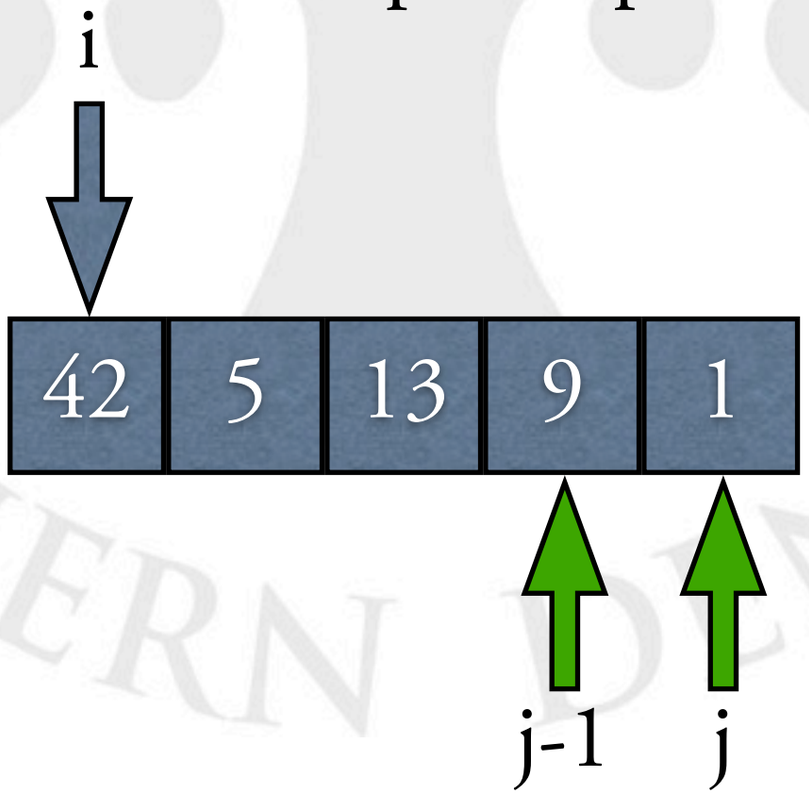


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

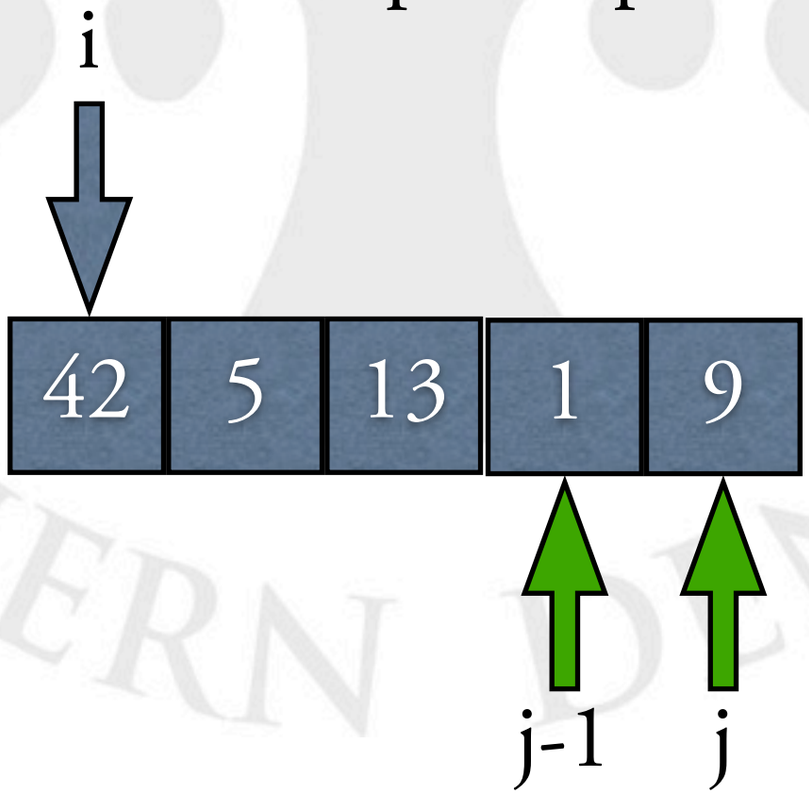


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

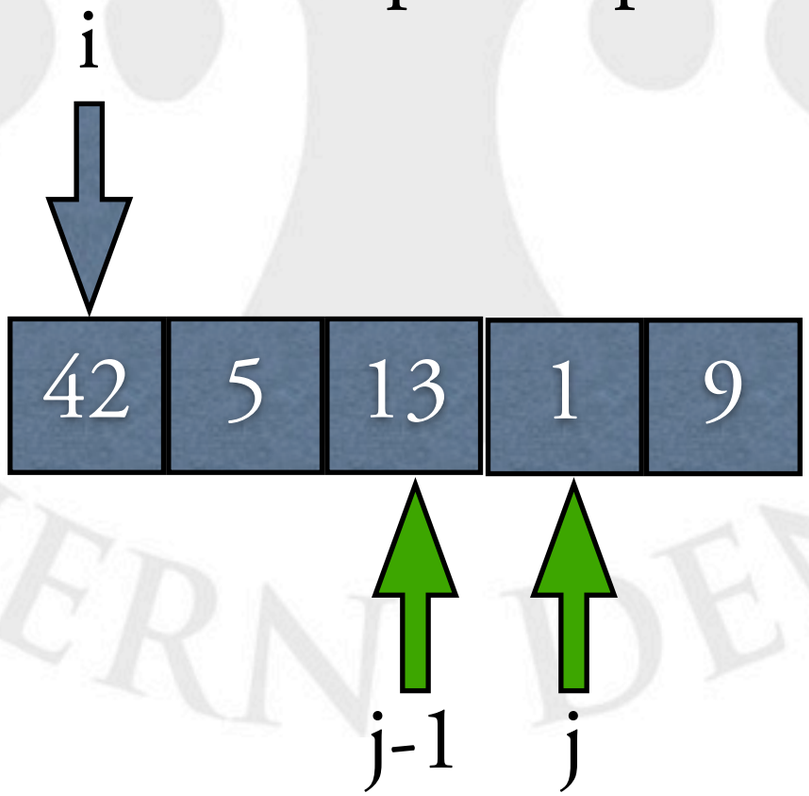


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

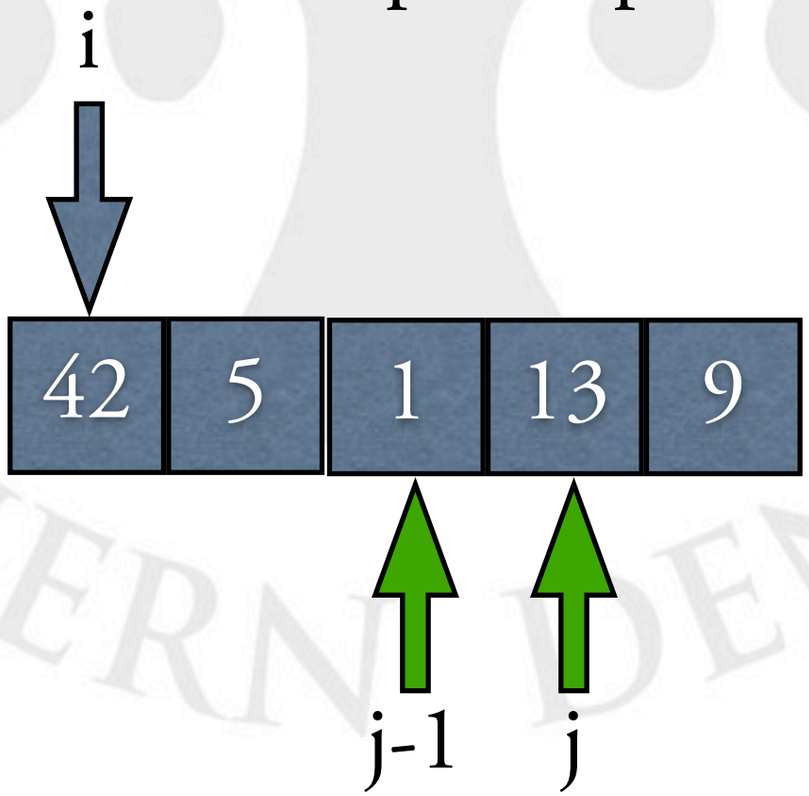


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

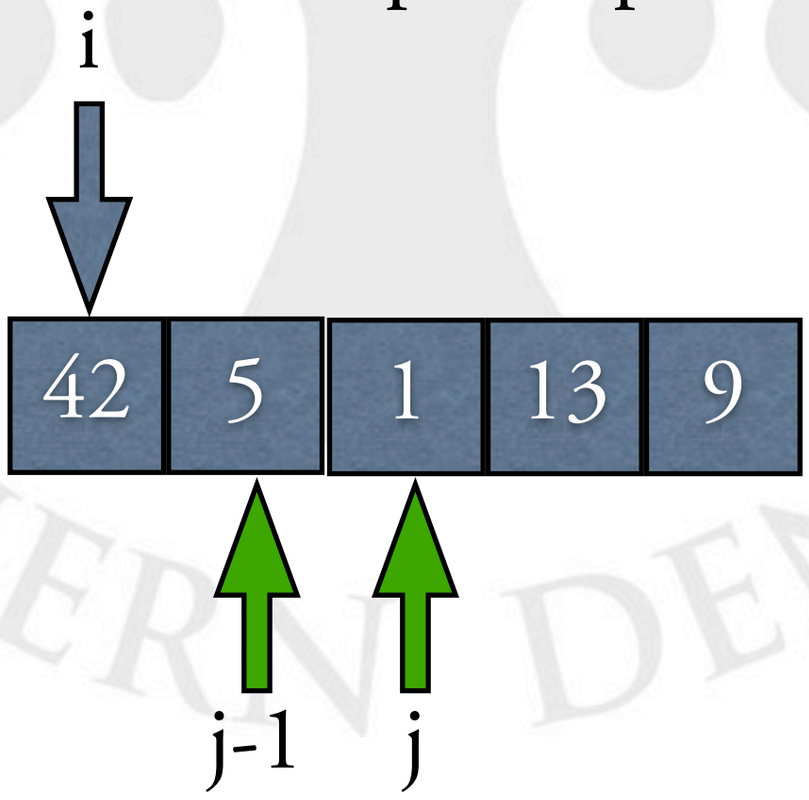


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

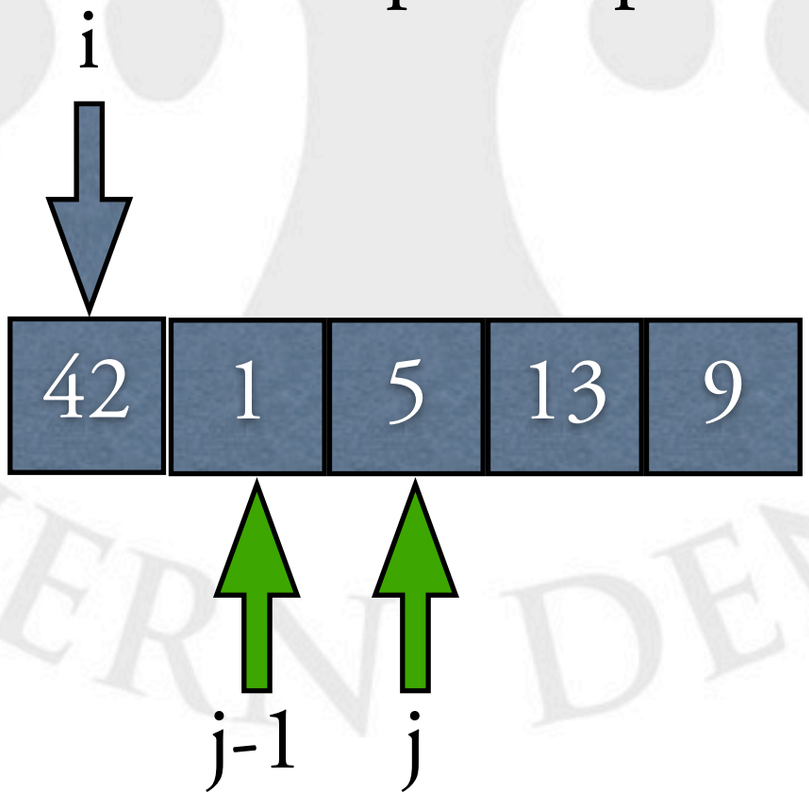


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

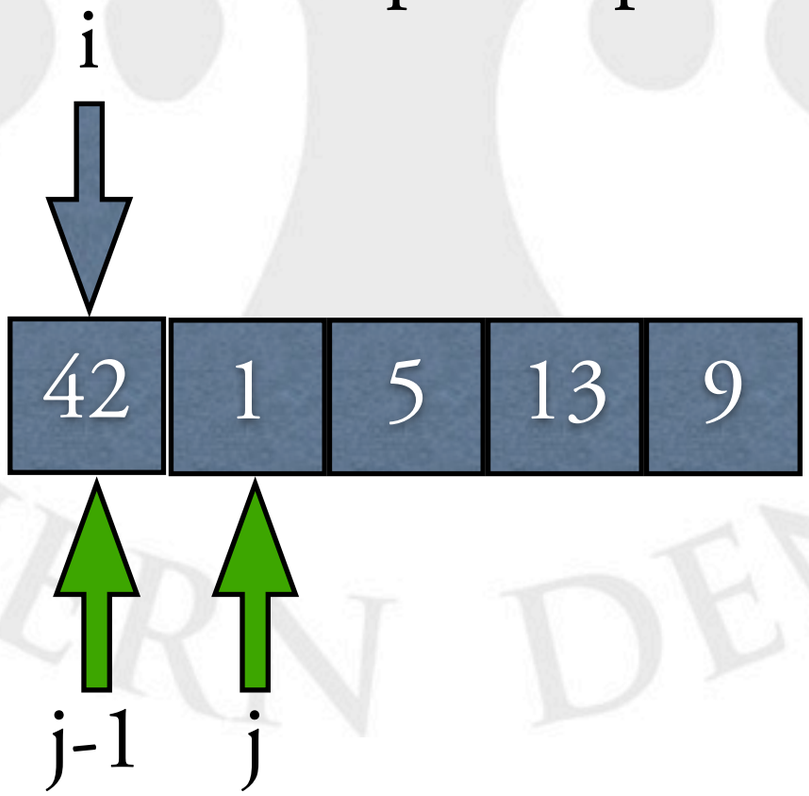


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

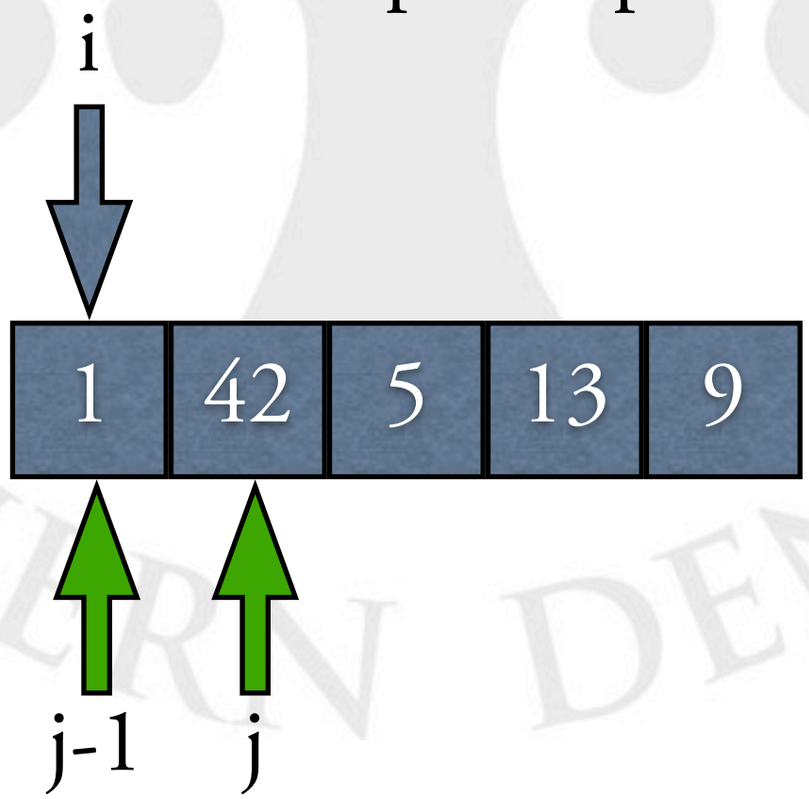


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

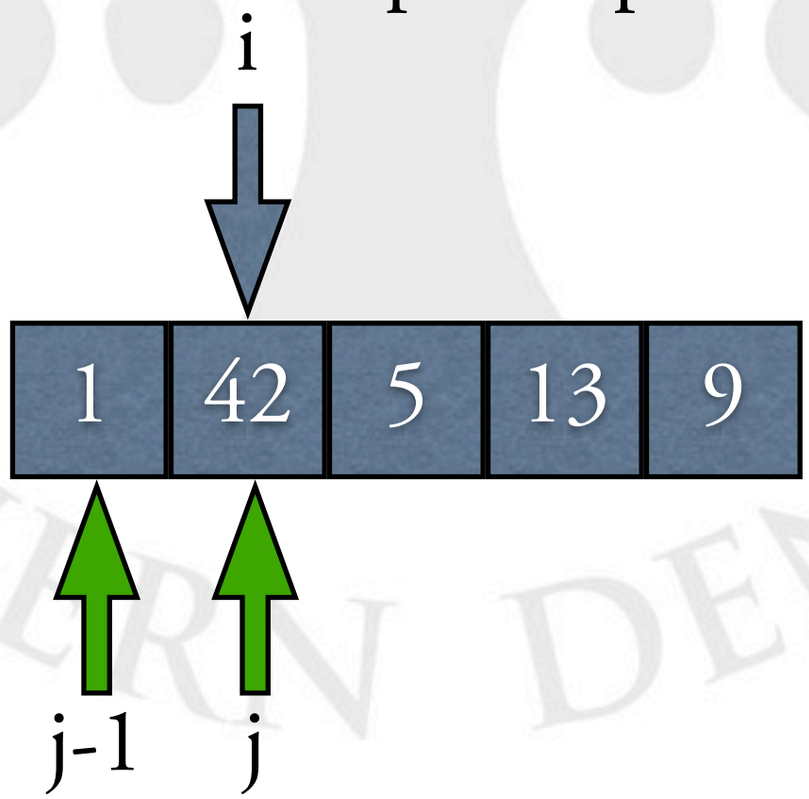


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

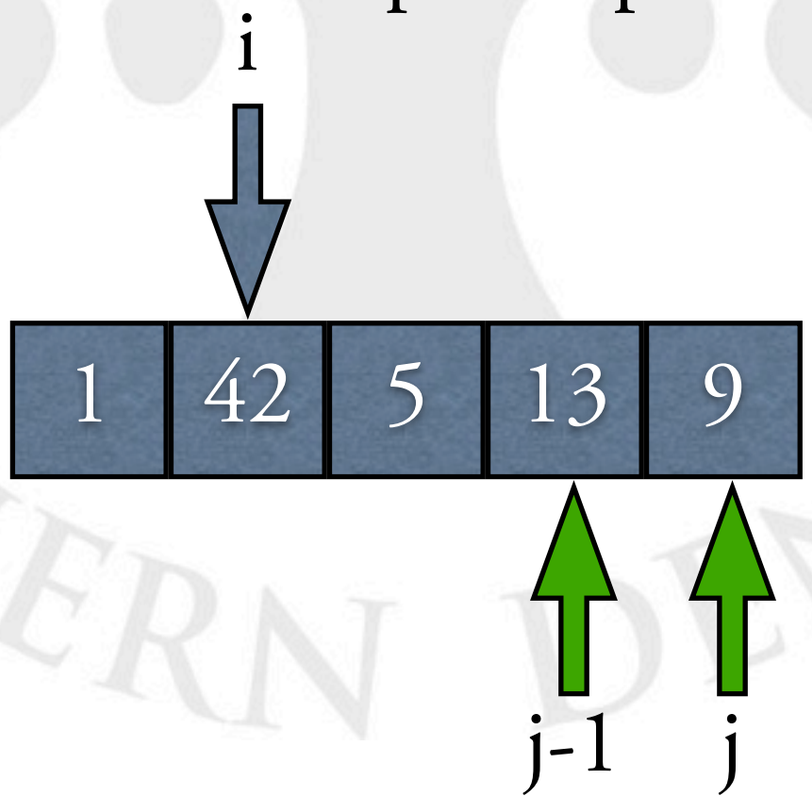


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

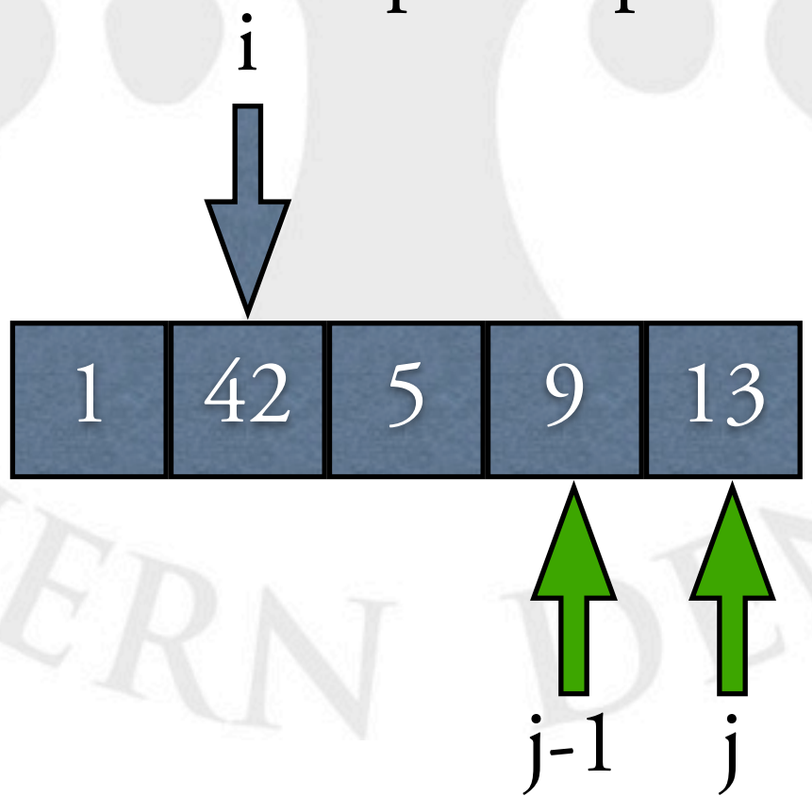


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

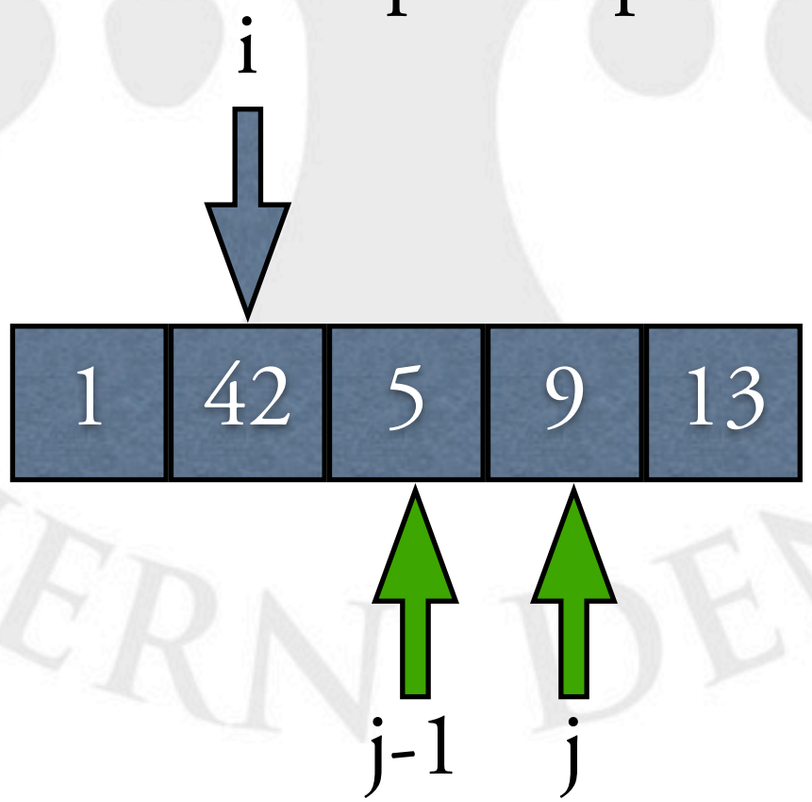


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

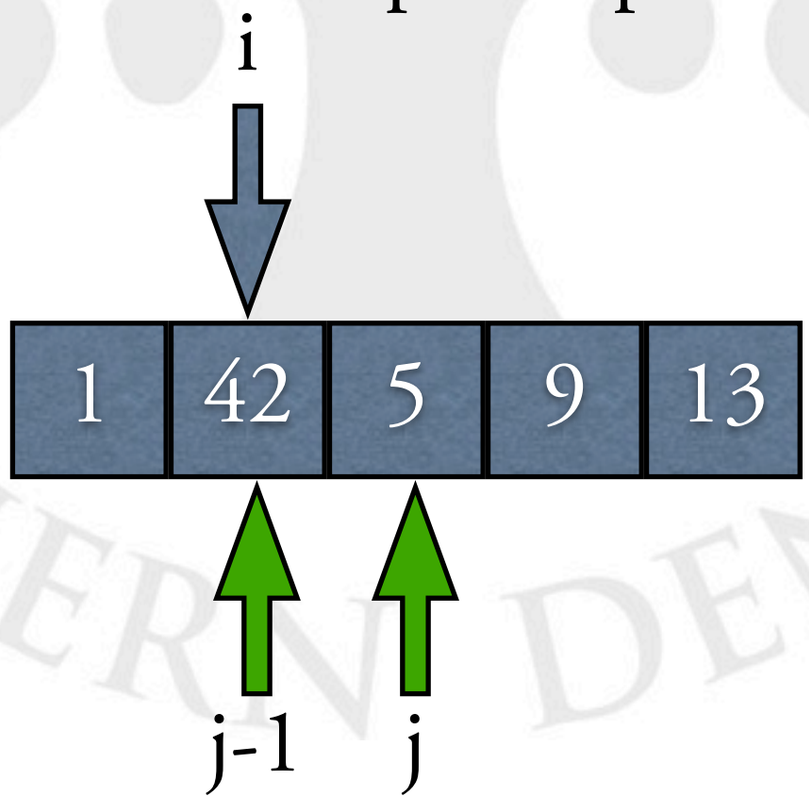


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

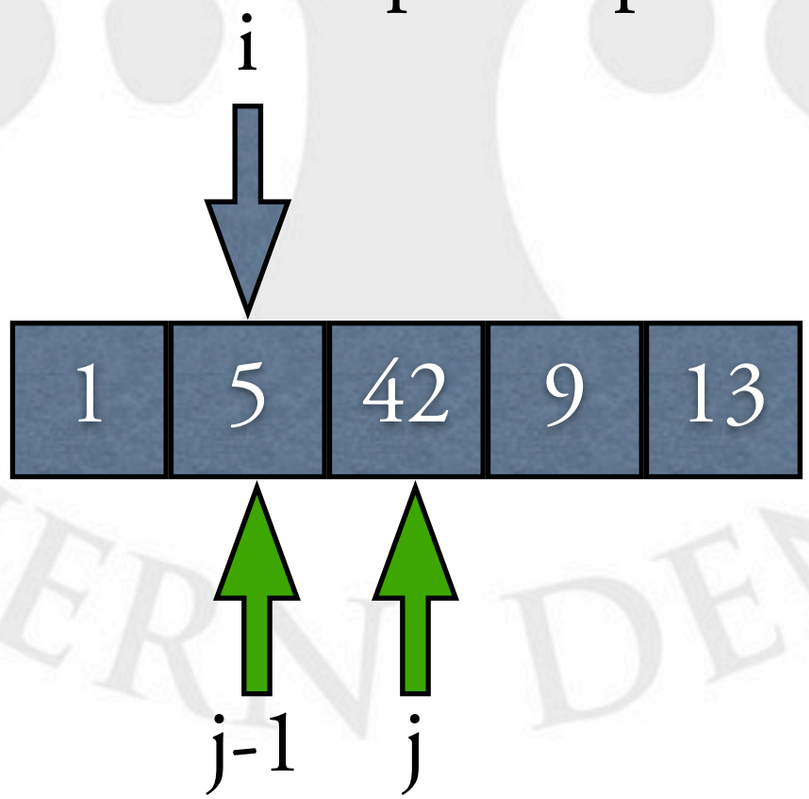


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

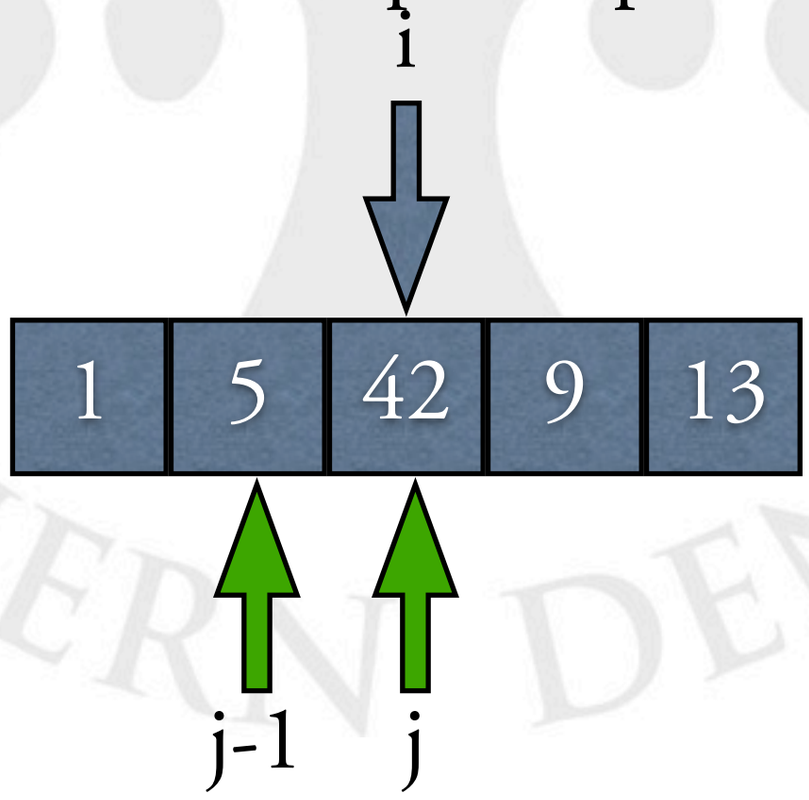


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

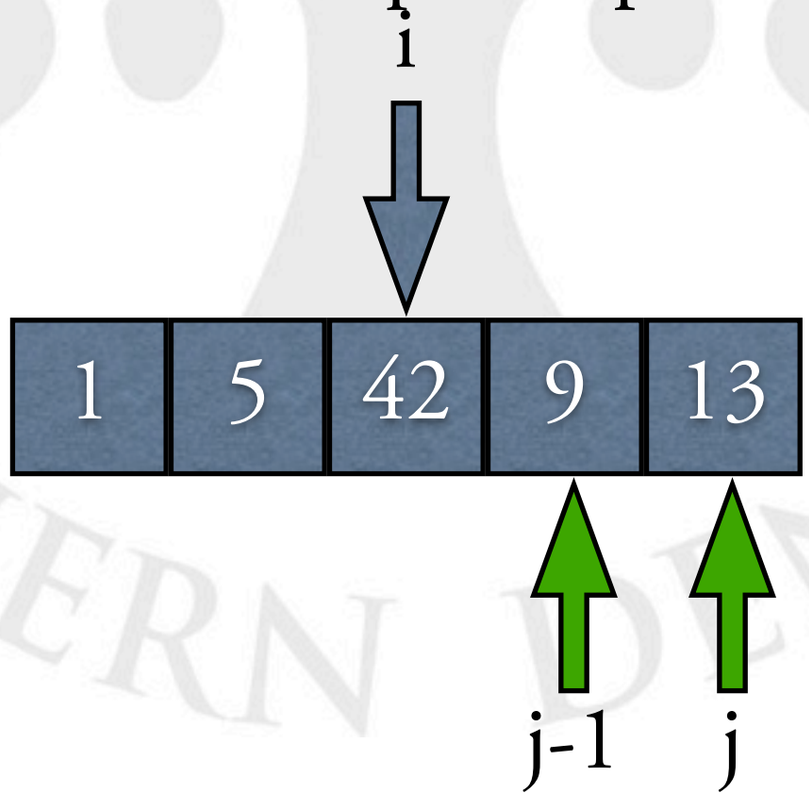


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

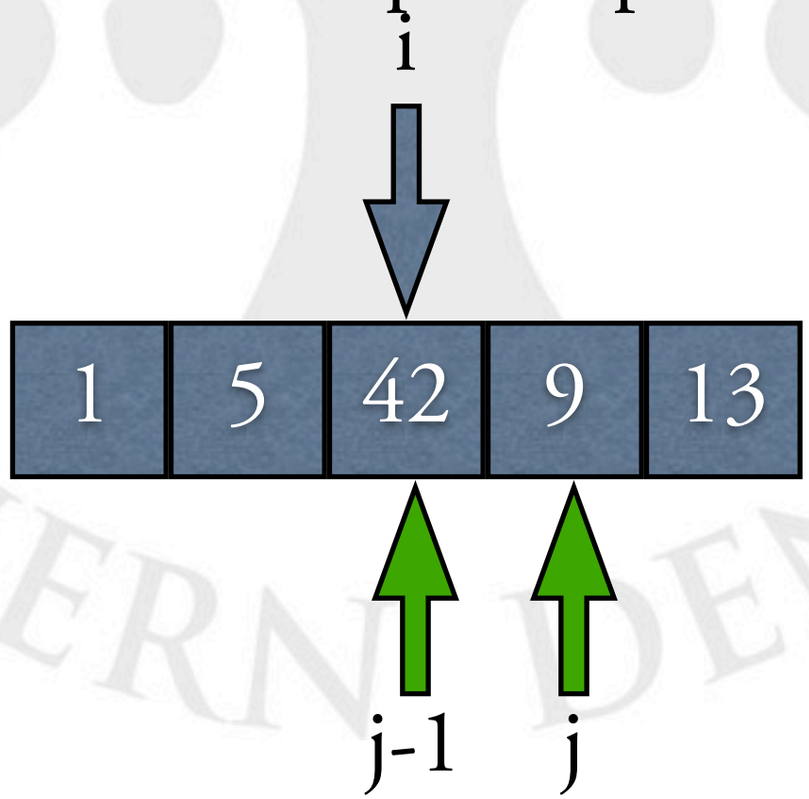


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

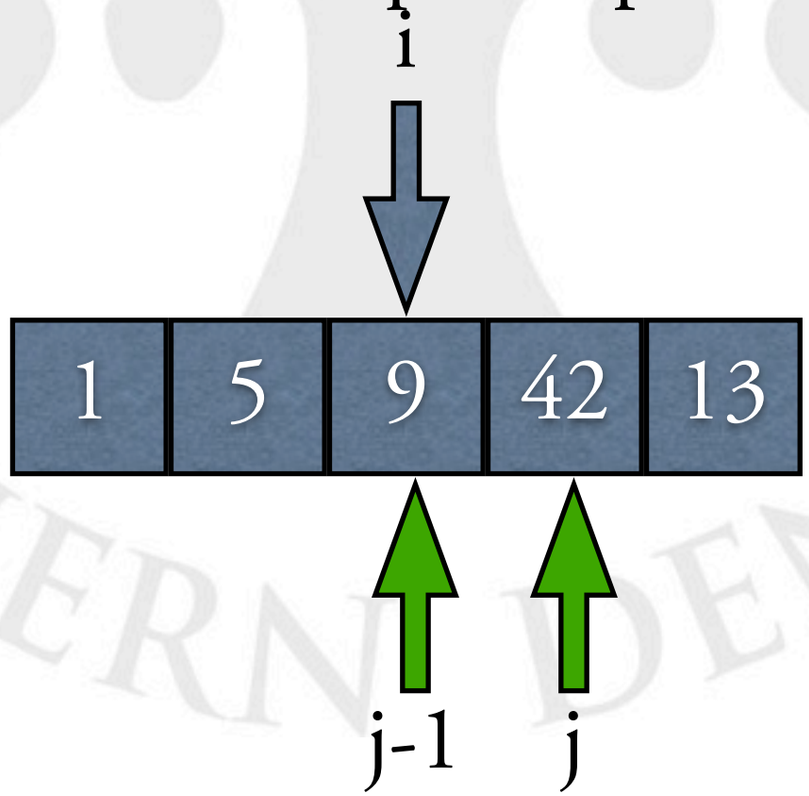


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

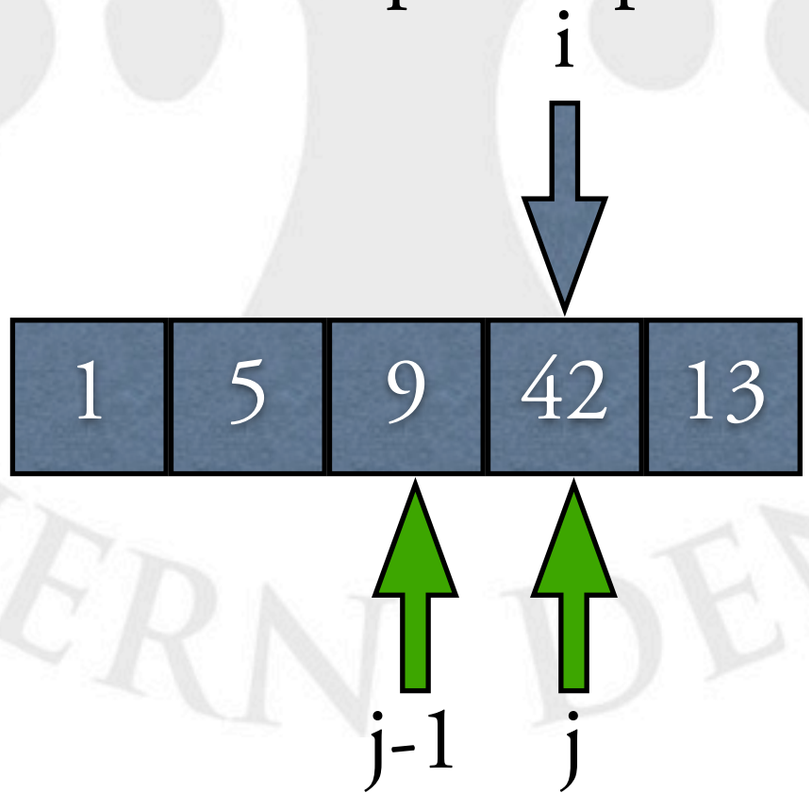


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

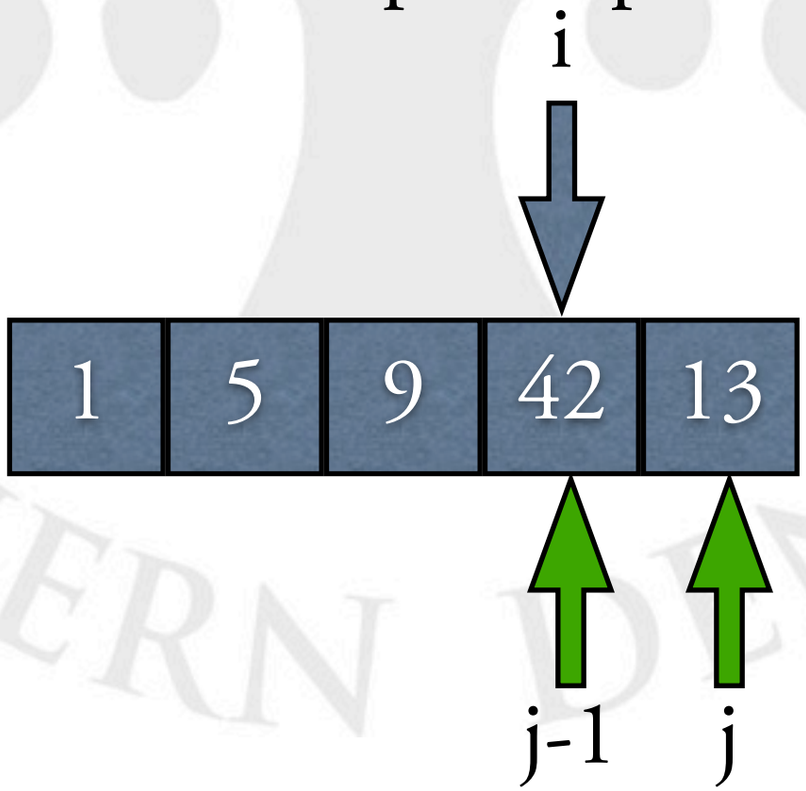


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads

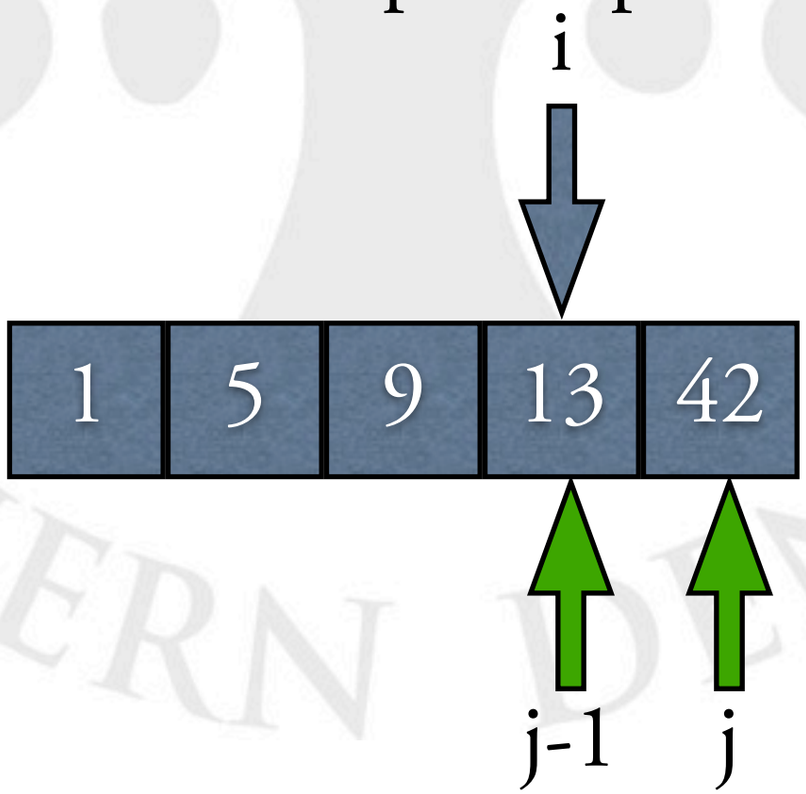


Boble-sortering

- Idé
  - Løb arrayet igennem og løbende ombyt naboelementer som står i den forkerte rækkefølge

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

  - Bemærk at for hver værdi af i, bliver det i'te-største element "boblet" ned på sin plads



Boble-sortering

Boble-sortering

- Analyse

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?
- A[n-1] < A[n-2]

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?
  - A[n-1] < A[n-2]
  - A[n-2] < A[n-3]

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?
- A[n-1] < A[n-2]
- A[n-2] < A[n-3]
- …

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?
  - A[n-1] < A[n-2]
  - A[n-2] < A[n-3]
  - ...
  - A[i+1] < A[i]

Boble-sortering

- Analyse
- Kig først på den inderste for-løkke

```
for j = n-1 downto i+1 do:
    if A[j] < A[j-1]:
        swap A[j] and A[j-1]
```

- Hvor mange gange bliver der sammenlignet?
  - A[n-1] < A[n-2]
  - A[n-2] < A[n-3]
  - ...
  - A[i+1] < A[i]
- I alt (n-1)-(i+1)+1 = n-i-1 sammenligninger

Boble-sortering

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...
  - i = n-2: n-i-1 = 1 sammenligning

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...
  - i = n-2: n-i-1 = 1 sammenligning
- Præcis samme udregning som for udvælgelsessortering

Boble-sortering

- Hele algoritmen

```
for i = 0 to n-2 do:
    for j = n-1 downto i+1 do:
        if A[j] < A[j-1]:
            swap A[j] and A[j-1]
```

- Bemærk at den inderste for-løkke udføres én gang for hvert i = 0,1,2,...,n-2
  - i = 0: n-i-1 = n-1 sammenligninger
  - i = 1: n-i-1 = n-2 sammenligninger
  - i = 2: n-i-1 = n-3 sammenligninger
  - ...
  - i = n-2: n-i-1 = 1 sammenligning
- Præcis samme udregning som for udvælgelsessortering
- Resultat $\frac{1}{2}n^2 - \frac{1}{2}n$

Indsættelsessortering

Indsættelsessortering

- Idé

Indsættelsessortering

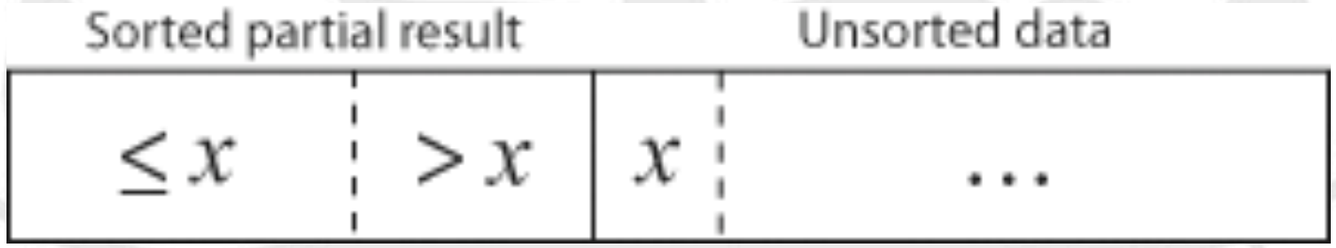
- Idé
  - Opbyg et sorteret del-array, et element af gangen

Indsættelsessortering

- Idé
  - Opbyg et sorteret del-array, et element af gangen
  - Indsæt det næste element i den sorterede del, ved at flytte større elementer en plads til højre

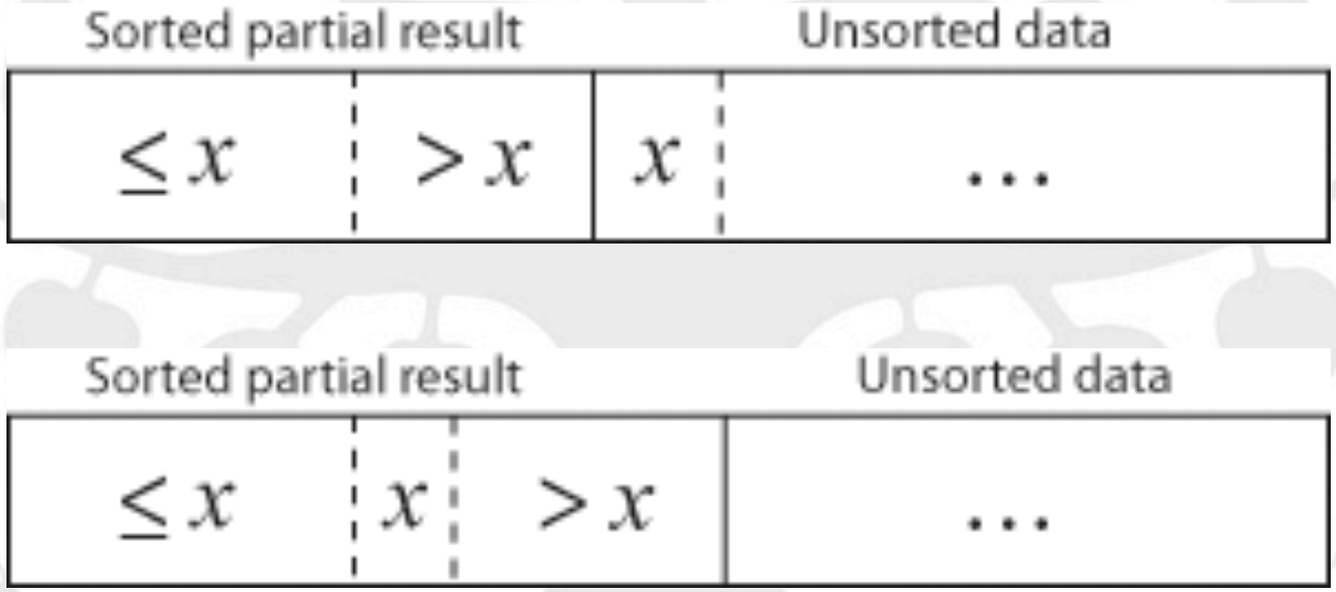
Indsættelsessortering

- Idé
  - Opbyg et sorteret del-array, et element af gangen
  - Indsæt det næste element i den sorterede del, ved at flytte større elementer en plads til højre



Indsættelsessortering

- Idé
  - Opbyg et sorteret del-array, et element af gangen
  - Indsæt det næste element i den sorterede del, ved at flytte større elementer en plads til højre



Indsættelsessortering

Indsættelsessortering

- Algoritme

```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```

Indsættelsessortering

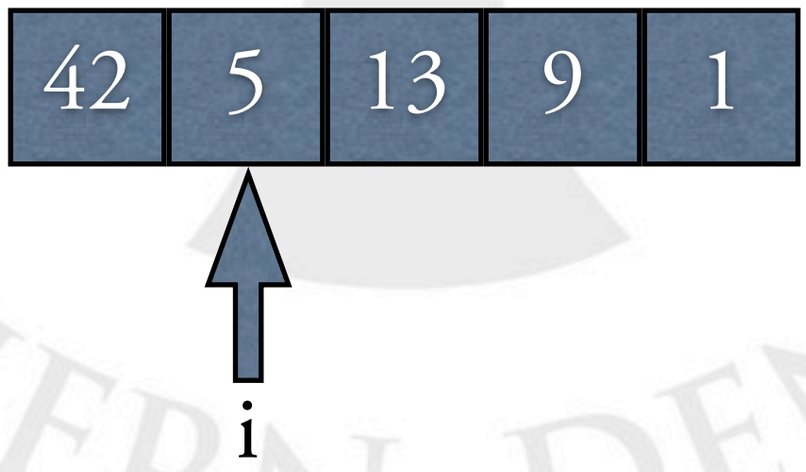
- Algoritme

```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```

Indsættelsessortering

- Algoritme

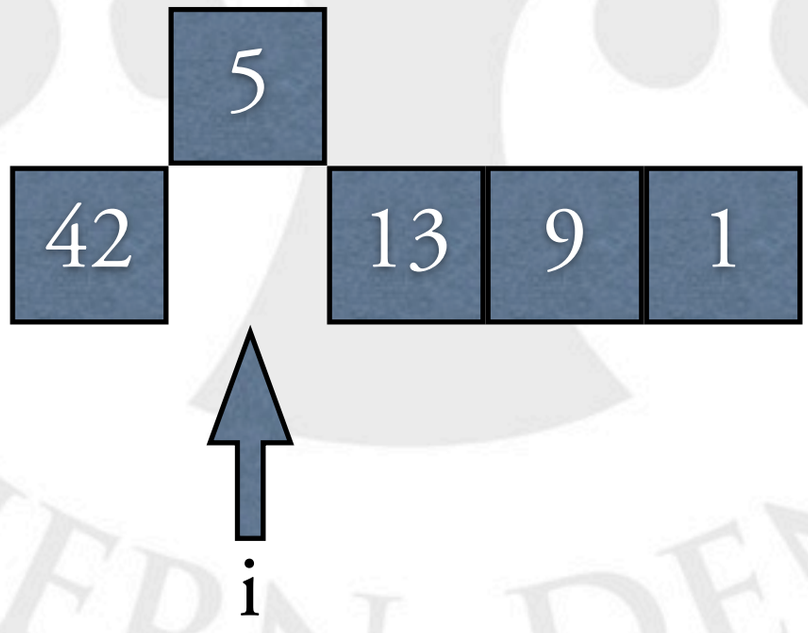
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

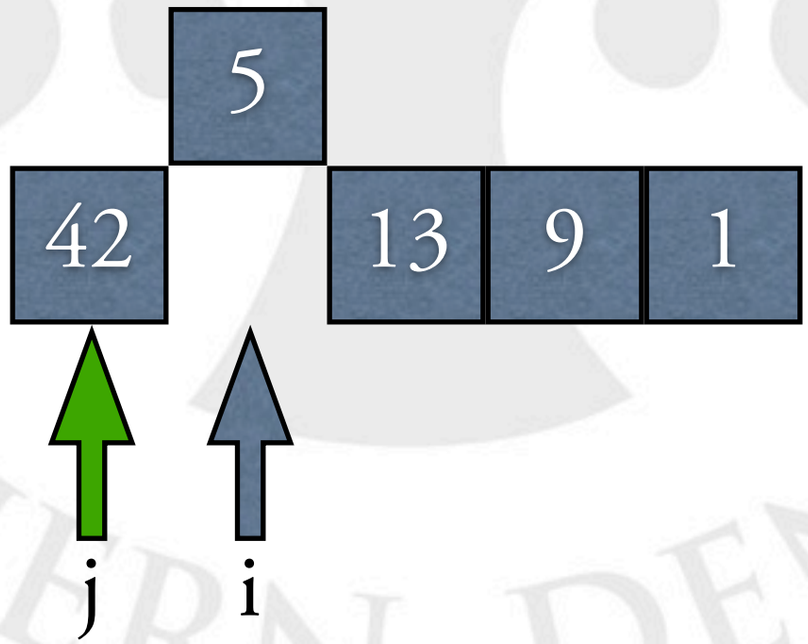
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

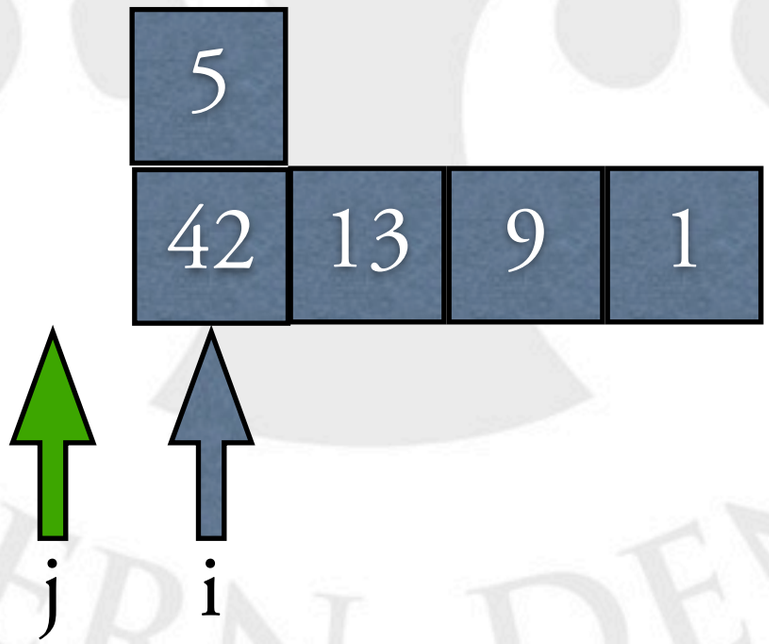
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

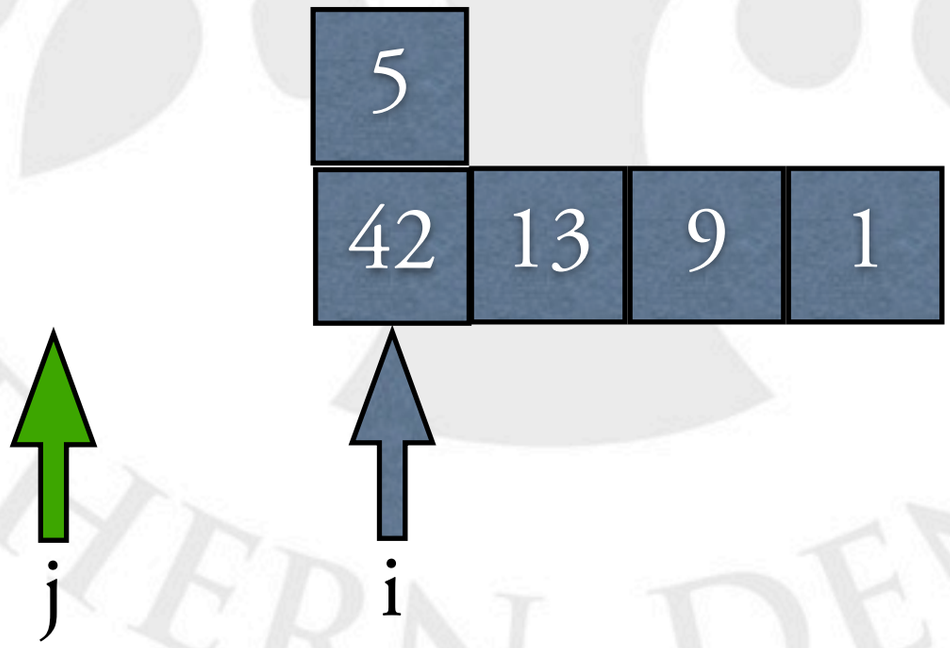
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

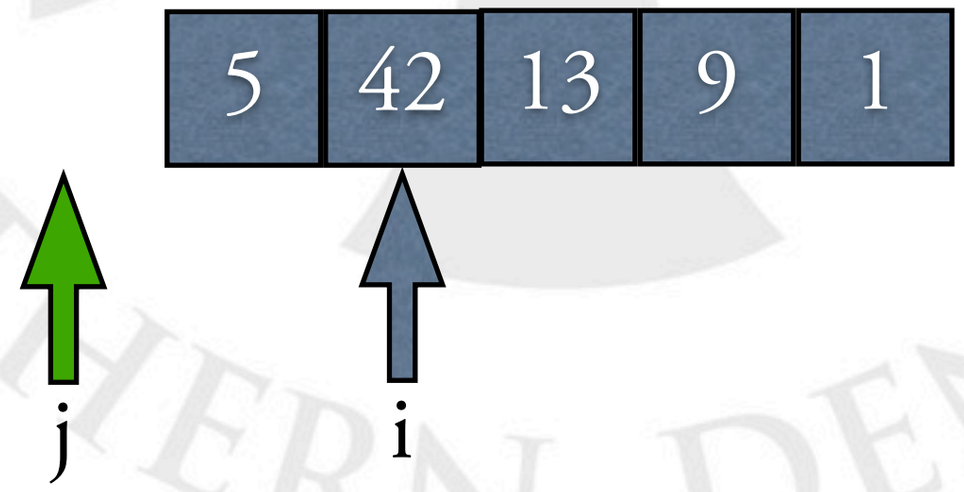
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

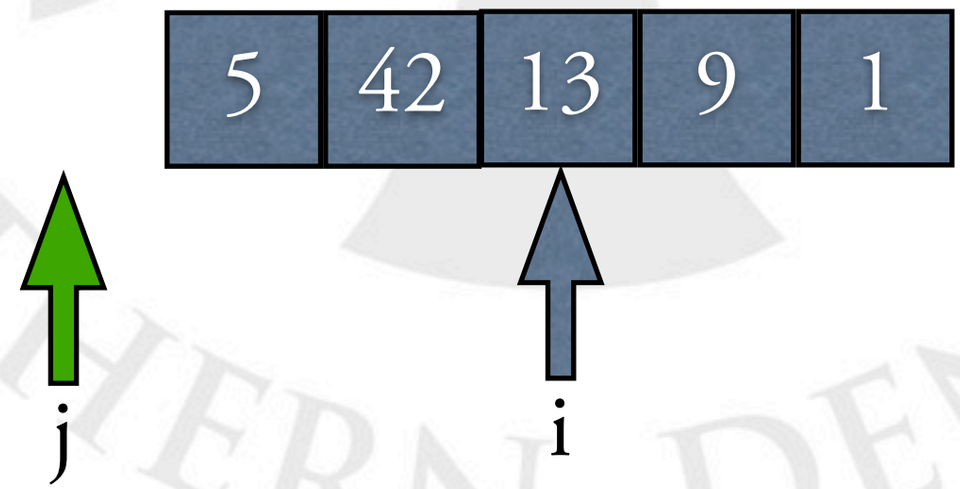
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

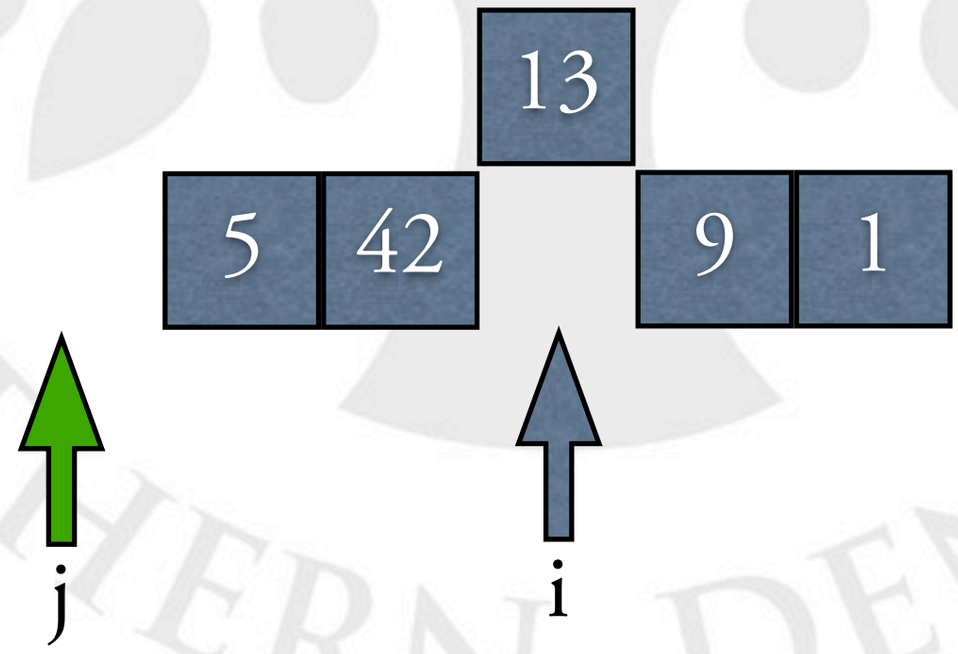
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

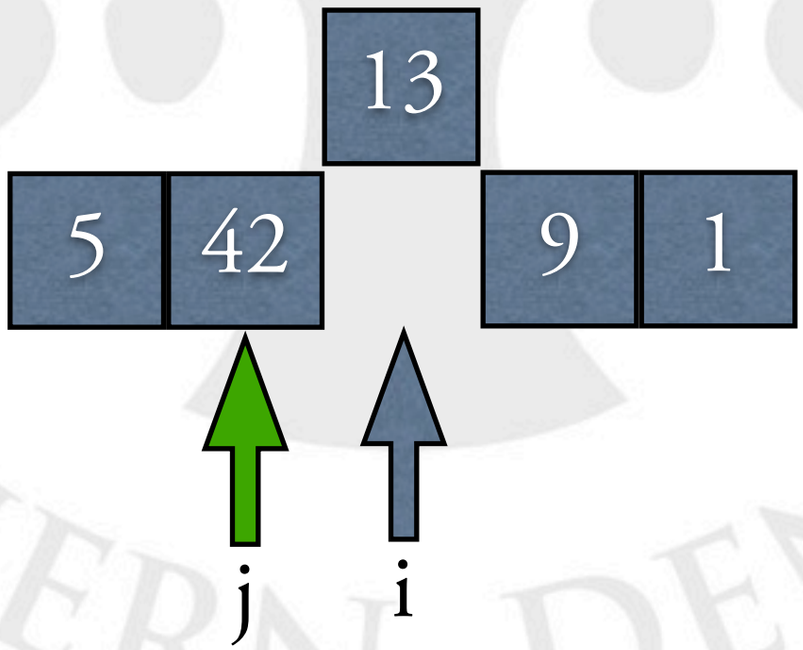
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

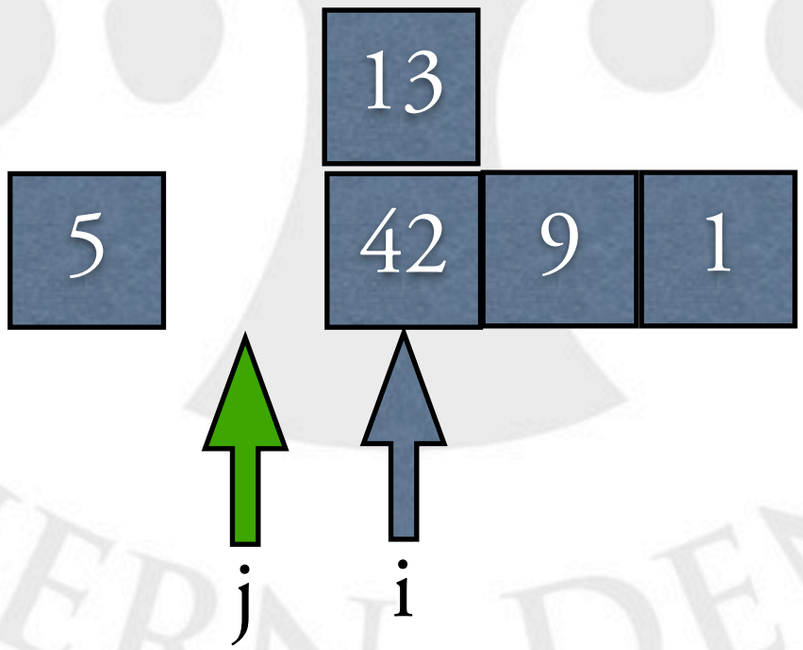
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

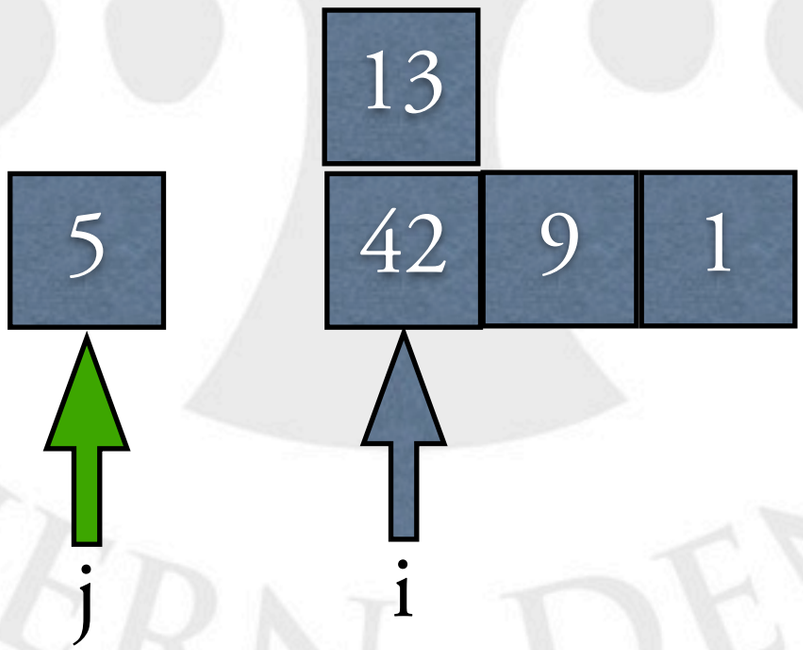
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

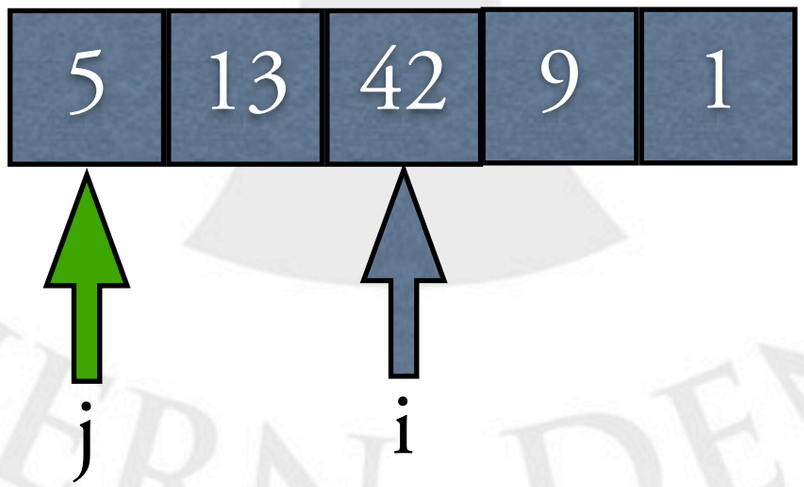
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

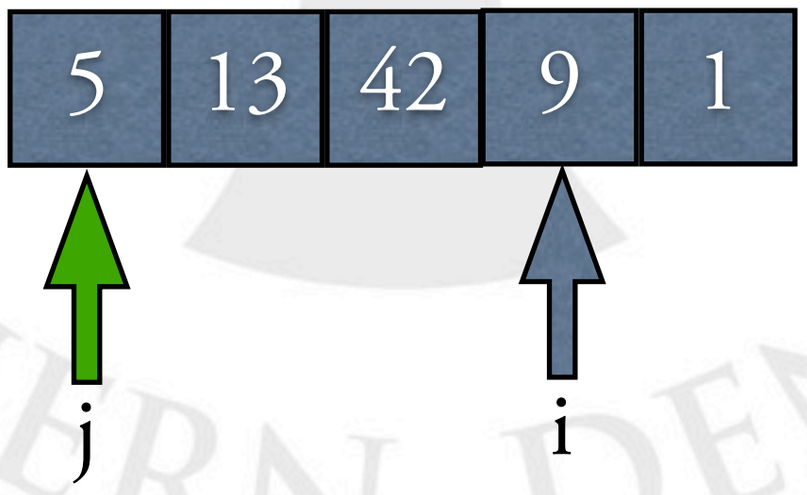
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

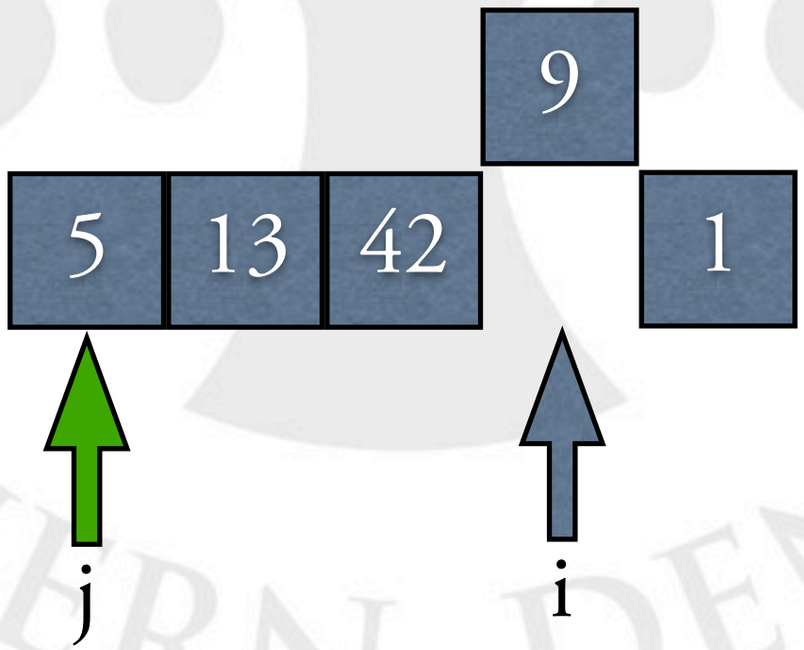
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

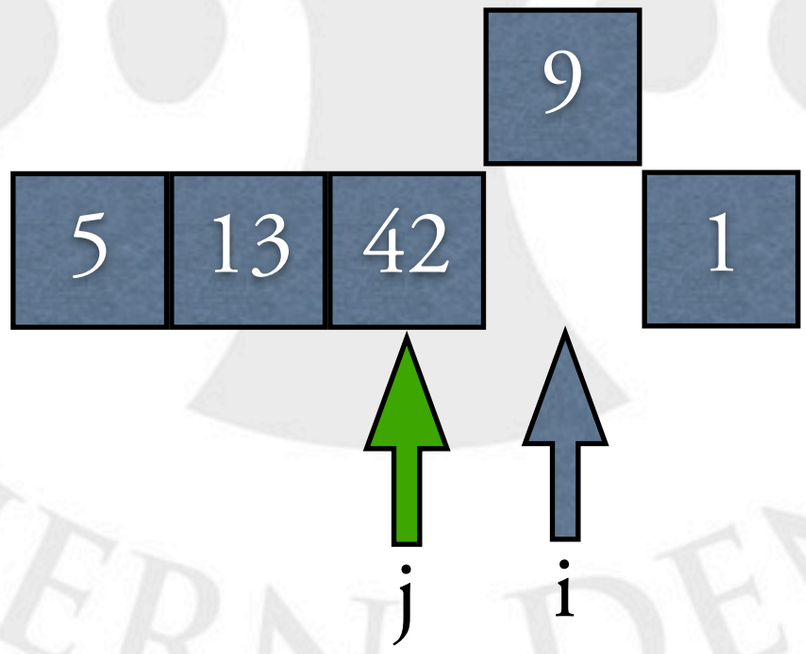
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

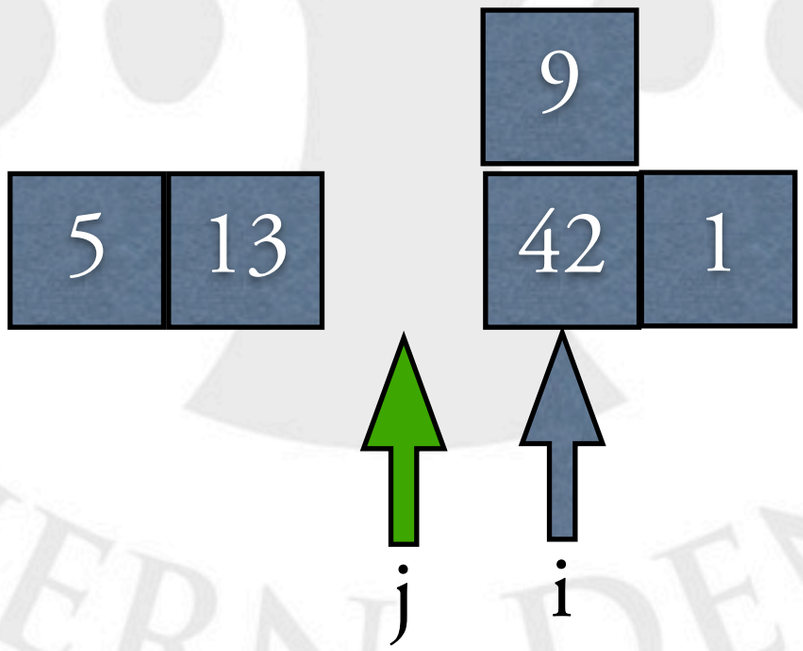
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

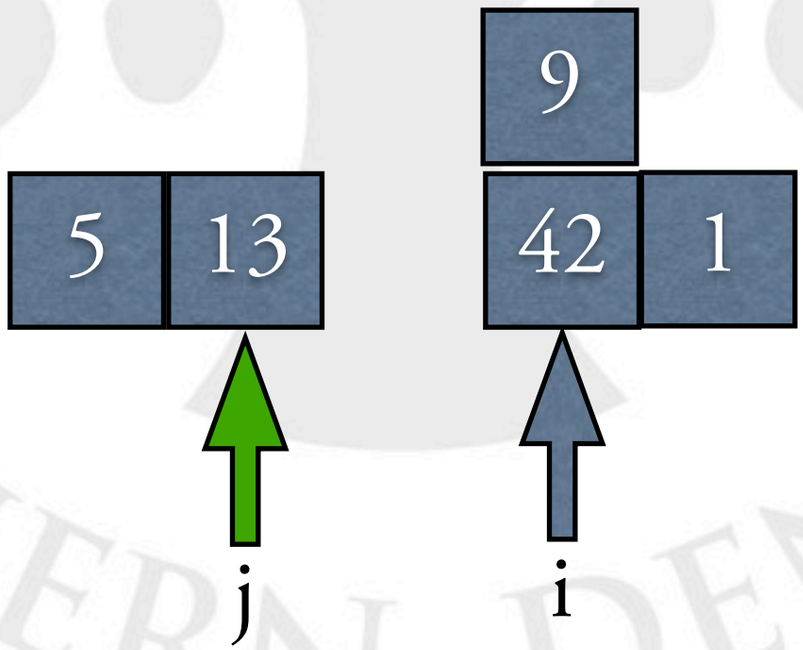
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

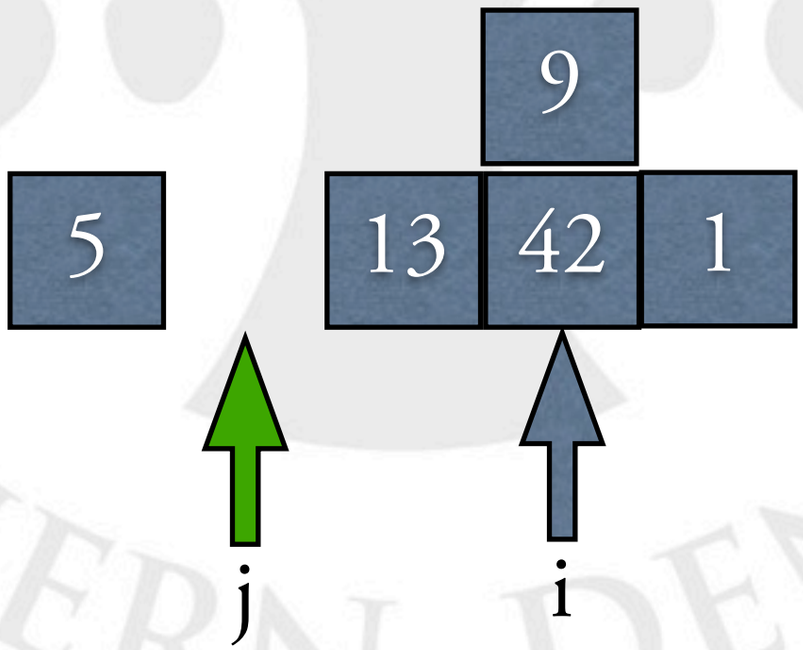
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

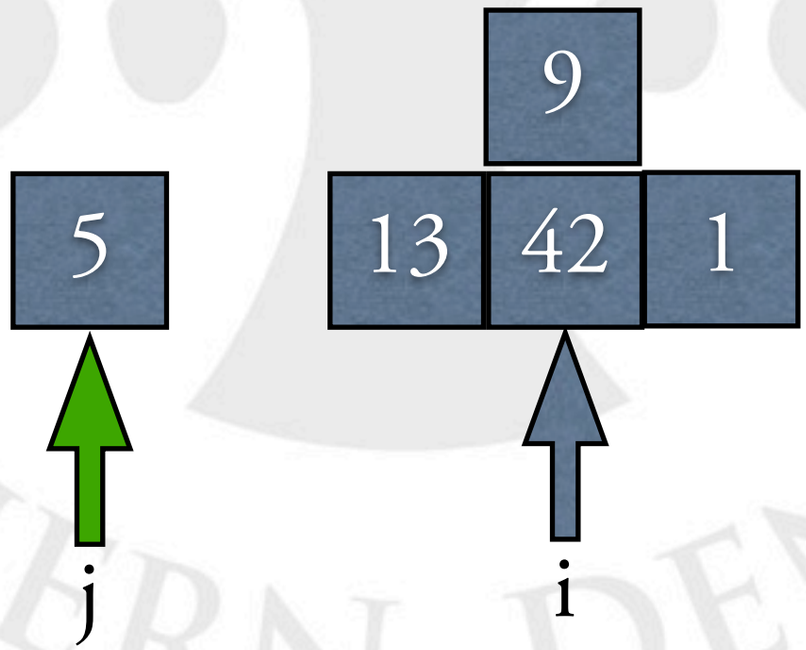
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

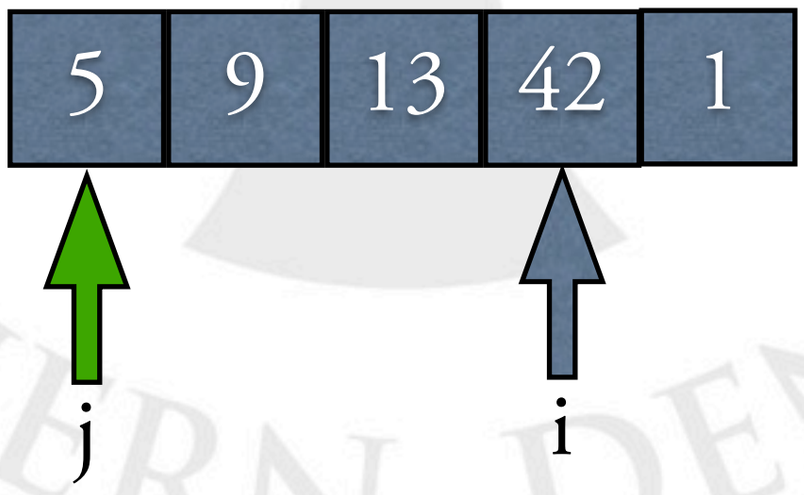
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

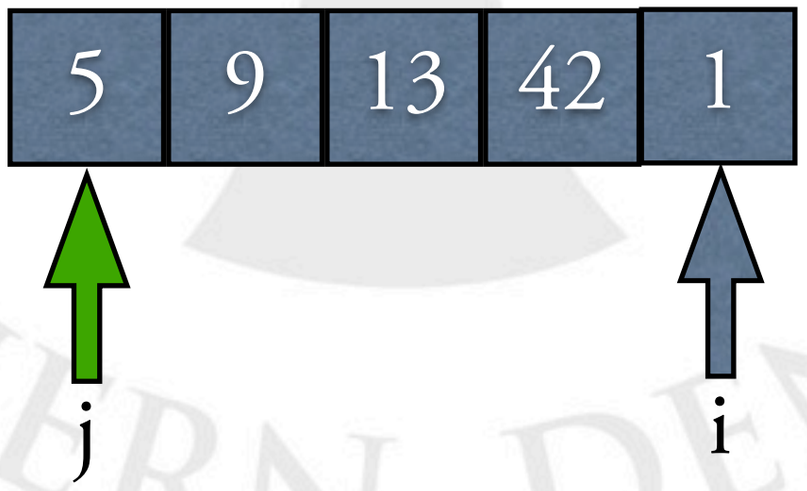
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

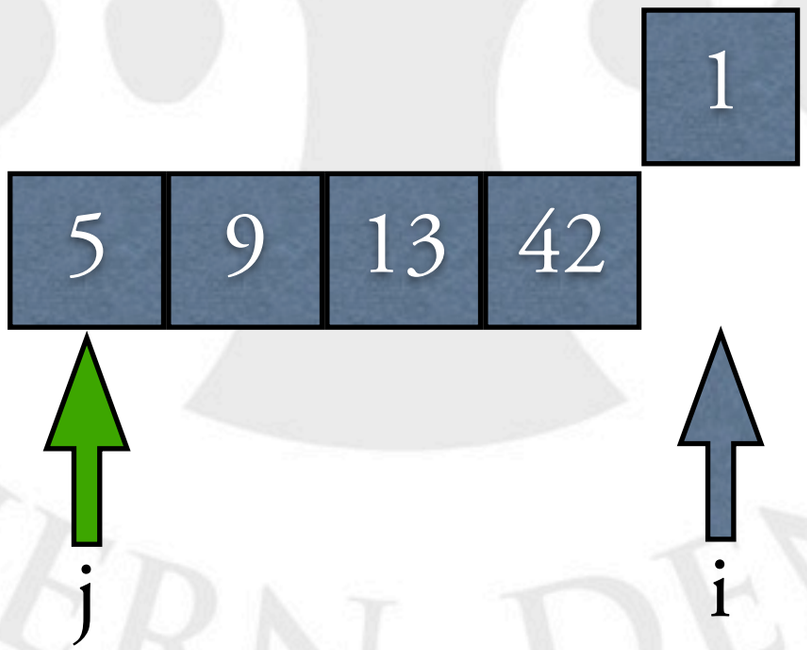
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

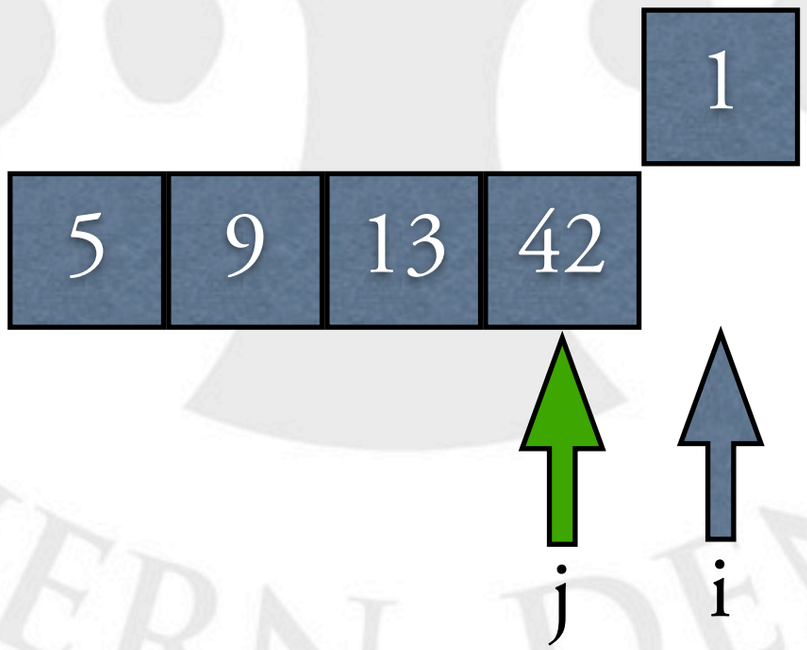
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

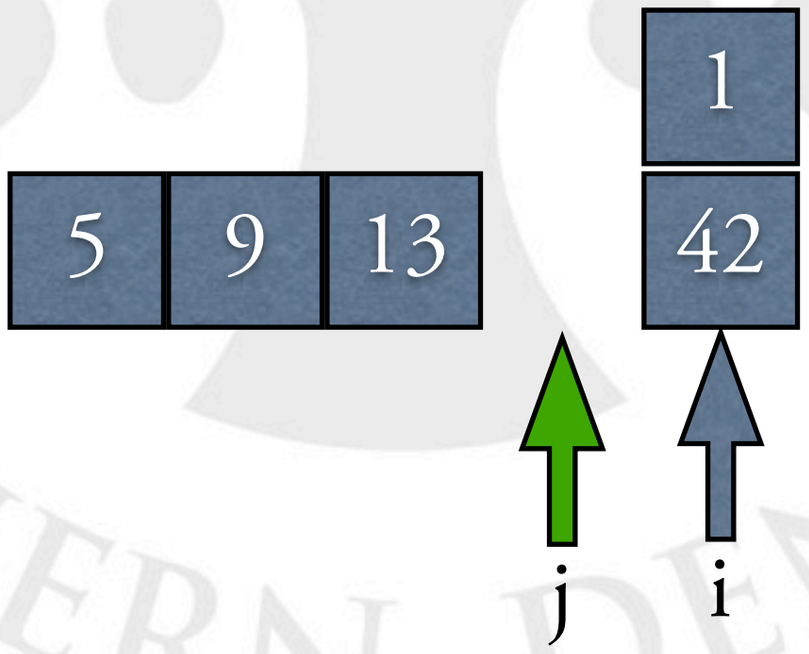
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

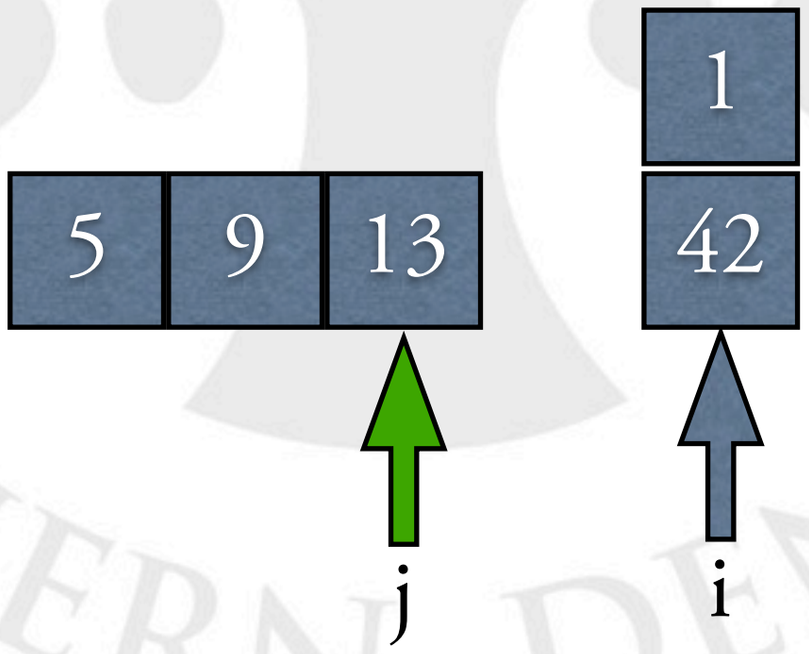
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

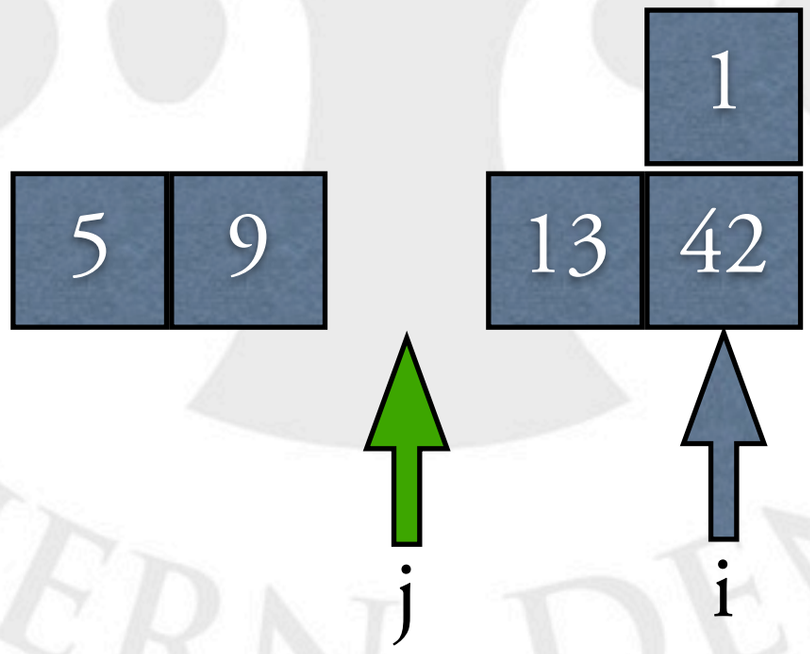
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

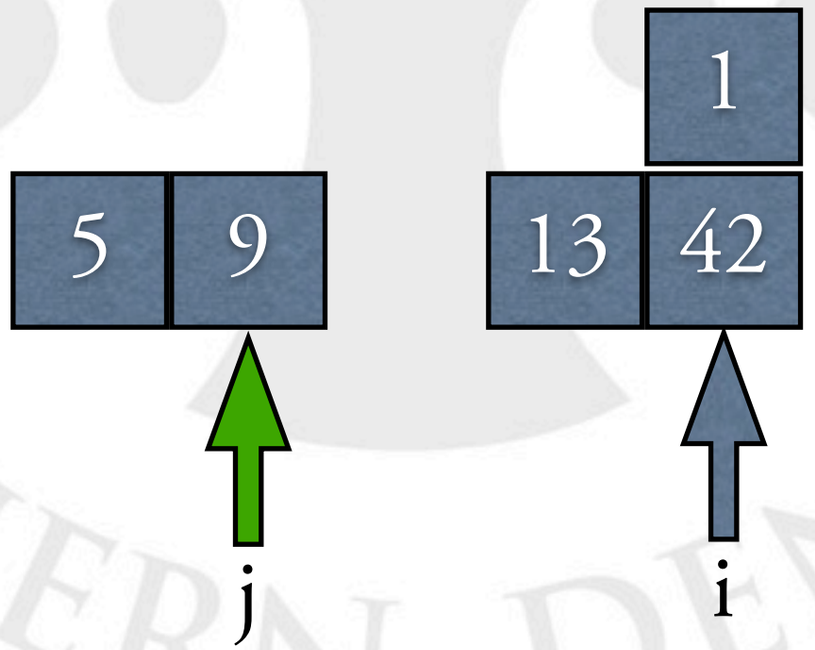
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

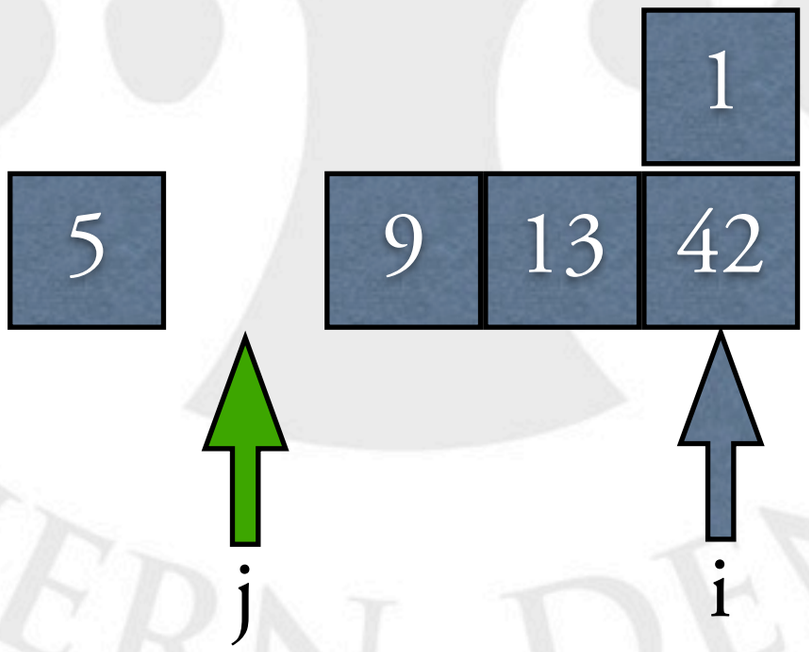
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

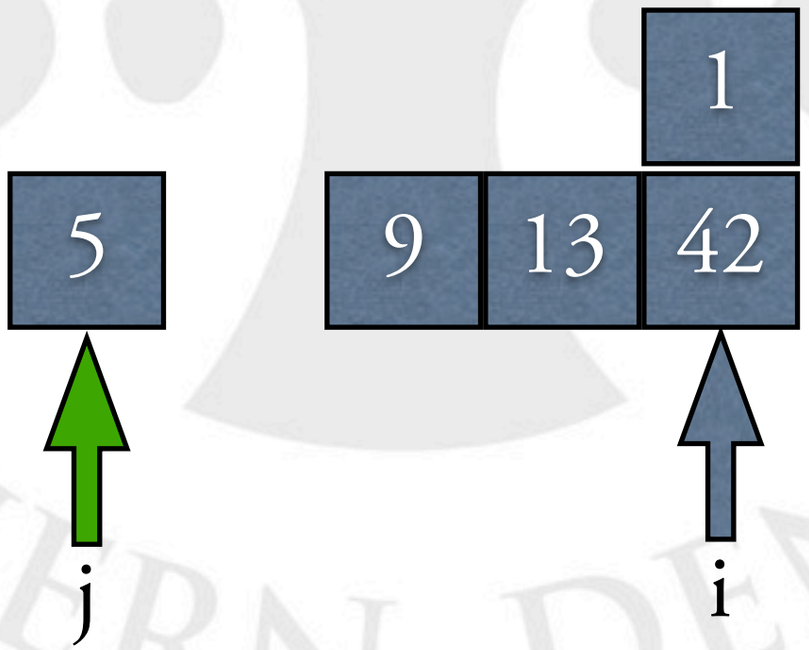
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

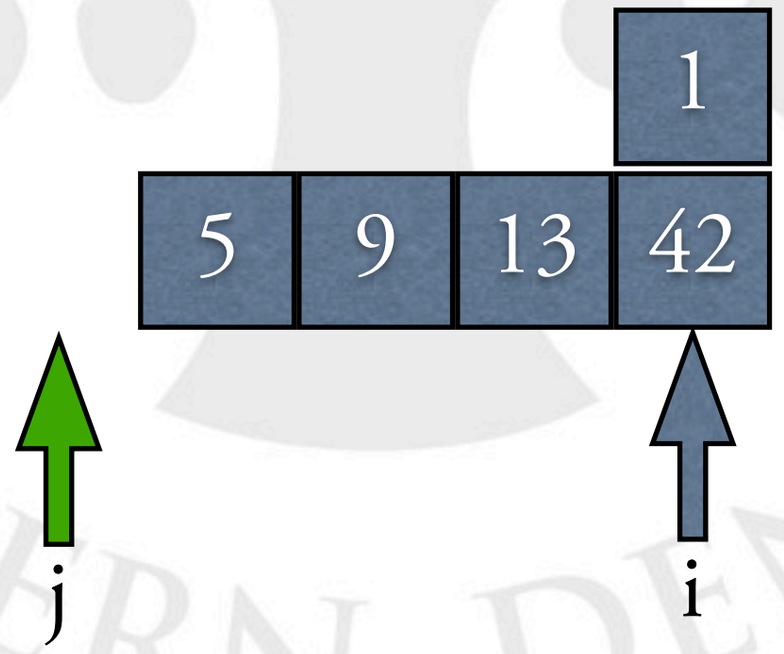
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

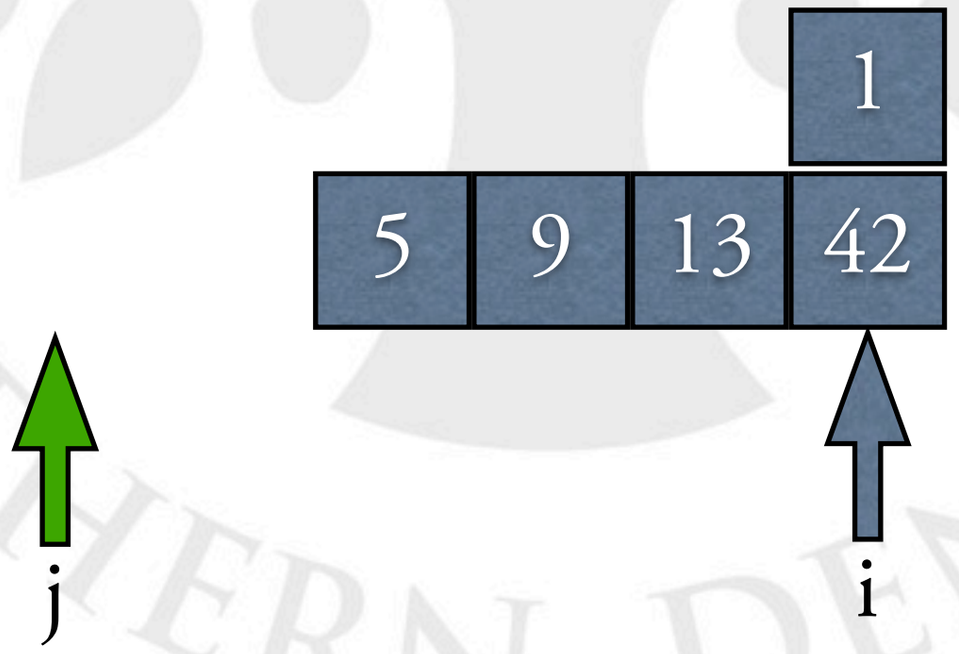
```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```



Indsættelsessortering

- Algoritme

```
for i = 1 to n-1 do:
    value = A[i]
    j = i-1
    while j >= 0 and A[j] > value do:
        A[j+1] = A[j]
        j = j-1
    A[j+1] = value
```

Indsættelsessortering

Indsættelsessortering

- Analyse

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

- Lidt svært at svare på... det afhænger jo af `value`

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
  - Hvad gør vi så?

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
    - Hvad gør vi så?
- Vi er kun ude efter værste tilfælde

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
    - Hvad gør vi så?
- Vi er kun ude efter værste tilfælde
  - Hvad er det?

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
  - Hvad gør vi så?
- Vi er kun ude efter værste tilfælde
  - Hvad er det?
  - Omvendt sorteret array

42	13	9	5	1

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
  - Hvad gør vi så?
- Vi er kun ude efter værste tilfælde
  - Hvad er det?
  - Omvendt sorteret array

42	13	9	5	1

  - Giver samme opførsel som boble-sortering

Indsættelsessortering

- Analyse
- Hvor mange sammenligninger i den inderste løkke?

```
while j >= 0 and A[j] > value do:
    A[j+1] = A[j]
    j = j-1
```

  - Lidt svært at svare på... det afhænger jo af `value`
  - Hvad gør vi så?
- Vi er kun ude efter værste tilfælde
  - Hvad er det?
  - Omvendt sorteret array

  - Giver samme opførsel som boble-sortering
  - Dvs. i værste tilfælde $\frac{1}{2}n^2 - \frac{1}{2}n$

Sortering

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100		

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000		

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000		

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000	6.1 sek.	

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000	6.1 sek.	22 min.

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000	6.1 sek.	22 min.
100.000		

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000	6.1 sek.	22 min.
100.000	1.3 min.	

Sortering

- Det ser ud til at $\frac{1}{2}n^2 - \frac{1}{2}n$ er det bedste man kan gøre
- ?
- Faktisk ikke, man kan komme ned på ca. $n \log n$ sammenligninger
  - Husk f(n) = n vokser hurtigere end f(n) = log n
- Gør det nu den store forskel?
- Ja!
- Antag algoritme A tager $46n \log n$ mikrosekunder
- Algoritme B tager $13n^2$ mikrosekunder

Input-størrelse	Tid for algoritme A	Tid for algoritme B
100	0.03 sek.	0.13 sek.
1000	0.45 sek.	13 sek.
10.000	6.1 sek.	22 min.
100.000	1.3 min.	1.5 dag