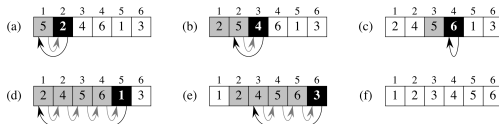


Invarianter

Invarianter

Invariant: Et forhold, som vedligeholdes af algoritmen gennem (dele af) dens udførelse. Udgør ofte kernen af ideen bag algoritmen.

Eksempel: Insertionsort:



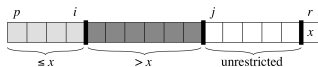
Invariant: Alt til venstre for det sorte felt er sorteret.

Når løkken stopper: hele array'et er til venstre for det sorte felt.

Af invarianten følger så: hele array'et er sorteret. Dvs. algoritmen er korrekt.

Invarianter

Eksempel: Partition fra Quicksort:



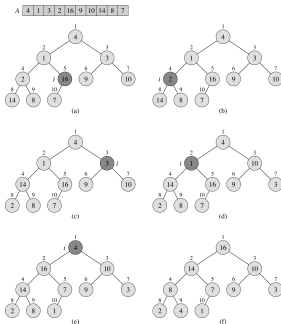
Invariant: Lysegrå del $\leq x <$ mørkegrå del.

Når løkken stopper: Kun x er hvid, resten enten lysegrå eller mørkegrå.

Af invarianten følger så: array'et er delt i tre dele: " $\leq x$ ", " $> x$ " og x selv. Dvs. algoritmen er korrekt.

Invarianter

Eksempel: Build-Heap:



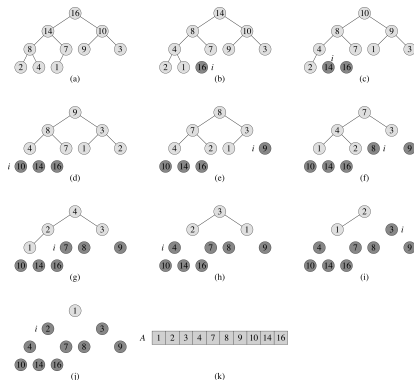
Invariant:

Undertræer, hvis rod har indeks større end den mørke knude, overholder heaporden.

Når løkken stopper: roden af hele træet har indeks større end den mørke knude. Af invarianten følger så: Hele træet overholder heaporden. Dvs. algoritmen er korrekt.

Invarianter

Eksempel: Heapsort (og enhver Selectionsort-baseret sortering):



Invariant: Det mørke er sorteret, og alt i det lyse er \leq det mørke.

Når løkken stopper: hele array'et er mørkt. Af invarianten følger så: hele array'et er sorteret. Dvs. algoritmen er korrekt.

Invarianter

Eksempel: søgning i binære søgetræer.

Invariant: Hvis søgte element k findes, er det i det undertræ, vi er kommet til.

Algoritmen må stoppe fordi vi kigger på mindre og mindre undertræer. Når algoritmen stopper: enten er k fundet eller vi er endt i et tomt undertræ.

I det sidste tilfælde følger så af invarianten: k findes ikke i træet. Dvs. algoritmen er korrekt (i begge tilfælde).

Invarianter

Invariant under rebalancering efter indsættelse i et rød-sort træ:

Der kan være to røde knuder i træk på en sti ét sted i træet, derudover er de rød-sortे krav overholdt. Efter k iterationer er der k færre sorte mellem problemet og roden end i starten.

Invariant under rebalancering efter sletning i et rød-sort træ:

Der kan være én sværtet knude et sted i træet, og hvis sværtningen tælles med, er de rød-sortे krav overholdt. Efter k iterationer er der k færre sorte mellem problemet og roden end i starten.

Invarianten viser her flere ting, som tilsammen gør algoritmen korrekt:

- ▶ Samme case analyse virker hver gang (dækker altid alle muligheder, så algoritmen kan ikke gå i stå, så længe problemet ikke er løst).
- ▶ Algoritmen må stoppe, enten ved at problemet er forsvundet, eller at det har nået roden (hvor det let løses).

Invarianter, mere formelt

Invariant for algoritme:

- ▶ Et udsagn om indholdet af hukommelsen (variable, arrays, ...) som er sandt efter alle skridt.
- ▶ Ved algoritmens afslutning kan korrekthed udledes af udsagnet (samt de omstændigheder som fik algoritmen til at stoppe).

Induktion

At en invariant gælder efter alle skridt, vises ved hjælp af **induktion**:

- 1) Invariant overholdt i starten
- 2) Invariant overholdt før et skridt \Rightarrow invariant overholdt efter skridtet

\Rightarrow

Invariant altid overholdt

(hvor “skridt” ofte er en iteration af en løkke). Dvs: **Vis 1) og 2).**

Induktion \sim “Dominoprincippet”:

- 1) Brik 1 falder
- 2) Brik k falder \Rightarrow brik $k + 1$ falder

\Rightarrow

Alle brikker falder



Brug af invarianter

Invarianter kan bruges på to forskellige detalje-niveauer (med en glidende overgang imellem dem):

1. Som værktøj til at udvikle algoritme-ideer: *Med den rette invariant fanges essensen af metoden*, og algoritmen skal “blot” skrives ud fra at denne invariant skal vedligeholdes.
2. Som værktøj til at nedskrive kode (eller detaljeret pseudo-kode) og *vise denne konkrete kode korrekt*.

På niveau 1 er blødere beskrivelser (tekst, figur) passende. Eksemplerne tidligere illustrerer niveau 1.

På niveau 2 må man nedskrive invarianten præcist i termer af konkrete variable fra koden samt argumentere via den konkrete kodes ændringer af disse.

Vi illustrerer nu niveau 2 på et simpelt eksempel med at finde største element i et array

Eksempel

Find største element i array:

```
max = A[0]
i = 1
while i < A.length
    max = maximum(max, A[i])
    i++
```

Invariant: Efter den k 'te iteration af while-løkken indeholder **max** den største værdi af $A[0..(i-1)]$.

Vises ved induktion på k .

