Sorted list

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Find 104. How many comparisons with sequential search?

A. 1

B. 4

C. 11

D. 12

E. 16

Vote at m.socrative.com. Room number 415439.

Sorted list

D. 12

Can we do better?

Binary search


```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry ← middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

Recursion

- contains reference to itself (subtask)
- termination condition (no infinite loops) — base case

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104
Middle index: 8
TestEntry: 75

Binary search


```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry ← middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104
Middle index: 12
TestEntry: 111

Binary search


```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry ← middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104
Middle index: 10
TestEntry: 99

Binary search


```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry ← middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104
Middle index: 11
TestEntry: 104

Binary search


```
procedure Search(List, TargetValue):
{ Input: List is a list; TargetValue is a possible entry }
{ Output: success if TargetValue in List; failure otherwise }

    if (List empty)
        then Output failure

        else
            TestEntry ← middle entry in List
            if (TargetValue = TestEntry)
                then Output success
                else if (TargetValue < TestEntry
                    then Search(left-of-TestEntry, TargetValue)
                    else Search(right-of-TestEntry, TargetValue)
```

Binary search

7	8	15	53	54	61	66	75	77	99	104	111	123	124	150
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

TargetValue: 104
Result: success

Binary search

Recursion

- contains reference to itself (subtask)
- termination condition (no infinite loops) — base case
- need:
  - initialization
  - modification
  - test for termination
- no more powerful than iteration, but easier to program

Divide-and-Conquer — algorithmic technique
reduce to smaller problem(s)

Binary search — analysis

Each list has length $\leq \frac{1}{2}$ the previous.

List sizes: $n,\ \lfloor \frac{n}{2} \rfloor,\ \lfloor \frac{n}{4} \rfloor,\ \lfloor \frac{n}{8} \rfloor, ..., 1$

1 comparison per list size.

Worst case: $1 + \lfloor \log_2 n \rfloor$ comparisons — $\Theta(\log(n))$

Can it take this many comparisons?

Binary search — analysis

Each list has length $\leq \frac{1}{2}$ the previous.

List sizes: $n,\ \lfloor \frac{n}{2} \rfloor,\ \lfloor \frac{n}{4} \rfloor,\ \lfloor \frac{n}{8} \rfloor, ..., 1$

1 comparison per list size.

Worst case: $1 + \lfloor \log_2 n \rfloor$ comparisons — $\Theta(\log(n))$

Can it take this many comparisons?
Yes.

Binary search — uses

Binary search can be used in many situations.
There does not need to be an explicit list.

In an implicit list, one could have functions of the index, such as $f(n) = (n+1)^2$ or $f(n) = 2^n$.

Sorting

How do you sort? Think about cards.

Insertion Sort

procedure Sort(List):
{ Input: List is a list }
{ Output: List, with same entries, but in nondecreasing order }

$N \leftarrow 2$
while ($N \leq$ length(List) **do**
    Pivot $\leftarrow N$th entry
    $j \leftarrow N - 1$
    while ($j > 0$ and jth entry $>$ Pivot) **do**
        move jth entry to loc. $j + 1$
        $j \leftarrow j - 1$
    place Pivot in $j + 1$st loc.
    $N \leftarrow N + 1$

Insertion Sort

What happens if List has 0 or 1 entry?

A. Sort crashes

B. Sort returns the input list unchanged

C. Sort returns something wrong

Vote at m.socrative.com. Room number 415439.

Insertion Sort

17	8	15	53	18	12	2	75
1	2	3	4	5	6	7	8

N: 2
Pivot: 8
j: 1
jth entry: 17

Insertion Sort

8	17	15	53	18	12	2	75
1	2	3	4	5	6	7	8

N: 2
Pivot: 8
j: 0
jth entry: none

Insertion Sort

8	17	15	53	3	12	2	75
1	2	3	4	5	6	7	8

N: 3
Pivot: 15
j: 2
jth entry: 17

Insertion Sort

8	17	15	53	3	12	2	75
1	2	3	4	5	6	7	8

N: 3
Pivot: 15
j: 2
jth entry: 17
Continue on board.

Insertion Sort — correctness

procedure Sort(List):
{ Input: List is a list }
{ Output: List, with same entries, but in nondecreasing order }

$N \leftarrow 2$
while ($N \leq$ length(List) **do**
{ loop invariants:
1. entries 1 thru $N-1$ in List are in sorted order
2. the same items are in List as originally }
Pivot $\leftarrow N$th entry
$j \leftarrow N-1$
while ($j > 0$ and jth entry $>$ Pivot) **do**
move jth entry to loc. $j+1$
$j \leftarrow j-1$
place Pivot in $j+1$st loc.
$N \leftarrow N+1$

Insertion Sort — correctness

procedure Sort(List):
{ Input: List is a list }
{ Output: List, with same entries, but in nondecreasing order }
    $N \leftarrow 2$
    while ($N \leq$ length(List) **do**
        Pivot $\leftarrow N$th entry
        $j \leftarrow N - 1$
        while ($j > 0$ and jth entry $>$ Pivot) **do**
{ loop invariants: 1. no item in loc. $j + 1$
    2. entries in locs. $j + 2$ to N are larger than Pivot
    3. entries in locs. 1 to $N - 1$ stay in same relative order
    4. no entries in locs. $N + 1$ to length(List) are changed }
            move jth entry to loc. $j + 1$
            $j \leftarrow j - 1$
        place Pivot in $j + 1$st loc.
        $N \leftarrow N + 1$

Insertion Sort — analysis

Suppose list has n entries.
How many comparisons occur in the best case?

A. 1

B. 2

C. n-1

D. n

E. n+1

Vote at m.socrative.com. Room number 415439.
(What is the best case?)

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.

Insertion Sort

procedure Sort(List):
{ Input: List is a list }
{ Output: List, with same entries, but in nondecreasing order }

$N \leftarrow 2$
while ($N \leq$ length(List) **do**
    Pivot $\leftarrow N$th entry
    $j \leftarrow N - 1$
    while ($j > 0$ and jth entry $>$ Pivot) **do**
        move jth entry to loc. $j + 1$
        $j \leftarrow j - 1$
    place Pivot in $j + 1$st loc.
    $N \leftarrow N + 1$

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.
Inner loop from $N-1$ to 1.
Total: $\sum_{N=2}^{n}(N-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.
Inner loop from $N-1$ to 1.
Total: $\sum_{N=2}^{n}(N-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Can it take this many comparisons?

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.
Inner loop from $N-1$ to 1.
Total: $\sum_{N=2}^{n}(N-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Can it take this many comparisons?
Yes.

Insertion Sort — analysis

What list gives this worst case?

A. an ordered list

B. a list in reverse order

C. a random list

D. none of the above

E. all of the above

Vote at m.socrative.com. Room number 415439.

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.
Inner loop from $N-1$ to 1.
Total: $\sum_{N=2}^{n}(N-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Average case number of comparisons:

On average place next Pivot half way down the list.
$\frac{n(n-1)}{4} \in \Theta(n^2)$.

Insertion Sort — analysis

Worst case number of comparisons:

Outer loop from 2 to n.
Inner loop from $N-1$ to 1.
Total: $\sum_{N=2}^{n}(N-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.

Average case number of comparisons:

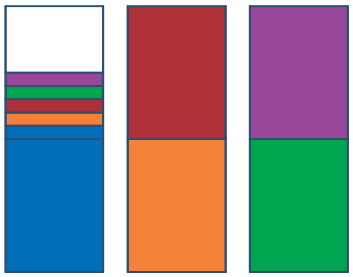
On average place next Pivot half way down the list.
$\frac{n(n-1)}{4} \in \Theta(n^2)$.

There exist algorithms which do $\Theta(n \log n)$ comparisons.

Classical bin packing

Use as few bins as possible:
Item sizes: $n \times [1/2, \epsilon]$
Bin size: 1

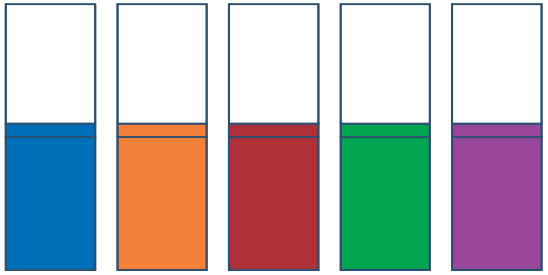
Result by First-Fit algorithm:



Classical bin packing

Use as few bins as possible:
Item sizes: $n \times [1/2, \epsilon]$
Bin size: 1

Result by Worst-Fit algorithm:



Dual bin packing

Given a fixed number of bins, pack as many items as possible.

Bin size: 1
Number of bins: 4
Item sizes:

- $\frac{1}{4}, \frac{1}{4}, \frac{1}{4}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$

Can they all be there?

Dual bin packing

Item sizes:

- $\frac{1}{4}, \frac{1}{4}, \frac{1}{4}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$

Can they all be there?

First check:

$3\frac{3}{12} + 3\frac{9}{12} + 3\frac{4}{12} = 4$

Dual bin packing

Item sizes:

- $\frac{1}{4}, \frac{1}{4}, \frac{1}{4}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{5}{12}, \frac{1}{3}$
- $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$

Can they all be there?
What about First-Fit?
An optimal algorithm?

Bin packing

First-Fit is an on-line algorithm:
It handles requests without looking at future requests.
Some problems are on-line in nature. Examples?

Solving bin packing optimally is NP-hard.
Brute force takes a long time.

Bin packing

First-Fit is an on-line algorithm:
It handles requests without looking at future requests.
Some problems are on-line in nature. Examples?

Solving bin packing optimally is NP-hard.
Brute force takes a long time.
Approximation algorithms: First-Fit-Decreasing, even better...

Special case: all sizes multiples of $\frac{1}{12}$.
Fill one bin completely if possible.

First-Fit for dual bin packing

procedure First-Fit-Dual(List):
{ Input: List is a list of items with sizes ≤ 1 }
{ Output: Number of rejected items }

$k \leftarrow$ number of bins { all empty }
Count $\leftarrow 0$ { number rejected }
get next item x and remove from list
$i \leftarrow 1$
while ($i \leq k$ and x does not fit in bin i) **do**
$i \leftarrow i + 1$
if ($i \leq k$)
then put x in bin i
else Count $\leftarrow$ Count+1
return(Count)

First-Fit for dual bin packing (correct)

procedure First-Fit-Dual(List):
{ Input: List is a list of items with sizes ≤ 1 }
{ Output: Number of rejected items }

$k \leftarrow$ number of bins { all empty }
Count $\leftarrow 0$ { number rejected }
while there are still items in the list
    get next item x and remove from list
    $i \leftarrow 1$
    while ($i \leq k$ and x does not fit in bin i) **do**
        $i \leftarrow i+1$
    if ($i \leq k$)
        then put x in bin i
        else Count $\leftarrow$ Count+1
return(Count)