

CS 457, Fall 2016

Drexel University, Department of Computer Science

Lecture 8

Second Problem Set

1. (20 pts) Solve problem 4-3 on page 108 of your textbook.
2. (15 pts) You are given a set S of n integers, as well as one more integer v . Design an algorithm that determines whether or not there exist two distinct elements $x, y \in S$ such that $x + y = v$. Your algorithm should run in time $O(n \log n)$, and it should return (x, y) if such elements exist and (NIL, NIL) otherwise. Prove the worst case running time bound and the correctness of the algorithm.
3. (15 pts) Prove tight worst-case asymptotic upper bounds for the following recurrence equations that satisfy $T(n) = 1$ for $n \leq 2$, and depend on a variable $q \in [0, n/4]$:
 - (a) $T(n) = T(n - 2q - 1) + T(3q/2) + T(q/2) + \Theta(1)$
 - (b) $T(n) = T(n - q - 1) + T(n/2 - q) + \Theta(n)$
 - (c) $T(n) = T(n - q - 1) + T(3q) + \Theta(n)$
4. (10 pts) Consider the following silly randomized variant of binary search. You are given a sorted array A of n integers and the integer v that you are searching for is chosen uniformly at random from A . Then, instead of comparing v with the value in the middle of the array, the randomized binary search variant chooses a random number r from 1 to n and it compares v with $A[r]$. Depending on whether v is larger or smaller, this process is repeated recursively on the left sub-array or the right sub-array, until the location of v is found. Prove a tight bound on the expected running time of this algorithm.
5. (10 pts) Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \log n$? Why or why not? Give an asymptotic upper bound for this recurrence.

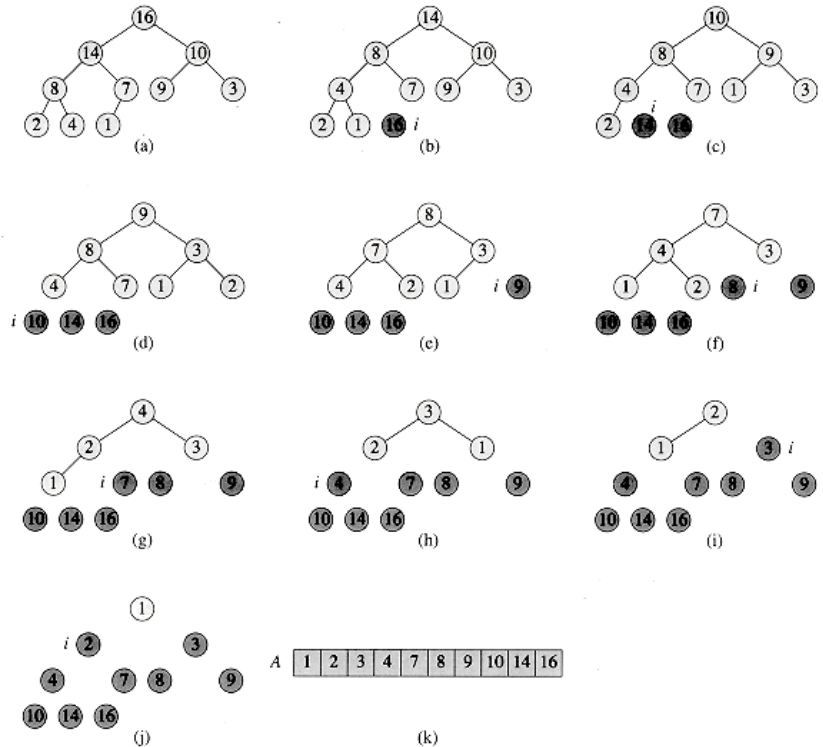
Second Problem Set

6. (10 pts) Use a recursion tree to give an asymptotically tight solution to the recurrence:
- (a) $T(n) = T(n - a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.
 - (b) $T(n) = T(an) + T((1 - a)n) + cn$, where $a \in (0, 1)$ and $c > 0$ are constants.
7. (10 pts) For the following recurrences, find the bound implied by the master theorem. Then, try to prove the same bound using the substitution method. If your initial approach fails, show how it fails, and try subtracting off a lower-order term to make it work:
- (a) $T(n) = 4T(n/3) + n$
 - (b) $T(n) = 4T(n/2) + n$
8. (5 pts) Solve the recurrence $T(n) = 3T(\sqrt{n}) + \log n$ by making a change of variable. Your solution should be asymptotically tight. Do not worry about whether values are integral.
9. (5 pts) Using proof by induction, show that there are at most $\lceil n/2^{h+1} \rceil$ nodes of height h in any n -element heap.

Heapsort

Heapsort(A)

1. Build-Max-Heap(A)
2. **for** $i = A.length$ **down to** 2
3. exchange $A[1]$ with $A[i]$
4. $A.heap\text{-}size = A.heap\text{-}size - 1$
5. Max-Heapify(A, 1)

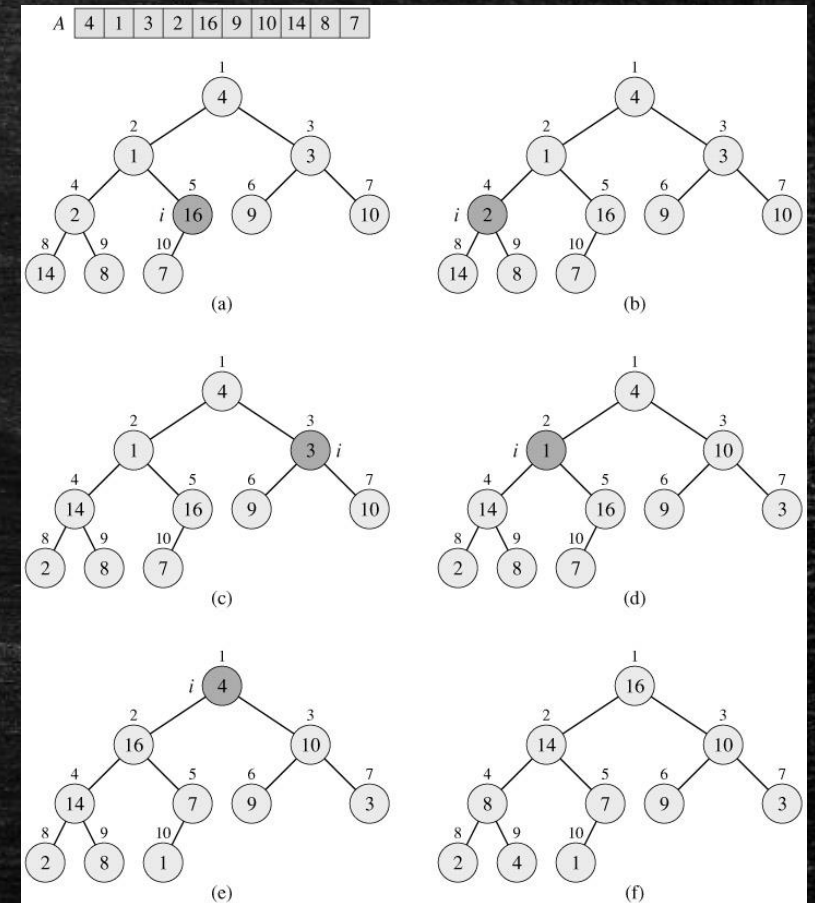


Heapsort

Build-Max-Heap (A)

1. A.heap-size = A.length
2. **for** $i = \lfloor A.length/2 \rfloor$ **down to** 1
3. Max-Heapify(A, i)

$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lfloor \frac{n}{2^{h+1}} \right\rfloor O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\right) = O(n)$$



Today's Lecture

- Selection Problem
- Binary Search Trees

Order Statistics

The i^{th} order statistic of a set of n numbers: the i^{th} smallest number in sorted sequence:

A	4	1	3	2	16	9	10	14	8	7
---	---	---	---	---	----	---	----	----	---	---

- Minimum or first order statistic: 1
- Maximum or n^{th} order statistic: 16
- Median or $(n/2)^{\text{th}}$ order statistic: 7 or 8
(both are medians, happens when n is even!)

The Selection Problem

- **Input:** An array A of **distinct** numbers of size n , and a number i .
- **Output:** The element x in A that is larger than exactly $i - 1$ other elements in A .
- Finding *maximum* and *minimum* can be easily solved in linear time (i.e. $O(n)$). (it's actually $\Theta(n)$).

Randomized Selection Algorithm

Randomized-Select(A, p, r, i)

1. if $p == r$
2. return $A[p]$
3. $q = \text{Randomized-Partition}(A, p, r)$
4. $k = q - p + 1$
5. if $i == k$
6. return $A[q]$
7. else if $i \leq k$
8. Randomized-Select($A, p, q - 1, i$)
9. else
10. Randomized-Select($A, q + 1, r, i - k$)

Running Time

- What is the worst-case running time of the algorithm?
 - a) $O(n^2)$
 - b) $O(n \log n)$
 - c) $O(n)$

- What about the average-case?
 - a) $O(n^2)$
 - b) $O(n \log n)$
 - c) $O(n)$

Running Time

- $X_k = \mathbb{I}\{\text{the subarray } A[p, \dots, q] \text{ has exactly } k \text{ elements}\}$
- $\mathbb{E}[X_k] = \frac{1}{n}$

$$\begin{aligned} T(n) &\leq \sum_{k=1}^n X_k (T(\max\{k-1, n-k\}) + O(n)) \\ &= \sum_{k=1}^n X_k (T(\max\{k-1, n-k\})) + \sum_{k=1}^n X_k O(n) \\ &= \sum_{k=1}^n X_k (T(\max\{k-1, n-k\})) + O(n) \end{aligned}$$

Running Time

$$\begin{aligned}\mathbb{E}[T(n)] &\leq \mathbb{E} \left[\sum_{k=1}^n X_k (T(\max\{k-1, n-k\})) + O(n) \right] \\&= \sum_{k=1}^n \mathbb{E}[X_k (T(\max\{k-1, n-k\}))] + O(n) \\&= \sum_{k=1}^n \mathbb{E}[X_k] \mathbb{E}[T(\max\{k-1, n-k\})] + O(n) \\&= \sum_{k=1}^n \frac{1}{n} \mathbb{E}[T(\max\{k-1, n-k\})] + O(n)\end{aligned}$$

Running Time

$$\text{Thus, } \mathbb{E}[T(n)] \leq \sum_{k=1}^n \frac{1}{n} \mathbb{E}[T(\max\{k-1, n-k\})] + O(n),$$

$$\text{and } \max\{k-1, n-k\} = \begin{cases} k-1 & \text{if } k > \lfloor n/2 \rfloor \\ n-k & \text{if } k \leq \lfloor n/2 \rfloor \end{cases}$$

$$\text{So, } \mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$$

Running Time

Using substitution, we solve: $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Assume: $\mathbb{E}[T(n')] \leq cn'$ for some constant c and every $n' < n$.

$$\text{Then, } \mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} ck + an \leq \frac{2c}{n} \left(\frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an$$

Which leads to $\mathbb{E}[T(n)] \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right)$

Running Time

Using substitution, we solve: $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Assume: $\mathbb{E}[T(n')] \leq cn'$ for some constant c and $n' \geq 1$ for $c > 4a$,

Then, $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} ck + an \leq \frac{2c}{n} \left(\frac{n^2 - n}{2} - \frac{n}{2} \right) + an$ and $n \geq \frac{2c}{c-4a}$

Which leads to $\mathbb{E}[T(n)] \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right) \leq cn$

Running Time (Alternative Analysis)

- $Y_k = \mathbb{I}\{\text{the longest of the two subarrays of partition has exactly } k \text{ elements}\}$
- $\mathbb{E}[Y_k] = \frac{2}{n}$ for $\lfloor \frac{n}{2} \rfloor \leq k \leq n-1$ (if n is odd, then $\mathbb{E}[Y_k] = \frac{1}{\lfloor n/2 \rfloor}$)

$$T(n) \leq \sum_{k=\lfloor n/2 \rfloor}^{n-1} Y_k (T(k) + O(n))$$

$$\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$$

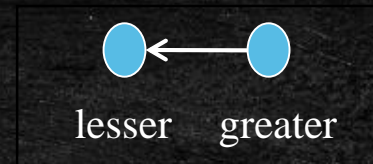
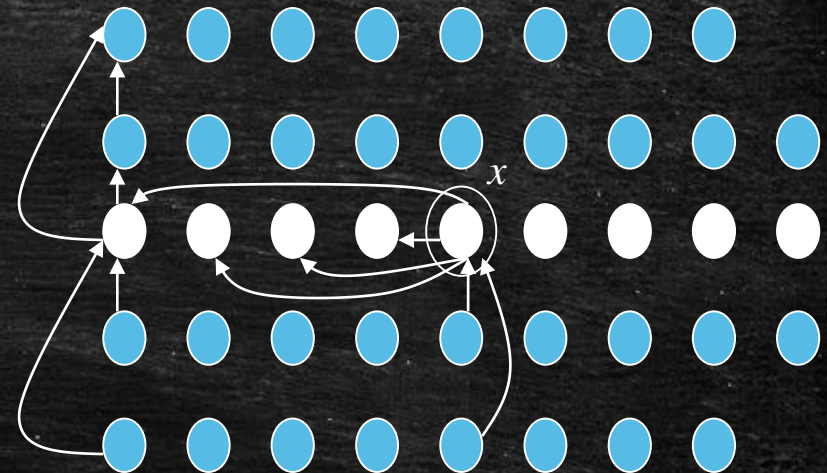
Selection Problem

- Can we design algorithms with better **worst case** guarantees?

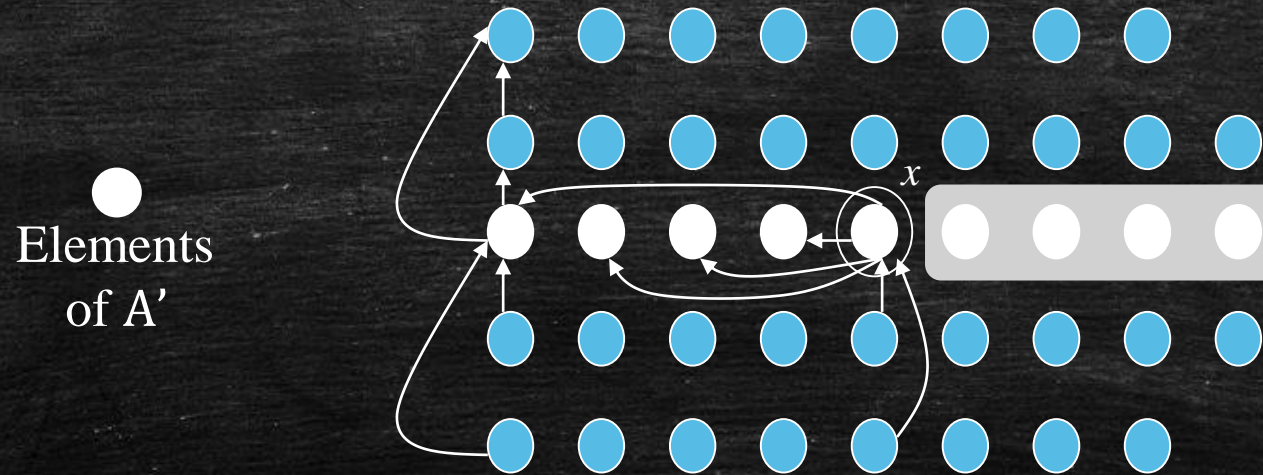
Worst case linear time selection

Select(A,p,r,i)

1. Divide **A** into $n/5$ groups of size 5.
2. Find the median of each group of 5 by brute force, and store them in a set **A'** of size $n/5$.
3. Recursively use **Select(A', 1, $n/5$, $n/10$)** to find the median **x** of $n/5$ medians.
4. Partition elements of **A** around **x**.
Let **k** be the order of **x** found in the partitioning.
5. if $i = k$
6. return **x**
7. else if $i < k$
8. **Select(A, p, q - 1, i)**
9. else
10. **Select(A, q + 1, r, i - k)**

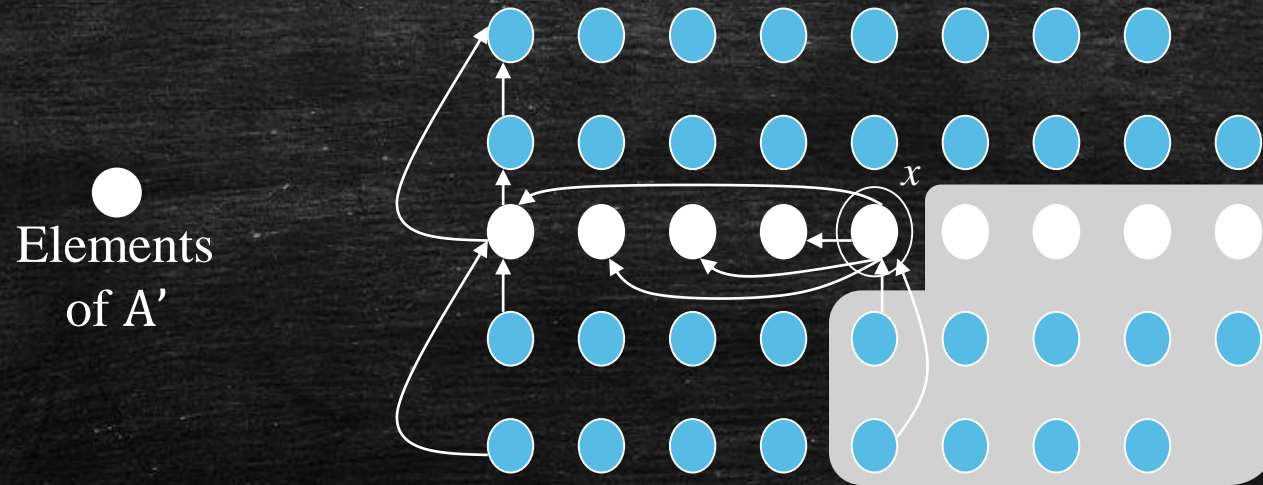


Analysis



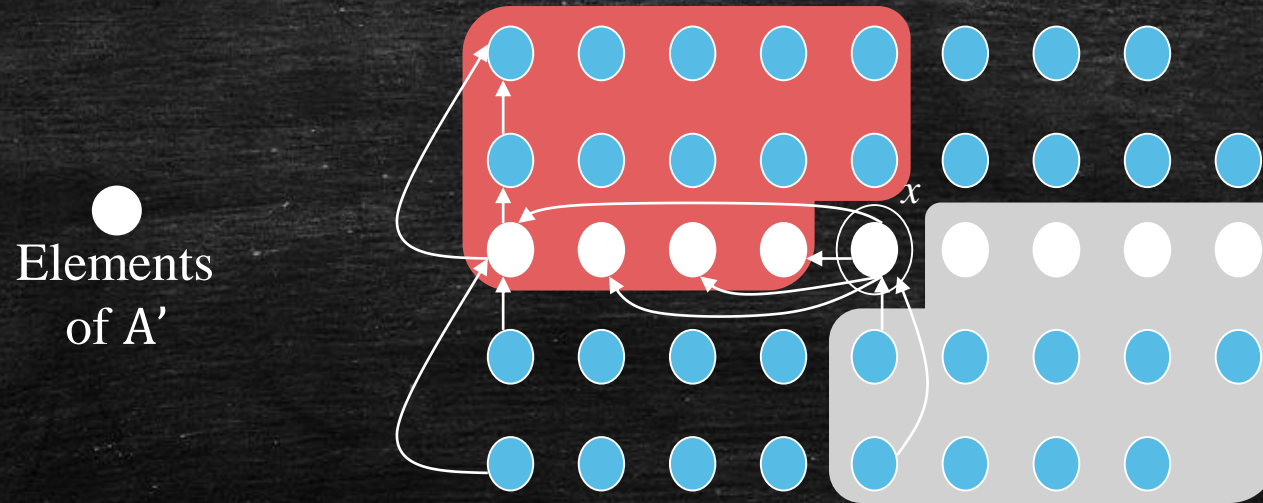
- At least **half** of the $\lfloor n/5 \rfloor$ elements in A' are $\geq x$

Analysis



- At least **half** of the $\lfloor n/5 \rfloor$ elements in A' are $\geq x$
- Groups whose **median** $\geq x$ have **at least 3 elements** $\geq x$.
- Therefore, at least $3n/10 - 6$ elements of A are $\geq x$.

Analysis



- At least **half** of the $[n/5]$ elements in A' are $\geq x$
- Groups whose **median** $\geq x$ have **at least 3 elements** $\geq x$.
- Therefore, at least $3n/10 - 6$ elements of A are $\geq x$.

Worst case linear time selection

Select(A,p,r,i)

1. Divide A into $n/5$ groups of size 5.
2. Find the median of each group of 5 by brute force, and store them in a set A' of size $n/5$.
3. Recursively use Select(A', 1, $n/5$, $n/10$) to find the median x of $n/5$ medians.
4. Partition elements of A around x.
Let k be the order of x found in the partitioning.
5. if $i = k$
6. return x
7. else if $i < k$
8. Select(A, p, q - 1, i)
9. else
10. Select(A, q + 1, r, i - k)

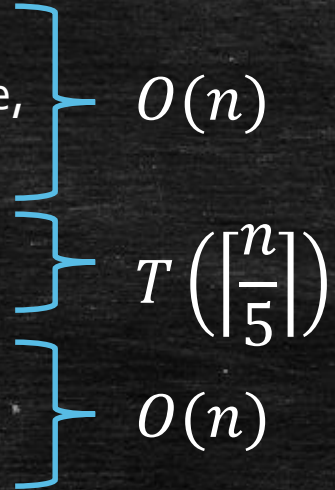
Worst case linear time selection

Select(A,p,r,i)

1. Divide **A** into $n/5$ groups of size 5.
 2. Find the median of each group of 5 by brute force, and store them in a set **A'** of size $n/5$.
 3. Recursively use **Select(A', 1, n/5, n/10)** to find the median **x** of $n/5$ medians.
 4. Partition elements of **A** around **x**.
Let **k** be the order of **x** found in the partitioning.
 5. if $i = k$
 6. return **x**
 7. else if $i < k$
 8. **Select(A, p, q - 1, i)**
 9. else
 10. **Select(A, q + 1, r, i - k)**
- $O(n)$
- $O(n)$

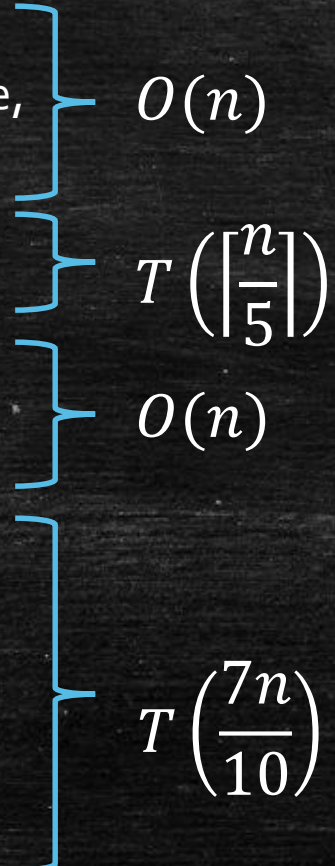
Worst case linear time selection

Select(A,p,r,i)

1. Divide **A** into $n/5$ groups of size 5.
 2. Find the median of each group of 5 by brute force, and store them in a set **A'** of size $n/5$.
 3. Recursively use **Select(A', 1, n/5, n/10)** to find the median **x** of $n/5$ medians.
 4. Partition elements of **A** around **x**.
Let **k** be the order of **x** found in the partitioning.
 5. if $i = k$
 6. return **x**
 7. else if $i < k$
 8. **Select(A, p, q - 1, i)**
 9. else
 10. **Select(A, q + 1, r, i - k)**
- 
- The diagram consists of three large curly braces on the right side of the list, grouping steps and their corresponding time complexities:
- A brace groups steps 1 and 2, with the time complexity $O(n)$ to its right.
 - A brace groups steps 3 and 4, with the time complexity $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$ to its right.
 - A brace groups steps 4 and 5, with the time complexity $O(n)$ to its right.

Worst case linear time selection

Select(A, p, r, i)

1. Divide **A** into $n/5$ groups of size 5.
 2. Find the median of each group of 5 by brute force, and store them in a set **A'** of size $n/5$.
 3. Recursively use **Select(A', 1, n/5, n/10)** to find the median **x** of $n/5$ medians.
 4. Partition elements of **A** around **x**.
Let **k** be the order of **x** found in the partitioning.
 5. if $i = k$
 6. return **x**
 7. else if $i < k$
 8. **Select(A, p, q - 1, i)**
 9. else
 10. **Select(A, q + 1, r, i - k)**
- 
- The diagram uses blue curly braces to group steps and their corresponding time complexities:
- Steps 1 and 2 are grouped with $O(n)$.
 - Step 3 is grouped with $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$.
 - Steps 4 and 5 are grouped with $O(n)$.
 - Steps 7, 8, 9, and 10 are grouped with $T\left(\frac{7n}{10}\right)$.

Worst case linear time selection

Select(A, p, r, i)

1. Divide **A** into $n/5$ groups of size 5.
 2. Find the median of each group of 5 by brute force, and store them in a set **A'** of size $n/5$.
 3. Recursively use **Select(A', 1, n/5, n/10)** to find the median **x** of $n/5$ medians.
 4. Partition elements of **A** around **x**.
Let **k** be the order of **x** found in the partitioning.
 5. if $i = k$
 6. return **x**
 7. else if $i < k$
 8. **Select(A, p, q - 1, i)**
 9. else
 10. **Select(A, q + 1, r, i - k)**
- }

}

}

}

}

$O(n)$

$T\left(\left\lceil \frac{n}{5} \right\rceil\right)$

$O(n)$

$T\left(\frac{7n}{10}\right)$

}

$= T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10}\right) + O(n)$

Worst case linear time selection

Assume $T(n') \leq cn'$ for some constant c and every $n' < n$,

and solve $T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$

Then, $T(n) \leq c \left\lceil \frac{n}{5} \right\rceil + c \left(\frac{7n}{10} + 6\right) + O(n) \leq c \left(\frac{n}{5} + 1\right) + c \left(\frac{7n}{10} + 6\right) + an.$

Therefore, $T(n) \leq cn - \left(\frac{cn}{10} - 7c - an\right)$

Worst case linear time selection

Assume $T(n') \leq cn'$ for some constant c and every $n' < n$,

and solve $T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$

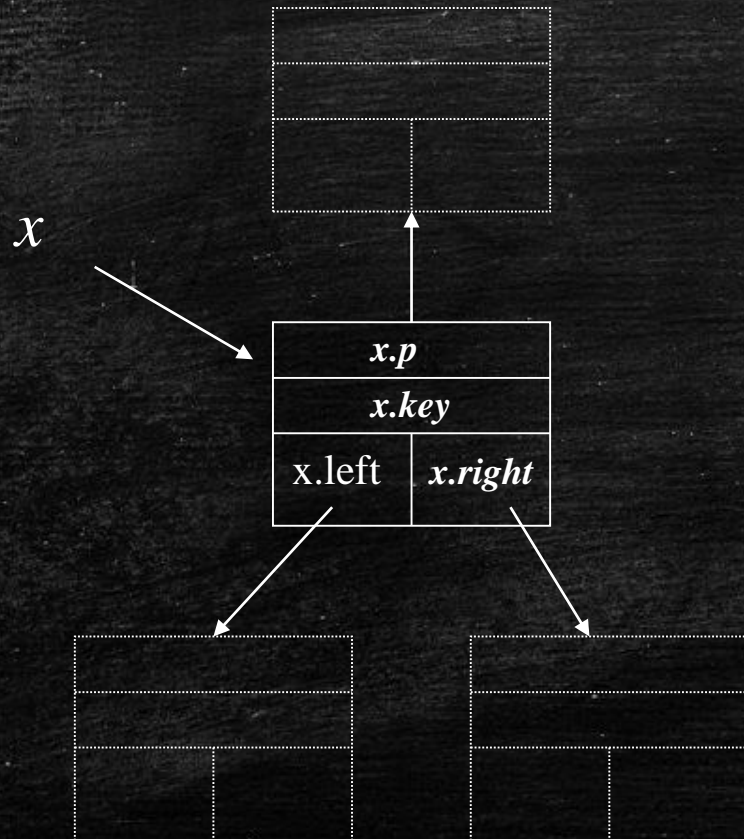
Then, $T(n) \leq c \left\lceil \frac{n}{5} \right\rceil + c \left(\frac{7n}{10} + 6\right) + O(n) \leq c \left(\frac{n}{5} + \frac{7n}{10} + 6\right) + an.$

for $c \geq 20a$
and $n \geq 140$.

Therefore, $T(n) \leq cn - \left(\frac{cn}{10} - 7c - an\right) \leq cn$

Binary Search Trees

- Each node x in a binary search tree (BST) contains:



- $x.key$ - The value stored at x .
- $x.left$ - Pointer to left child of x .
- $x.right$ - Pointer to right child of x .
- $x.p$ - Pointer to parent of x .

Binary Search Tree Property

- Keys in BST satisfy the following properties:

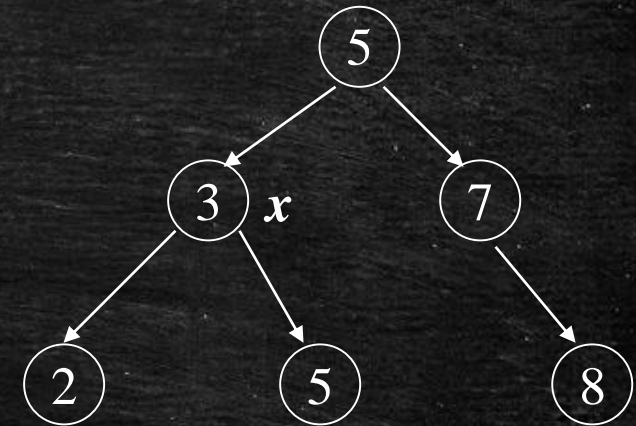
Let x be a node in a BST:

- If y is in the left subtree of x then:

$$y.key \leq x.key$$

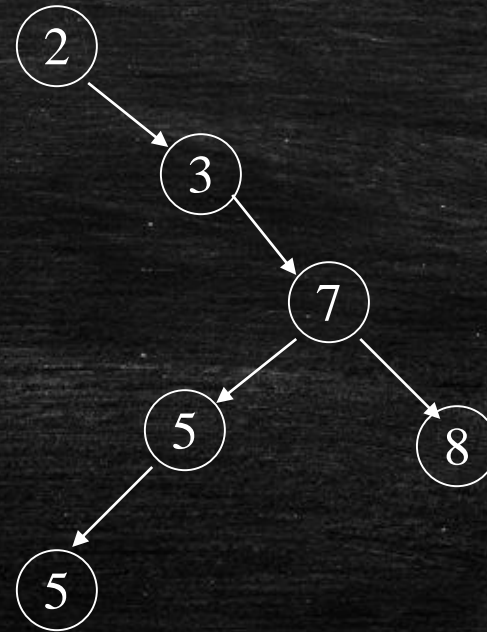
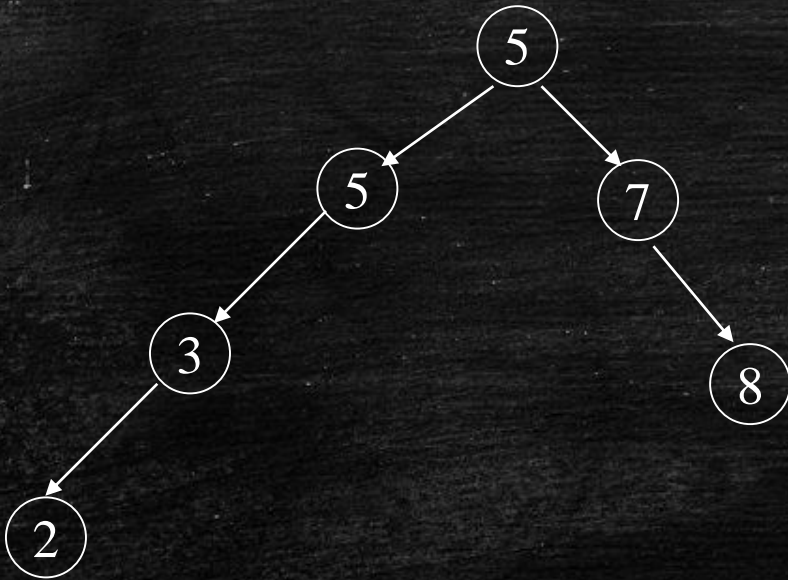
- If y is in the right subtree of x then:

$$y.key > x.key$$



Binary Search Tree Examples

- Two valid BST's for the keys: 2,3,5,5,7,8.



In-Order Tree walk

- Can print keys in BST with in-order tree walk
- Key of each node printed between keys in left and those in right subtrees
- Prints elements in monotonically increasing order

In-Order Traversal

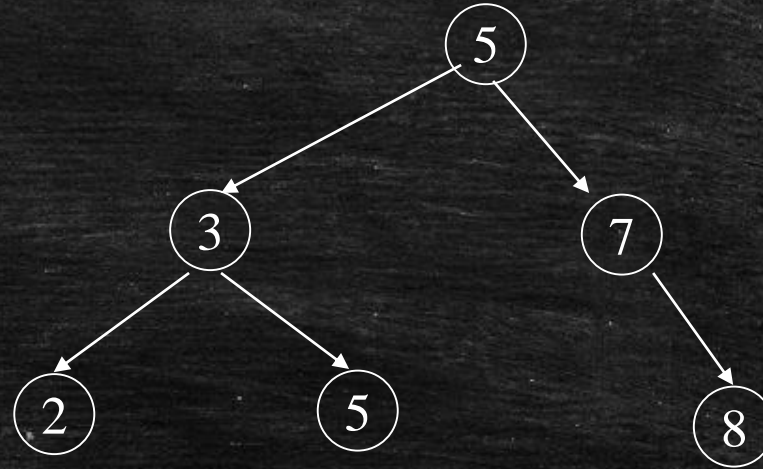
Inorder-Tree-Walk(*x*)

- 1: If *x* \neq *NIL* then
- 2: Inorder-Tree-Walk(*x.left*)
- 3: Print(*x.key*)
- 4: Inorder-Tree-Walk(*x.right*)

-
- What is the recurrence relation for $T(n)$?
 - What is the running time?

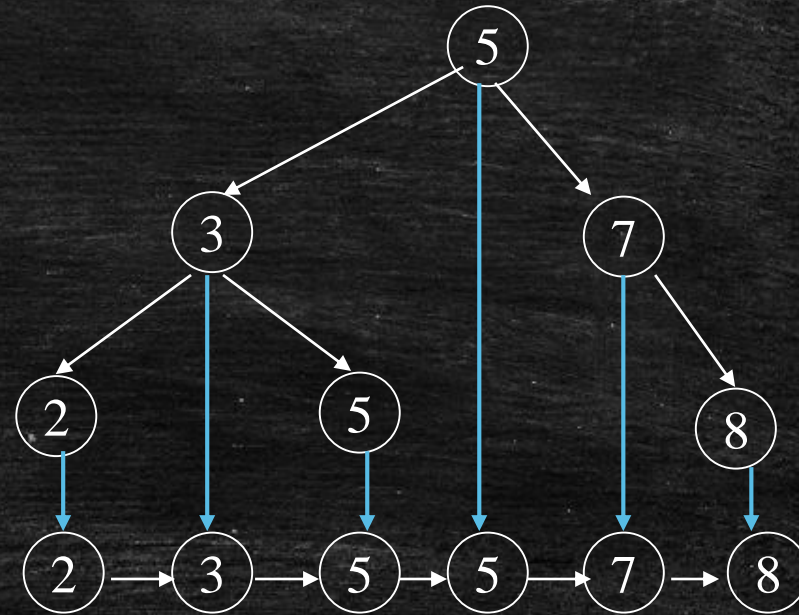
In-Order Traversal

- In-Order traversal can be thought of as a projection of BST nodes on an interval.



In-Order Traversal

- In-Order traversal can be thought of as a projection of BST nodes on an interval.



Other Tree Walks

Preorder-Tree-Walk(x)

- 1: If $x \neq \text{NIL}$ then
- 2: Print($\text{key}[x]$)
- 3: Preorder-Tree-Walk($\text{left}[x]$)
- 4: Preorder-Tree-Walk($\text{right}[x]$)

Postorder-Tree-Walk(x)

- 1: If $x \neq \text{NIL}$ then
- 2: Postorder-Tree-Walk($\text{left}[x]$)
- 3: Postorder-Tree-Walk($\text{right}[x]$)
- 4: Print($\text{key}[x]$)

Searching in BST

- To find element with key k in tree T :
 - Compare k with the root
 - If $k < \text{key}[\text{root}[T]]$ search for k in left subtree
 - Otherwise, search for k in right subtree

Search(T, k)

$x = \text{root}[T]$

if $x == \text{NIL}$

 return("not found")

if $k == \text{key}[x]$

 return("found the key")

if $k < \text{key}[x]$

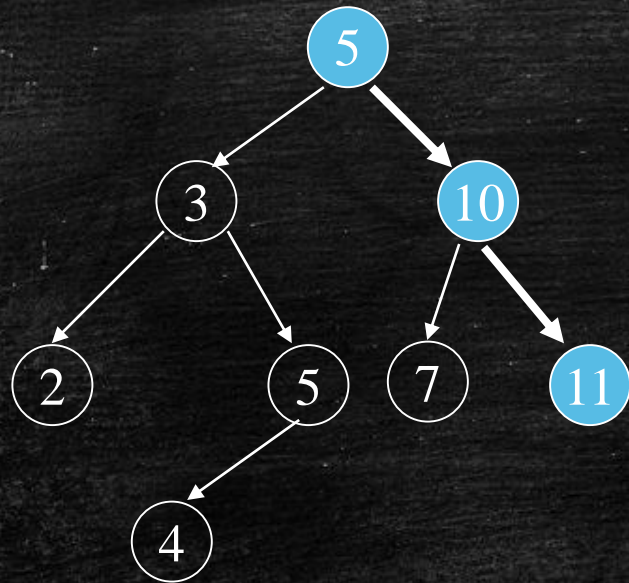
 Search($\text{left}[x], k$)

else

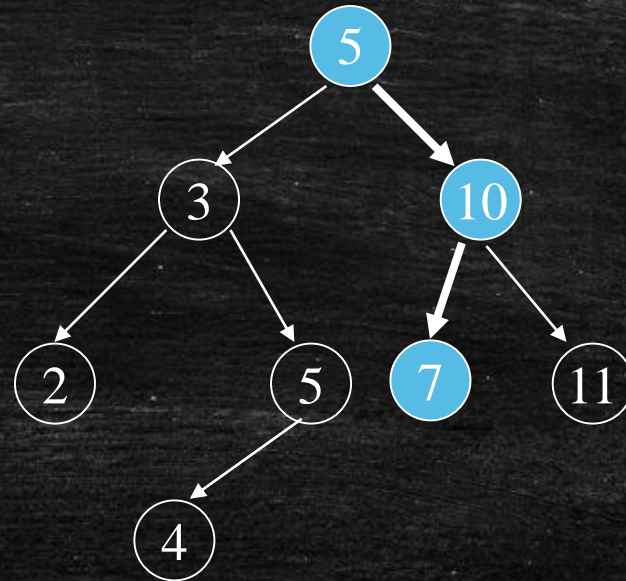
 Search($\text{right}[x], k$)

Examples:

▪ $\text{Search}(T, 11)$

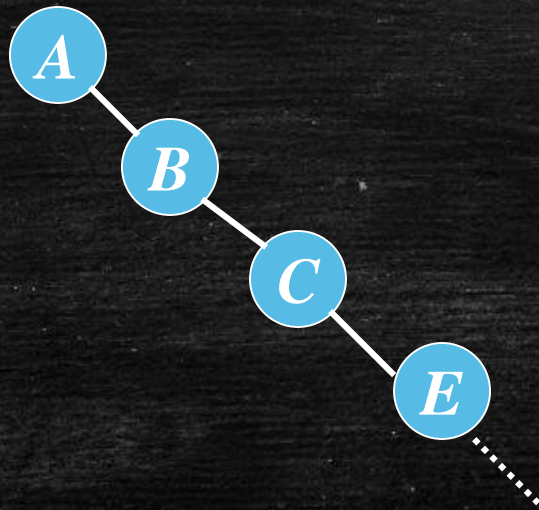


▪ $\text{Search}(T, 6)$



Analysis of Search

- Running time of search for a tree of height h is $O(h)$
- After insertion of n keys, worst case running time of search is $O(n)$



Locating the Minimum

BST-Minimum(T)

```
1:  $x = \text{root}[T]$   
2: while  $\text{left}[x] \neq \text{NIL}$  do  
3:    $x = \text{left}[x]$   
4: return  $x$ 
```

