CSC 411
Design and Analysis of Algorithms

Chapter 5 Divide and Conquer - Part 1

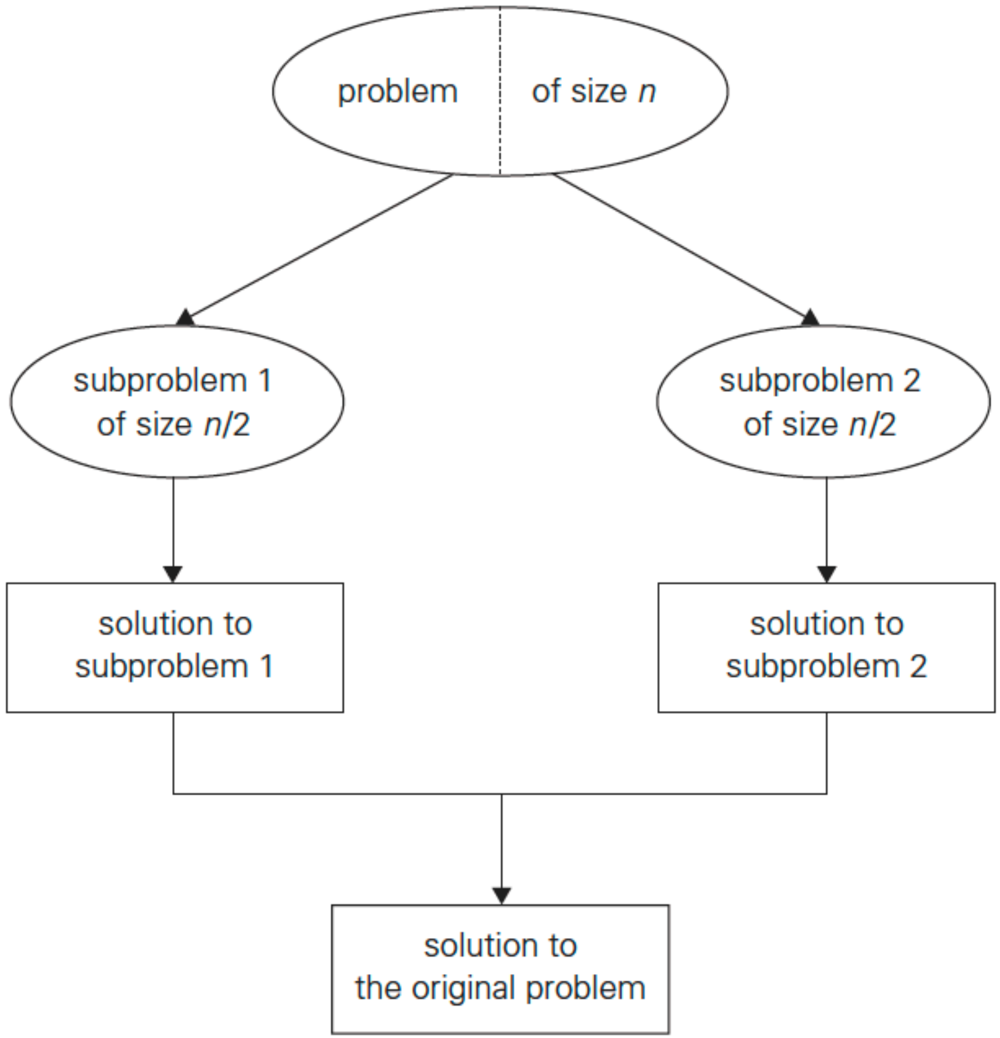
Instructor: Minhee Jun

Divide-and-Conquer

The most-well known algorithm design strategy:

1. A problem is divided into several subproblems of the same type
   - ideally of about equal size.
2. The subproblems are solved .
   - typically recursively, though sometimes a different algorithm is employed, especially when subproblems become small enough.
3. The solutions to the subproblems are combined to get a solution to the original problem

Divide-and-Conquer Technique



General Divide-and-Conquer Recurrence

$$T(n) = aT(n/b) + f(n) \quad \text{where } f(n) \in \Theta(n^d),\ d \geq 0$$

- $aT(n/b)$: sample size n is divided into b instances of size n/b, with a of them needing to be solved.

- $f(n)$: a function that accounts for the time spent on dividing the problem into smaller ones and on combining their solutions.

General Divide-and-Conquer Recurrence

$$T(n) = aT(n/b) + f(n),$$

Master Theorem If $f(n) \in \Theta(n^d)$ where $d \geq 0$ in recurrence (5.1), then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Analogous results hold for the O and Ω notations, too.

- Example: $T(n) = 2T(n/2) + 1$?
  - $a = 2,\ b = 2,\ d = 0 \rightarrow T(n) \in \Theta(n)$

See Appendix B
Page 486

Exercise 3.1-5

5. Find the order of growth for solutions of the following recurrences.

a. $T(n) = 4T(n/2) + n$, $T(1) = 1$

b. $T(n) = 4T(n/2) + n^2$, $T(1) = 1$

c. $T(n) = 4T(n/2) + n^3$, $T(1) = 1$

General Divide-and-Conquer Recurrence

$$T(n) = aT(n/b) + f(n),$$
$$f(n) \in \Theta(n^d)$$

- Master Theorem:
  - If $a < b^d$, $\quad T(n) \in \Theta(n^d)$
  - If $a = b^d$, $\quad T(n) \in \Theta(n^d \log_b n)$
  - If $a > b^d$, $\quad T(n) \in \Theta(n^{\log_b a})$

A. $T(n) = 4T(n/2) + n$?
- $a = 4,\ b = 2,\ d = 1 \rightarrow$ case 3 $\quad \Rightarrow T(n) \in \Theta(n^2)$

B. $T(n) = 4T(n/2) + n^2$?
- $a = 4,\ b = 2,\ d = 2 \rightarrow$ case 2 $\quad \Rightarrow T(n) \in \Theta(n^2 \log_2 n)$

C. $T(n) = 4T(n/2) + n^3$?
- $a = 4,\ b = 2,\ d = 3 \rightarrow$ case 1 $\quad \Rightarrow T(n) \in \Theta(n^3)$

Divide-and-Conquer Examples

- Sorting: mergesort and quicksort (5.1 & 5.2)
- Binary tree traversals (5.3)
- Multiplication of large integers and Matrix multiplication: Strassen's algorithm (5.4)
- Closest-pair and convex-hull algorithms (5.5)

5.1 Mergesort

- Split array A[0..n-1] in two about equal halves and make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into array A as follows:
  - Repeat the following until no elements remain in one of the arrays:
    - compare the first elements in the remaining unprocessed portions of the arrays
    - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
  - Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

Pseudocode of Mergesort

ALGORITHM *Mergesort*$(A[0..n-1])$

//Sorts array $A[0..n-1]$ by recursive mergesort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in nondecreasing order

if $n > 1$

copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$

copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$

Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$

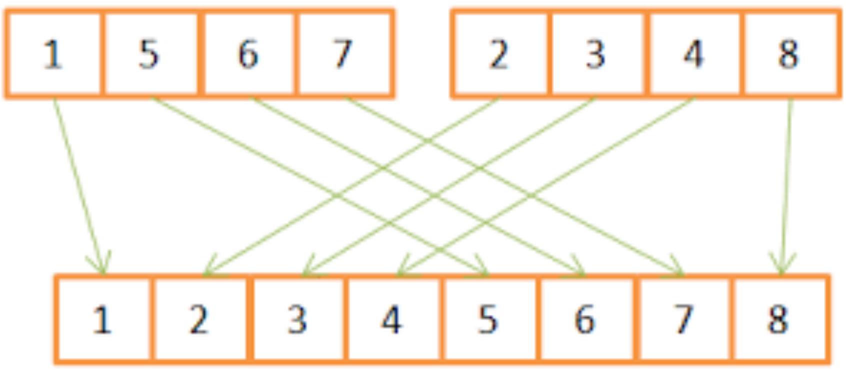
Mergesort$(C[0..\lceil n/2 \rceil - 1])$

Merge(B, C, A) //see below

$$T(n) = T(n/2) + \mathrm{T}(n/2) + \mathrm{T}(\text{merge})$$

Merging Two Sorted Arrarys

- To merge two sorted array into a third (sorted) array, repeatedly **compare the two least elements** and copy the smaller of the two onto the third array.



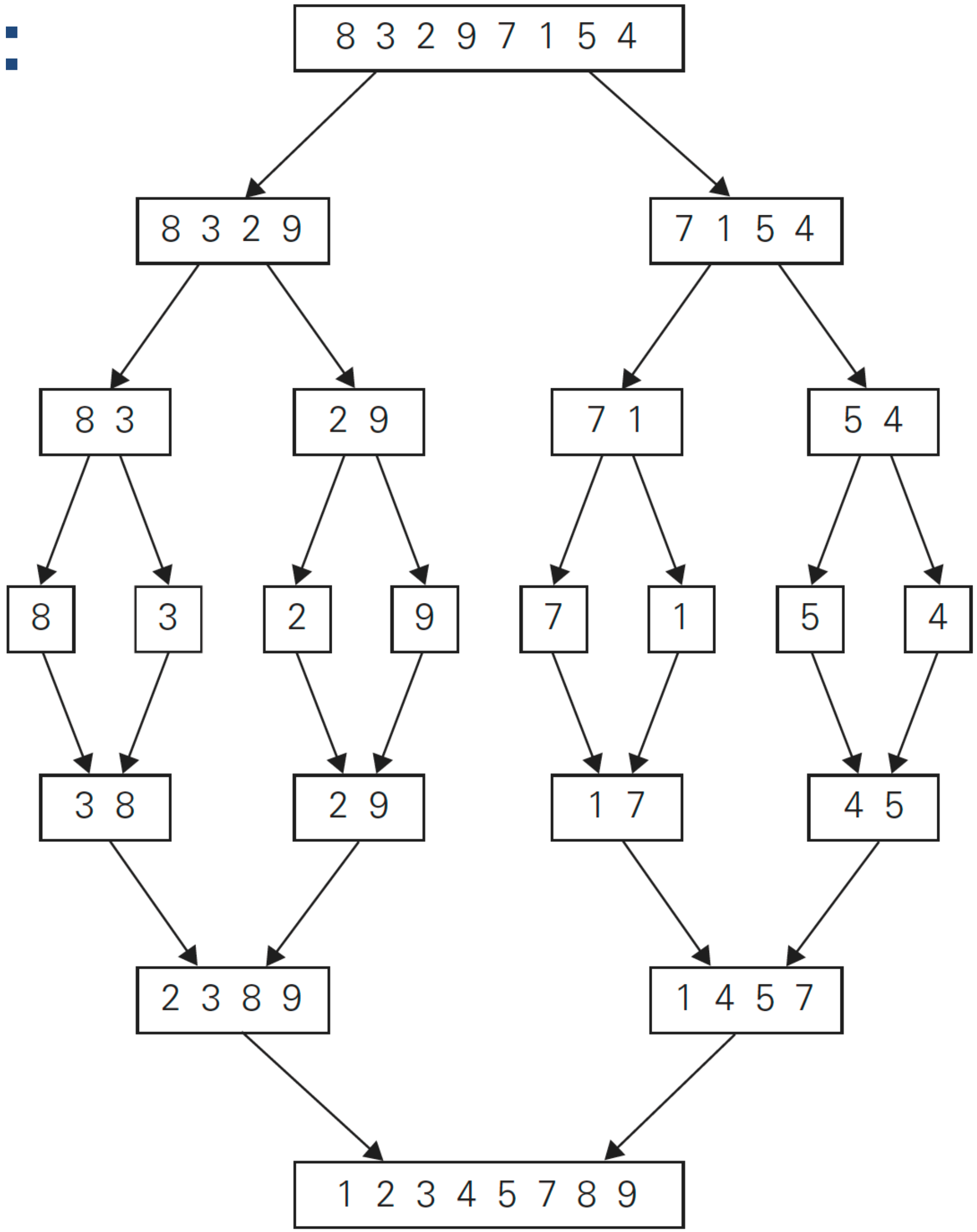
# of comparison:
7 in this example
(1, 2)
(5, 2)
(5, 3)
(5, 4)
(5, 8)
(6, 8)
(7, 8)

Pseudocode of Merge

ALGORITHM $Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0;\ \ j \leftarrow 0;\ \ k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
  if $B[i] \leq C[j]$
    $A[k] \leftarrow B[i];\ \ i \leftarrow i + 1$
  else $A[k] \leftarrow C[j];\ \ j \leftarrow j + 1$
  $k \leftarrow k + 1$
if $i = p$
  copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Example:



8 3 2 9 7 1 5 4
8 3 2 9
7 1 5 4
8 3
2 9
7 1
5 4
8
3
2
9
7
1
5
4
3 8
2 9
1 7
4 5
2 3 8 9
1 4 5 7
1 2 3 4 5 7 8 9

Analysis of Mergesort

- How efficient is mergesort?

$$C(n) = 2C(n/2) + C_{merge}(n) \quad \text{for } n > 1, \quad C(1) = 0.$$

- Worst case:
  - neither of the two arrays becomes empty before the other one contains just one element (e.g., smaller elements may come from the alternating arrays).

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0.$$

  - Master's Theorem

$$C_{worst}(n) \in \Theta(n \log n)$$

Exercise 5.1-6

6. Apply mergesort to sort the list E, X, A, M, P, L, E in alphabetical order.

5.2 Quicksort

- Select a ***pivot*** (partitioning element) – here, the first element

$$\underbrace{\boxed{\;p\;\big|\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}}_{A[i]}$$

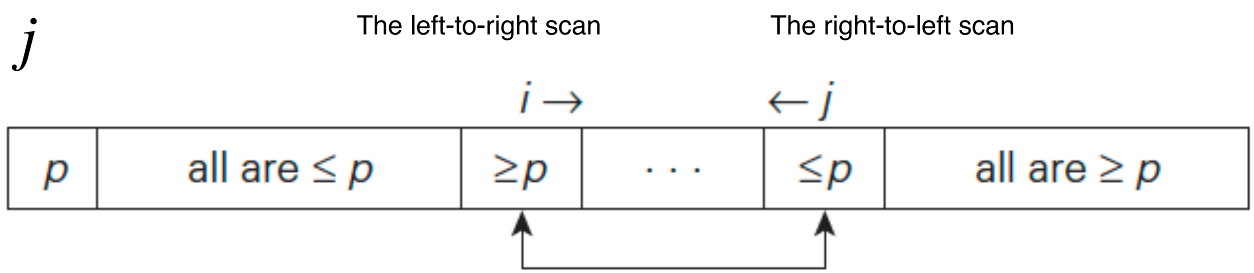
- Rearrange the list so that

$$\underbrace{A[0]\dots A[s-1]}_{\text{all are } \le A[s]}\quad A[s]\quad \underbrace{A[s+1]\dots A[n-1]}_{\text{all are } \ge A[s]}$$

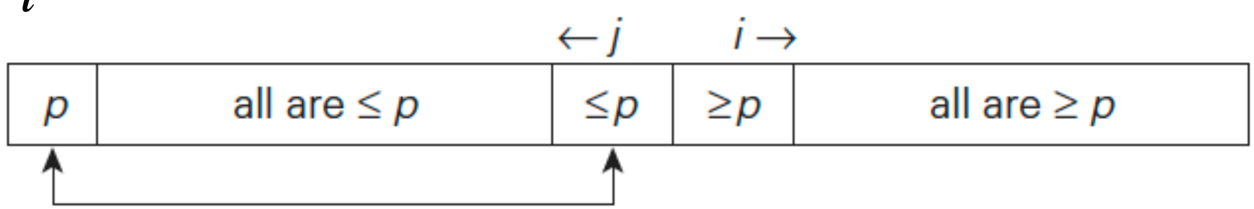
  - all the elements in the first s positions are smaller than or equal to the pivot
  - all the elements in the remaining $n - s$ positions are larger than or equal to the pivot
- Sort the two subarrays recursively

5.2 Quicksort

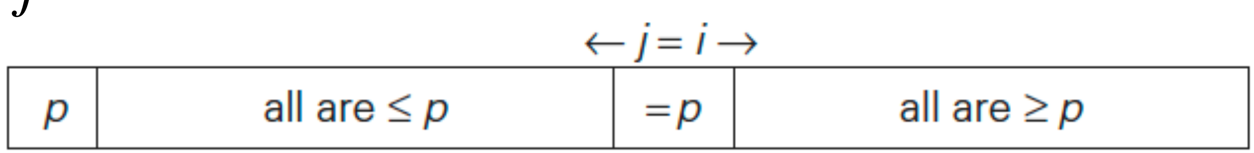
- $i < j$



- $j < i$



- $i = j$



Pseudocode of Partitioning Algorithm

ALGORITHM *HoarePartition*$(A[l..r])$

//Partitions a subarray by Hoare's algorithm, using the first element
//   as a pivot
//Input: Subarray of array $A[0..n-1]$, defined by its left and right
//   indices l and r $(l < r)$
//Output: Partition of $A[l..r]$, with the split position returned as
//   this function's value
$p \leftarrow A[l]$
$i \leftarrow l;\ j \leftarrow r+1$
repeat
    repeat $i \leftarrow i+1$ **until** $A[i] \geq p$
    repeat $j \leftarrow j-1$ **until** $A[j] \leq p$
    swap$(A[i], A[j])$
until $i \geq j$
swap$(A[i], A[j])$ //undo last swap when $i \geq j$
swap$(A[l], A[j])$
return j

Pseudocode of Quicksort

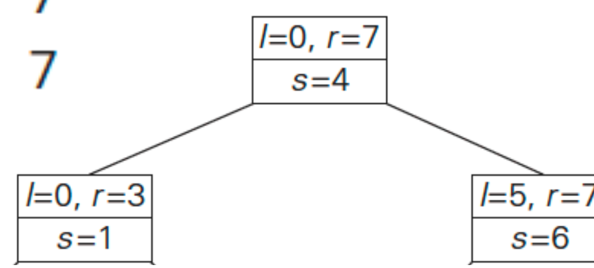
ALGORITHM $Quicksort(A[l..r])$

    //Sorts a subarray by quicksort
    //Input: Subarray of array A[0..n − 1], defined by its left and right
    //   indices l and r
    //Output: Subarray A[l..r] sorted in nondecreasing order
    if l < r
        s ← Partition(A[l..r]) //s is a split position
        Quicksort(A[l..s − 1])
        Quicksort(A[s + 1..r])

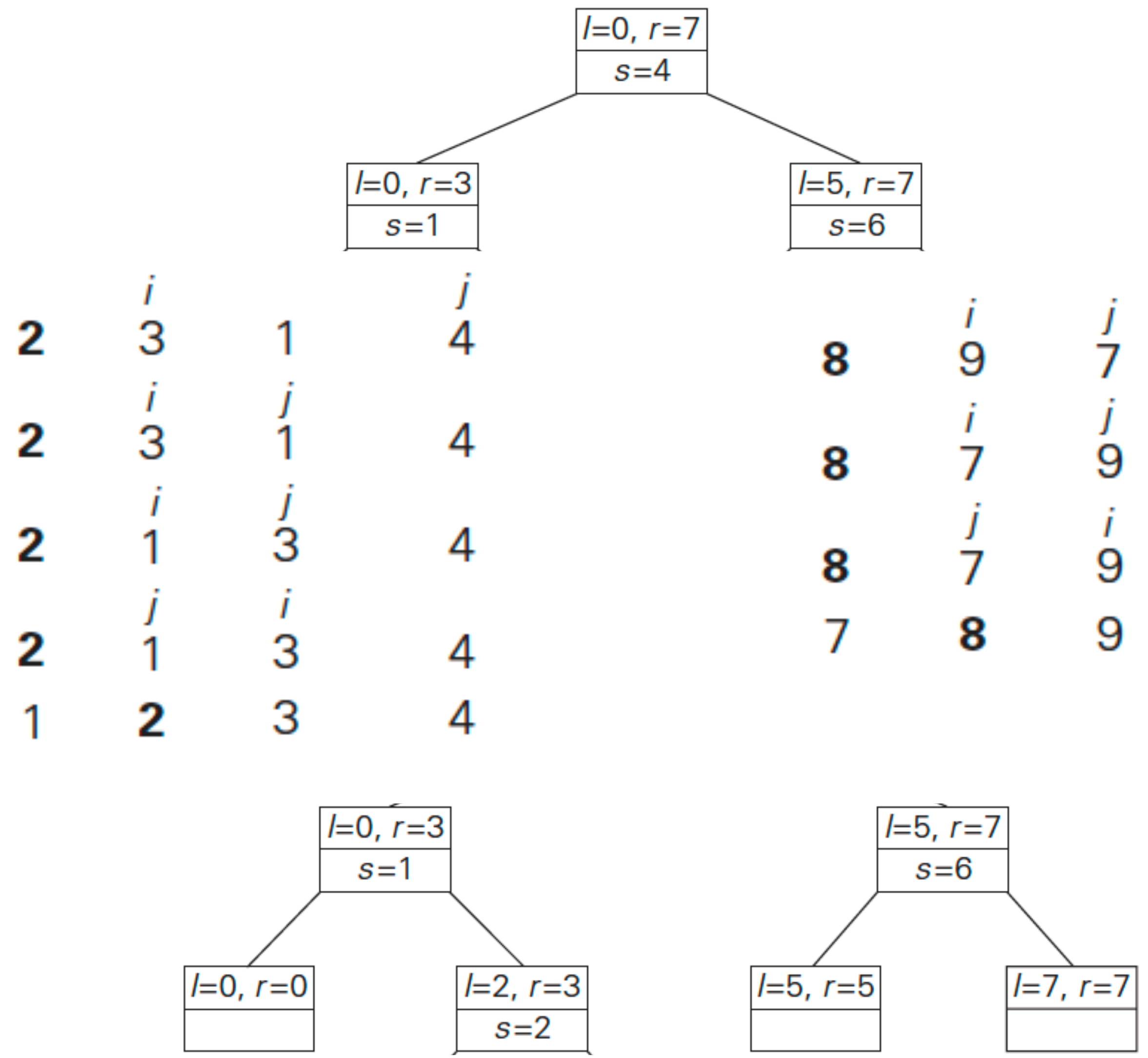
What is the best case?
What is the worst case?

Quicksort Example

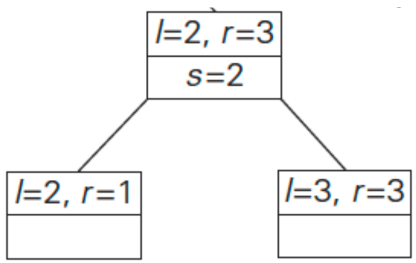
0	1	2	3	4	5	6	7
5	$\overset{i}{3}$	1	9	8	2	4	$\overset{j}{7}$
5	3	1	$\overset{i}{9}$	8	2	$\overset{j}{4}$	7
5	3	1	$\overset{i}{4}$	8	2	$\overset{j}{9}$	7
5	3	1	4	$\overset{i}{8}$	$\overset{j}{2}$	9	7
5	3	1	4	$\overset{i}{2}$	$\overset{j}{8}$	9	7
5	3	1	4	$\overset{j}{2}$	$\overset{i}{8}$	9	7
2	3	1	4	**5**	8	9	7

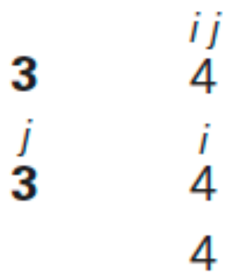


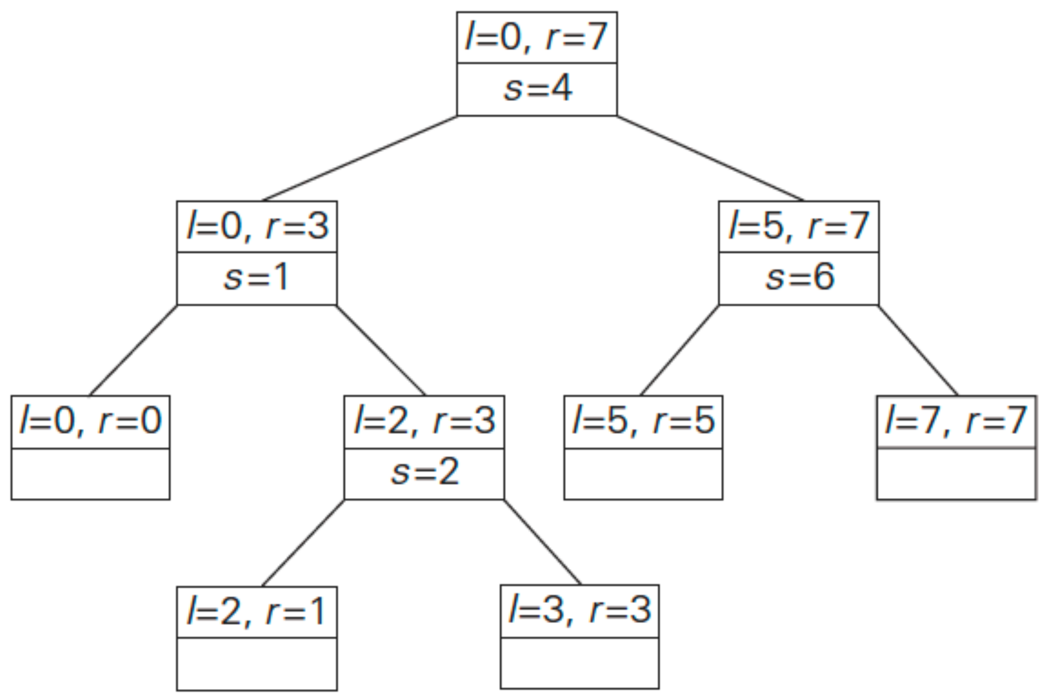
Quicksort Example



Quicksort Example







Quicksort Example

5 3 1 9 8 2 4 7

pivot

$\boxed{2\ 3\ 1\ 4}$ 5 $\boxed{8\ 9\ 7}$

the best case

1 3 5 9 8 2 4 7

pivot

1 $\boxed{3\ 5\ 9\ 8\ 2\ 4\ 7}$

the worse case

Analysis of Quicksort

- Best case: split in the middle - $\Theta(n \log n)$

$$C_{best}(n) = 2C_{best}(n/2) + n \quad \text{for } n > 1, \quad C_{best}(1) = 0.$$

$$C_{best}(n) \in \Theta(n \log_2 n)$$

- Worst case: sorted array! - $\Theta(n^2)$

$$C_{worst}(n) = (n+1) + n + \cdots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2).$$

- Average case: random arrays - $\Theta(n \log n)$

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0.$$

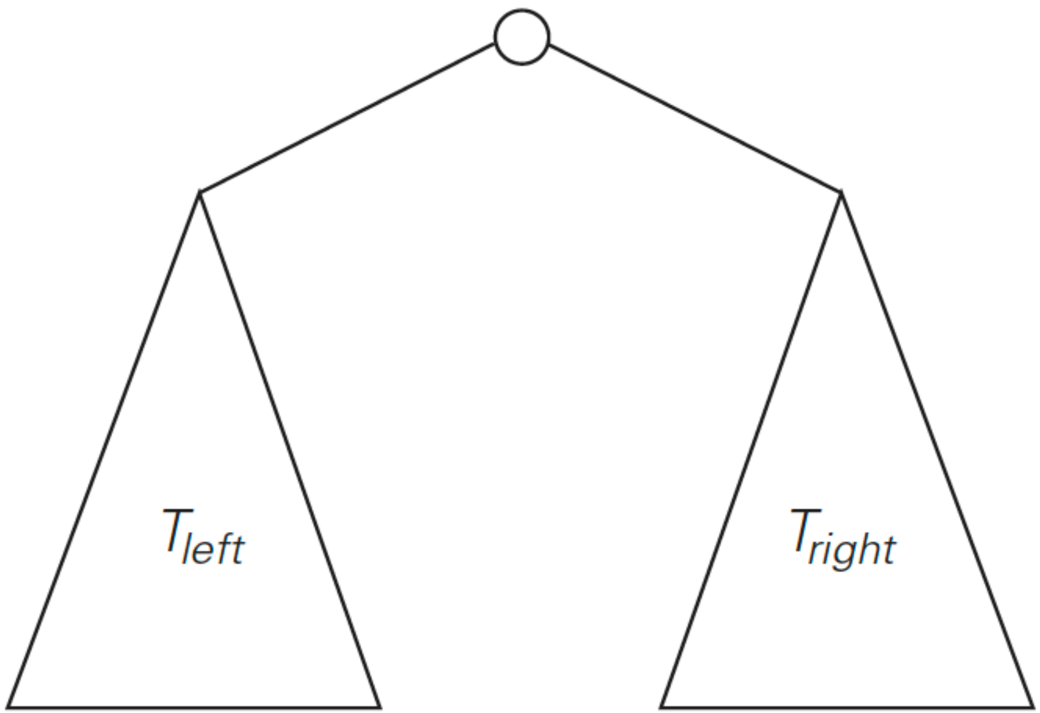
$$C_{avg}(n) \approx 2n \ln n \approx 1.39n \log_2 n.$$

Exercise 5.2-1

1. Apply quicksort to sort the list E, X, A, M, P, L, E in alphabetical order. Draw the tree of the recursive calls made.

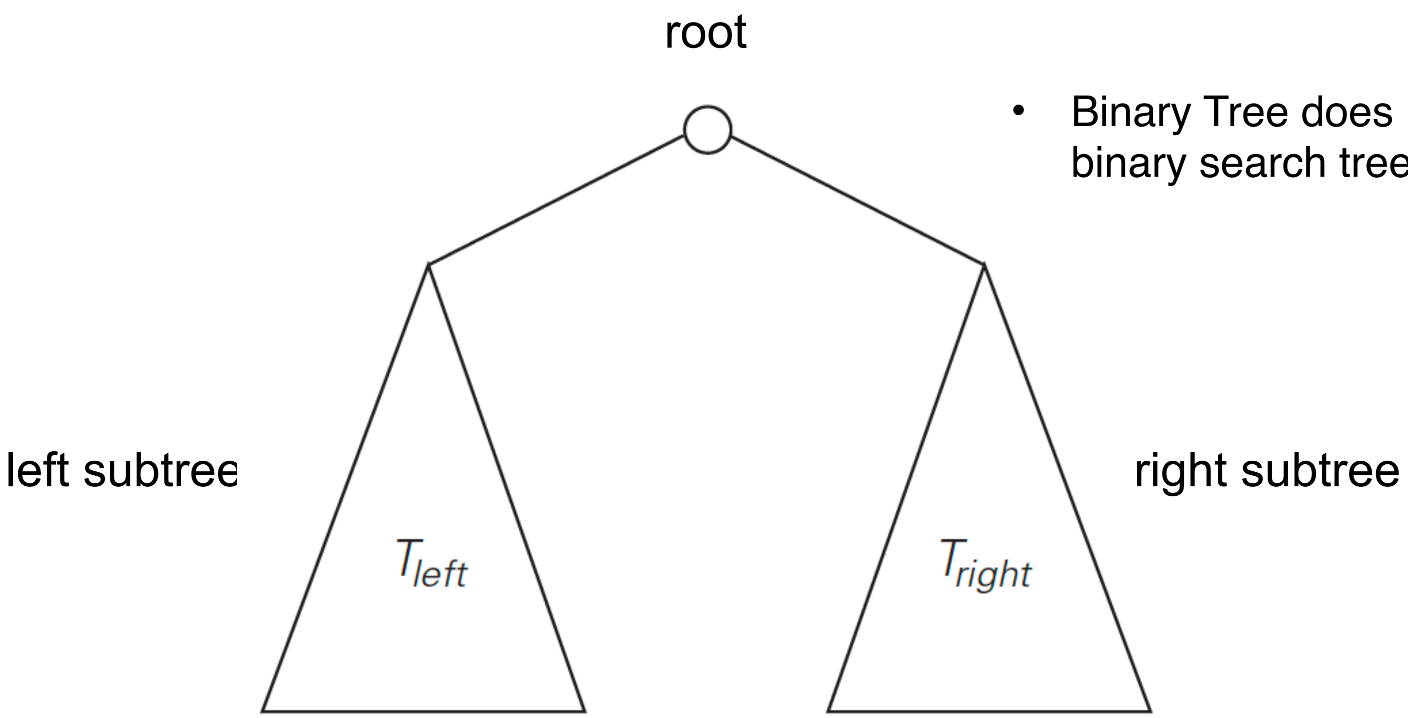
5.3 Binary Search

- A binary tree T
  - a finite set of nodes that is either empty or consists of a root and two disjoint binary trees T_L and T_R called, respectively, the left and right subtree of the root.



Binary Tree Algorithms

- The definition itself divides a binary tree into two smaller structures of the same type
- Binary tree is a divide-and-conquer ready structure!
- Three parts: root, left subtree, and right subtree



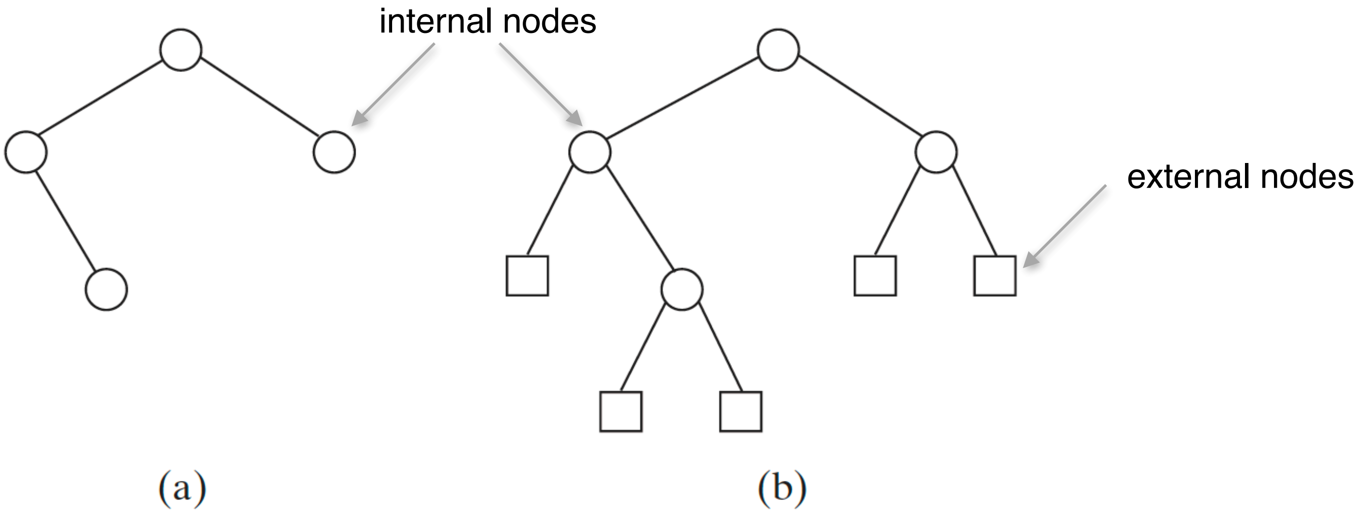
- Binary Tree does not imply binary search tree

Example: Binary Tree Algorithm

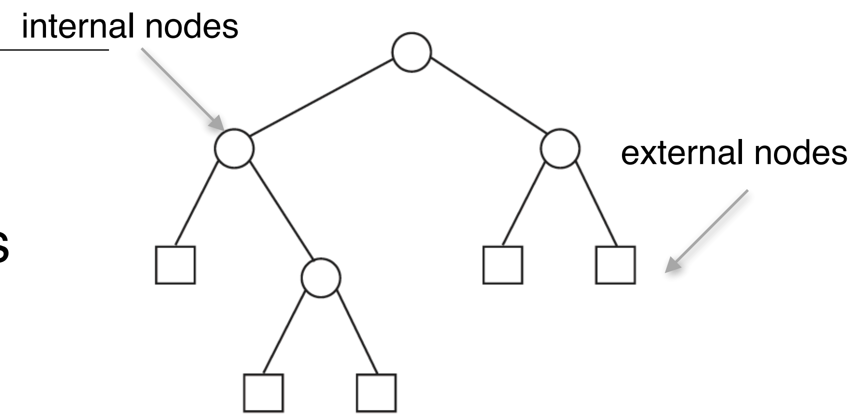
- Height of Binary Tree

ALGORITHM *Height*(T)

//Computes recursively the height of a binary tree
//Input: A binary tree T
//Output: The height of T
if $T = \varnothing$ **return** -1
else return $\max\{Height(T_{left}), Height(T_{right})\} + 1$



Example: Binary Tree Algorithm



- $x = n + 1$
  - x : the number of external nodes
  - n : the number of internal nodes

- Proof: the total number of nodes is $2n + 1 = n + x$

- Efficiency? Θ(height)

$$A(n(T)) = A(n(T_{left})) + A(n(T_{right})) + 1 \quad \text{for } n(T) > 0,$$

$$A(0) = 0. \qquad\qquad A(n) = n.$$

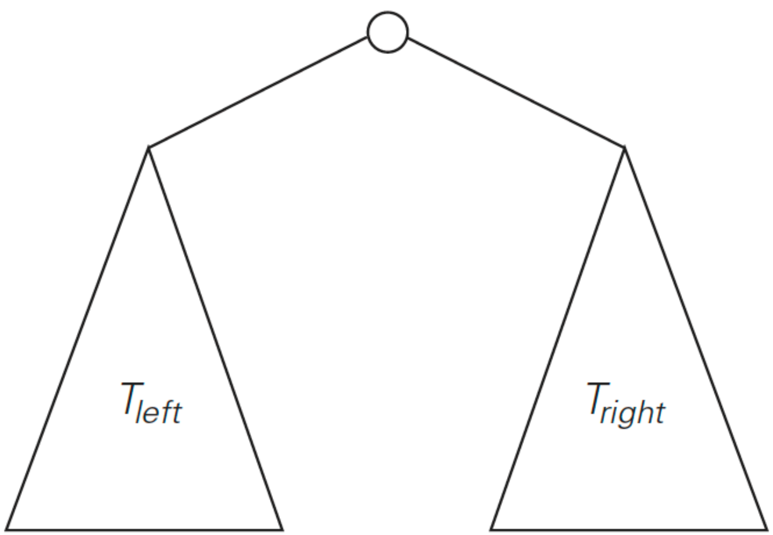
$$C(n) = n + x = 2n + 1,$$

Binary Search

- Very efficient algorithm for searching in sorted array:

  K vs A[0] ….. A[m] ….. A[n-1]

  - If K = A[m], stop (successful search);
  - otherwise,
    - If K < A[m], continue searching by the same method in A[0..m-1]
    - If K > A[m], in A[m+1..n-1]



$l \leftarrow 0$; $r \leftarrow n-1$

while $l \leq r$ do

$m \leftarrow \lfloor (l + r)/2 \rfloor$

if K = A[m] return m

else if K < A[m] $r \leftarrow m - 1$

else $l \leftarrow m+1$

return -1

Exercise 5.3-2

2. The following algorithm seeks to compute the number of leaves in a binary tree.

ALGORITHM *LeafCounter(T)*

//Computes recursively the number of leaves in a binary tree
//Input: A binary tree T
//Output: The number of leaves in T
if $T = \varnothing$ **return** 0
else return $LeafCounter(T_{left}) + LeafCounter(T_{right})$

Is this algorithm correct? If it is, prove it; if it is not, make an appropriate correction.

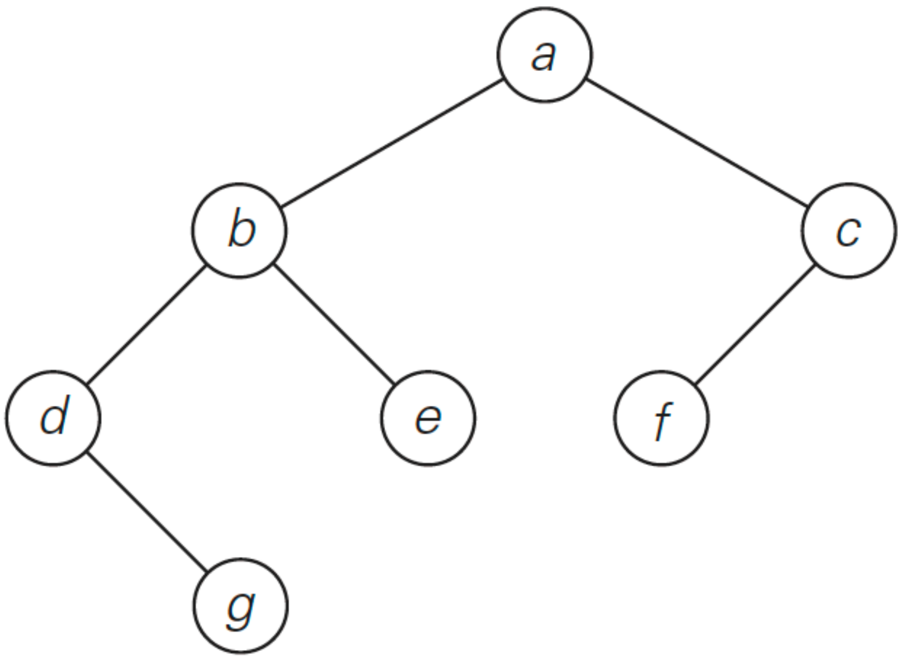
Binary Tree Traversals

The most important divide-and-conquer algorithms for binary trees are the three classic traversals: preorder, inorder, and postorder.

- **Preorder**: root, T_L, and T_R
  - the root is visited before the left and right subtrees are visited
- **Inorder**: T_L, root and T_R
  - the root is visited after visiting its left subtree but before visiting its right subtree
- **Postorder**: T_L, T_R and root
  - the root is visited after the left and right subtrees are visited

Classic traversals

- Preorder: root, TL, and TR
- Inorder: TL, root and TR
- Postorder: TL, TR and root



preorder: *a, b, d, g, e, c, f*
inorder: *d, g, b, e, a, f, c*
postorder: *g, d, e, b, f, c, a*

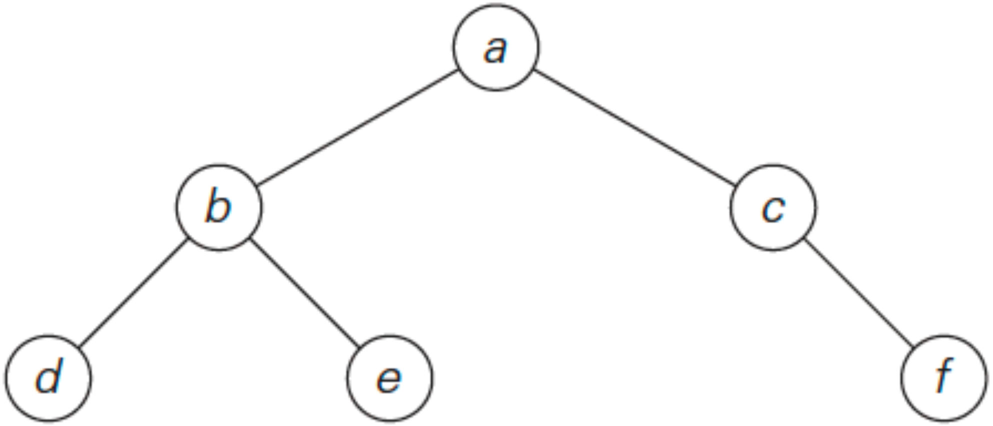
Exercise 5.3-5

5. Traverse the following binary tree

a. in preorder.

b. in inorder.

c. in postorder.



Exercise 5.3-6

6. Write pseudocode for one of the classic traversal algorithms (preorder, inorder, and postorder) for binary trees. Assuming that your algorithm is recursive, find the number of recursive calls made.