Module 2
Structure of Algorithms

S. Lakshmivarahan
School of Computer Science
University of Oklahoma
USA-73019
Varahan@ou.edu

Structure of Algorithms

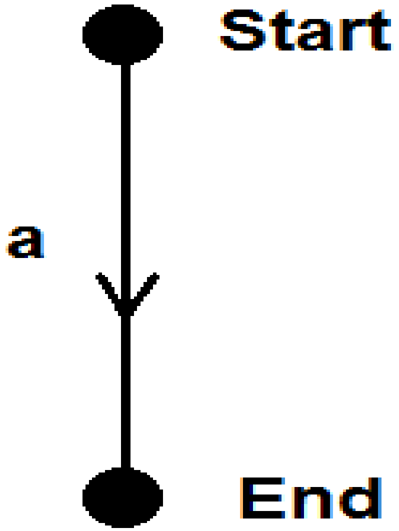
- Some algorithms have a conditional branching arising out of data dependent computations, while others do not have any branching at all
- Given this dichotomy of algorithms, there is a need to refine the concept of the time complexity function T(n) defined earlier

Structure of Algorithms

- To develop an appreciation for this need, we first consider the structure of typical algorithms or programs

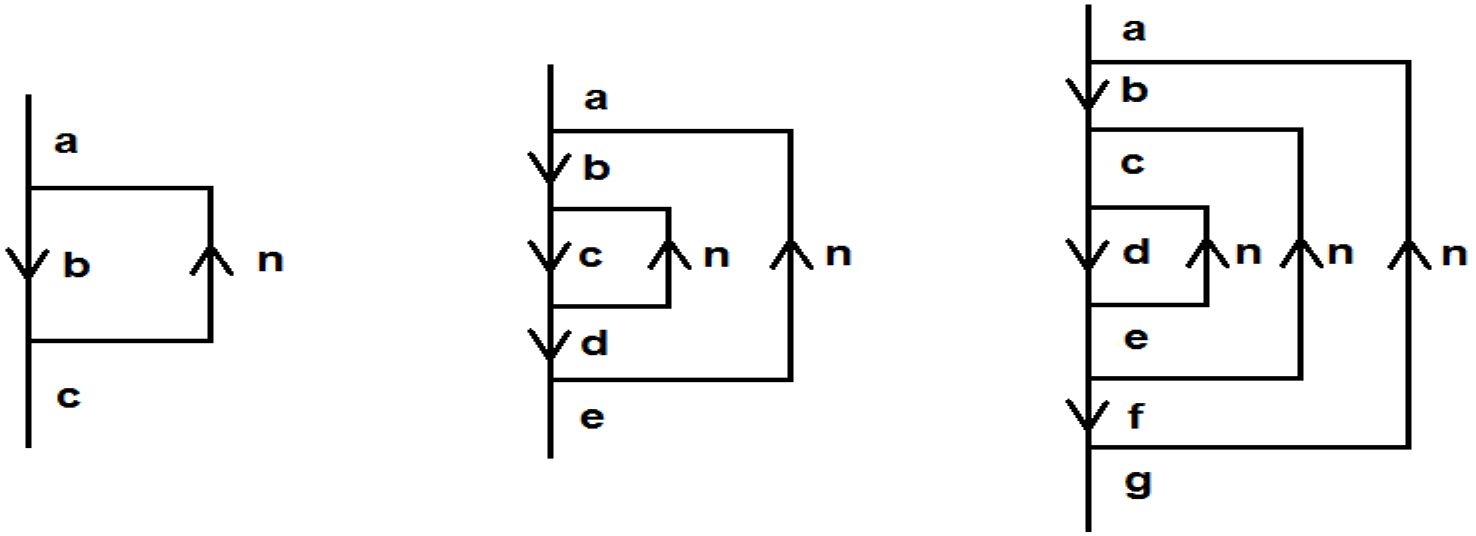
- There are 3 basic structures:
    1. No repetition or branching
    2. Repetition but no branching
    3. Repetition with branching

1) No repetition and no branching



- In this case, the program starts and performs a fixed number of operations
- Let **a** be the total number of arithmetic operations performed by this program
- T(n) = a, a constant

2) Repetitions but no branching



Single Loop
Case I

Double-nested Loops
Case II

Triple-nested Loops
Case III

2) Repetitions but no branching

- In here, a, b, c, … denote the number of operations performed in each segment of the program

<u>Case I</u>

$$T(n) = (a + c) + bn$$

Linear in n

<u>Case II</u>

$$T(n) = (a + e) + (b + d)n + cn^2$$

Quadratic in n

<u>Case III</u>

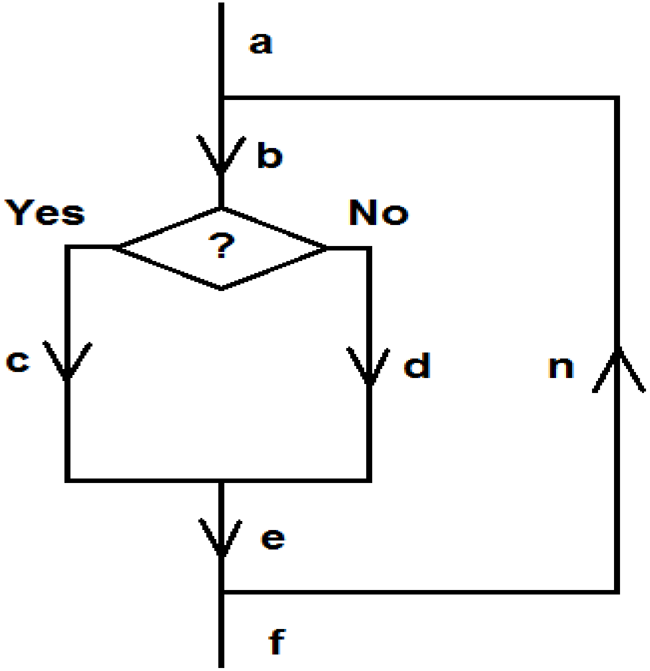
$$T(n) = (a + g) + (b + f)n + (c + e)n^2 + dn^3$$

Cubic in n

2) Repetitions but no branching

- In each of these cases, every instruction is performed for every input. Thus, these programs perform the same number of operations on every input

3) Repetition with conditional branching



- There is a branching embedded in a loop
- It is impossible to predict which of the two branches will be performed a priori
- That is, this program does not perform the same number of operations on all inputs

Time Complexity

- This calls for defining T(n) as the <u>**worst case time**</u> as measured by

$$T(n) = (a + f) + (b + e)n + \max\{c, d\}n$$

- Recall that when there is no branching, every part of the program is performed all the time
- To unify these two cases, from now on we will denote <u>the worst case time as the time complexity</u>

Time Complexity

- Of course, this leads to the notion of the **best case time** as well:

$$T_{B(n)} = (a + f) + (b + e)n + \min\{c, d\}n$$

Time Complexity

- In cases where $T_B(n) << T(n)$, we would often characterize the average case time $T_a(n)$
- Thus, in general,

$$T_B(n) < T_a(n) < T(n)$$

- Understanding of the gap between $T(n)$ and $T_B(n)$ and the computation of $T_a(n)$ is very important for algorithms with conditional branching

Time Complexity

- As an example, when we do sorting we will come across an algorithm called **quick-sort**, for which:

$$c_1 n \log(n) = T_B(n) \leq T_a(n) \leq T(n) = c_2 n^2$$

- where c_1 and c_2 are positive constants. The gap

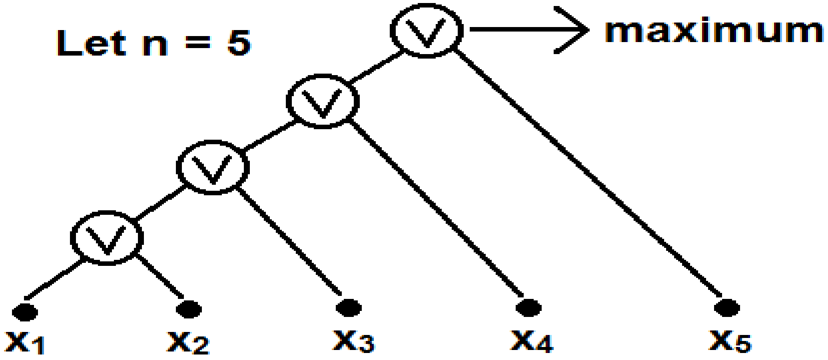
$$T(n) - T_B(n) = c_2 n^2 - c_1 n \log(n)$$

increases with n

Selection Problems

- Given an array of n items (for instance, integers) $x_1, x_2, \ldots, x_n$, find
  - Maximum
  - Minimum
  - Average or mean
  - Variance
  - Median
  - K^{th} percentile
  - Etc.

Maximum/Minimum



$x_1 \vee x_2 = \max\{x_1, x_2\}$

$x_1 \wedge x_2 = \min\{x_1, x_2\}$

- This is a binary tree with n leafs and (n-1) internal nodes, where the operation V is performed

Maximum/Minimum

- Hence, the time T(n) to find the maximum of n numbers is: $T(n) = n - 1$
- By changing V to Λ we get the minimum
- If $x_1 = x_2$, then $x_1 V\, x_2 = x_1$, the least indexed item

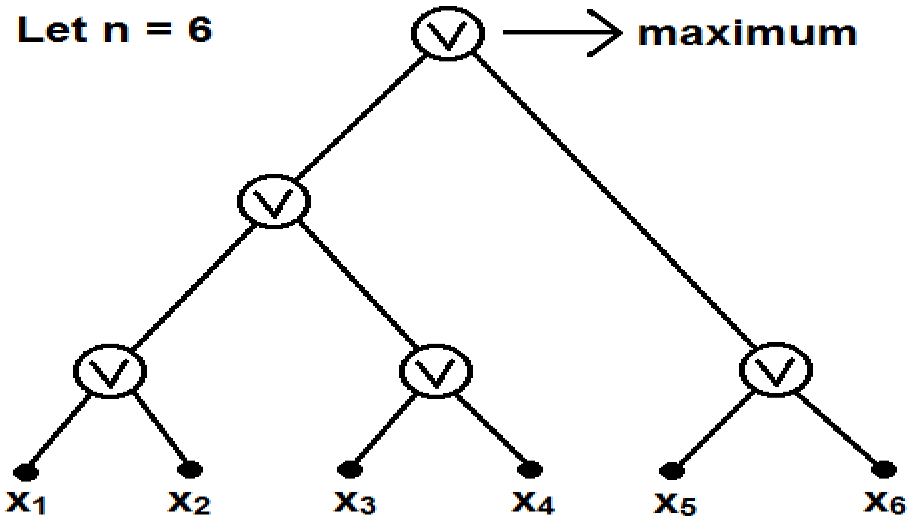
Maximum/Minimum

- **Lower Bound**: We cannot find the minimum in less than (n-1) operations
- Proof:
  - Let n=3 and we would need 2 operations to find the max or min. Assume we can do it in 1 operation, say by comparing x_1 and x_2 and let $x_2 = x_1 \vee x_2$
  - If we declare x_2 is the max, then we will have a correct answer only for those x_3 such that $x_3 < x_2$
  - Thus, for those $x_3 > x_2$ this claim would be false

Maximum/Minimum

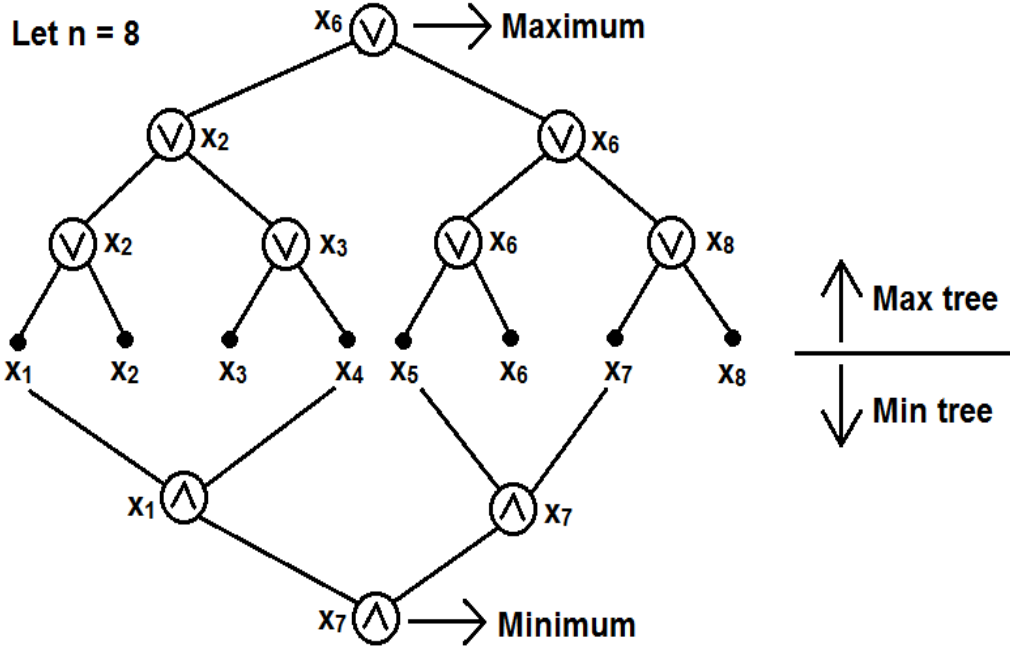
- Recall, an algorithm by definition should work correctly for all inputs. Hence, two operations are necessary
- Hence, the algorithm defined by the binary tree is called **optimal** since we cannot reduce the time

Other Equivalent Tree Structures



- When $n = 2^k$, this tree is balanced and is a complete binary tree
- When $n \neq 2^k$, search as above, the binary tree is not a complete binary tree

Simultaneous max and min



- Maximum is the ultimate winner and minimum is the ultimate loser
- $T(n) = (n-1) + \left(\frac{n}{2} - 1\right) = \frac{3n}{2} - 2$

Homework

2.1. Draw the various forms of the Binary Tree on n=4, 6, and 8 nodes

2.2. Compute the total number of operations to find the following

- $\bar{x}$, sample mean/average = $\frac{1}{n}\sum_{i=1}^{n} x_i$
- s^2, sample variance = $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$

Homework

2.3. Develop an algorithm to find the second and third maximums efficiently

2.4. Let n be odd and develop an algorithm to find the $\frac{n}{2}$ th maximum which is called the median. What is the complexity? What happens when n is even?

Search Problems

- Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of items/numbers in the file. Let Y be the input query
- The search problem seeks an answer to the question: does Y belong to X?
- This problem is quite basic and occurs many times in our daily life- when we pay the electricity bill, search for a book in a library, look for a spare part for a 1950 Chevy, pay your tuition fee at the Bursar’s office, etc.

Search Problems

- The answer is either yes or no
- We declare "YES" at the instant when Y is found, but to say "NO", one must have searched the entire file and not have found Y in X
- Thus, there is an **inherent asymmetry** as measured by the amount of work in terms of number of comparisons needed to declare "YES" or "NO"

Search Problems

- There are two types of algorithms depending on whether the file X is sorted or not:

1. Sequential Search
2. Binary Search

Case A: Sequential Search

- It is assumed that the file X is <u>not</u> sorted.
- The search compares Y with x_1 and the answer is "YES" if equal, else Y is compared with x_2 and declares "YES" if equal, …, so on until Y is compared with x_n. If equal, we declare "YES", otherwise, we say "NO"
- Thus, we can say "NO" only after n comparisons- comparing Y successively with x_1 through x_n and not getting a "YES" answer

Sequential Search

- On the other hand, a "YES" answer can occur after 1,2,...,n comparisons
- Notice that this algorithm involves data dependent comparison and hence

$$T(n) = n$$

is the worst case

Sequential Search

- The best case is when we get "YES" after the first comparison
- Since there is a wide gap between the best and worst cases, we often compute the average case

Sequential Search: Average Case Complexity

- Recall that "YES" can occur in n ways and "NO" can occur in one way, giving raise to (n+1) distinct events
- In the absence of any prior information, it is natural to assume that these events are equally likely

Sequential Search: Average Case Complexity

- Recall: $T_a(n) = \sum_{i=1}^{n} \begin{Bmatrix} probability\ of \\ the\ event\ i \end{Bmatrix} * \begin{Bmatrix} work\ done\ when \\ this\ event\ occurs \end{Bmatrix}$

$$= \frac{1}{n+1}[1 + 2 + 3 + \ \dots\ + n + n]$$

$$= \frac{1}{n+1}\sum_{i=1}^{n} i + \frac{n}{n+1}$$

$$= \frac{n(n+1)}{2(n+1)} + \frac{n}{n+1}$$

$$= \frac{n}{2} + 1 \quad \text{(when n is large)}$$

Note: We used the summation formula for arithmetic form:

$$\sum_{i=1}^{n} i = \frac{n\,(n+1)}{2}$$

- I.e. we need to search about half the file on average

Case B: Binary Search

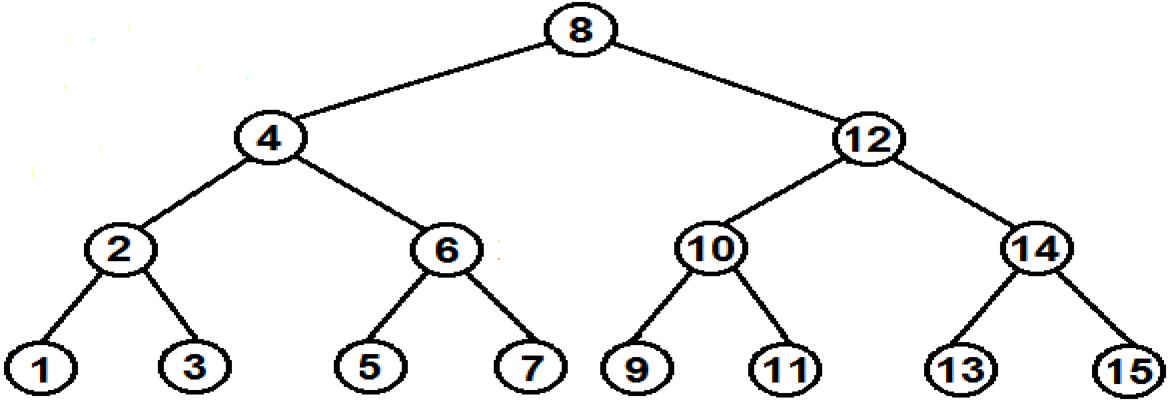
- It is assumed that the file is sorted (in increasing order) and all the items are distinct, that is:

  $$x_1 < x_2 < \dots < x_n$$

- Let us assume that $n = 2^k-1$ for some k. That is, n = 3, 7, 15, 31,…
- For definiteness, let n = 15

Binary Search

- The logic of the search is given by the complete binary tree:



Binary Search

- First, Y is compared with the middle item in the sorted file which is $x_{\lceil \frac{n}{2} \rceil} = x_{\lceil \frac{15}{2} \rceil} = x_8$
- If $Y = x_8$, then "YES“
- If $Y < x_8$, then compare Y with the middle of the left half, which is $x_{\lceil \frac{1}{2} \lceil \frac{n}{2} \rceil \rceil} = x_{\lceil \frac{8}{2} \rceil} = x_4$
- If $Y > x_8$, then compare Y with the middle of the left half, which is $x_{\lceil \frac{3}{2} \lceil \frac{n}{2} \rceil \rceil} = x_{\lceil \frac{24}{2} \rceil} = x_{12}$ and the search continues

Binary Search

- Recall that there are k $= \lceil \log_2 n \rceil = \lceil \log_2 15 \rceil = 4$ levels and a total of $2^k - 1$ nodes in this complete binary tree
- The depth of this tree is k $= 4$
- Thus, "YES" answer can come after 1, 2, ... , $\lceil \log_2 n \rceil =$ k comparisons, but "NO" occurs only after k comparisons

Binary Search

- Hence, the worst case time is

  T(n) $= \lceil \log_2 n \rceil$
- The time for "YES" answer may vary from 1 to $\lceil \log_2 n \rceil$
- It is interesting to quantify the average case complexity

Binary Search: Average Case

- Recall that there are n ways in which a “YES” can occur
- There are (n+1) ways in which “NO” answer can occur:

  $Y < x_1,\ \ x_1 < Y < x_2,\ \ x_2 < Y < x_3,\ \ldots\ ,\ x_n < Y,$

- The total number of events = 2n + 1

Binary Search: Average Case

- We now itemize the number of comparisons and the number of "YES"/"NO" answers. We can get

# of comparison		# of Answers	
1	$\longrightarrow$	1	– YES
2	$\longrightarrow$	2	– YES
3	$\longrightarrow$	$2^2=4$	– YES
i	$\longrightarrow$	2^{i-1}	– YES
k	$\longrightarrow$	2^{k-1}	– YES
k	$\longrightarrow$	(n+1)	– NO

Binary Search: Average Case

- So, the average case complexity can be calculated as follows:

$$T_a(n) = \frac{1}{2n+1}\left[\sum_{i=1}^{k}(i * 2^{i-1}) + k(n+1)\right]$$

$$= \frac{1}{2n+1}\sum_{i=1}^{k}(i * 2^{i-1}) + \frac{k(n+1)}{2n+1} \rightarrow ①$$

Binary Search: Average Case

- Recall that

$$\sum_{i=1}^{k} i2^{i-1} = \sum_{i=1}^{k} i(2^{i} - 2^{i=1}) = \sum_{i=1}^{k} i2^{i} - \sum_{i=1}^{k} i2^{i-1}$$

$$= 1 * 2^{1} + 2 * 2^{2} + 3 * 2^{3} + \cdots + (k-1) * 2^{k-1} + k2^{k}$$

$$-1 * 2^{0} - 2 * 2^{1} - 3 * 2^{2} - \cdots - (k-1) * 2^{k-2} - k2^{k-1}$$

$$= k2^{k} - 1 - [2^{1}(2-1) + \cdots + 2^{k-1}(k-(k-1))]$$

$$= k2^{k} - 1 - [2 + 2^{2} + \ldots + 2^{k-1}]$$

$$= k2^{k} - 1 - \left(\frac{2^{k}-1}{2-1}\right) + 1$$

$$= 2^{k}(k-1) + 1 \rightarrow 2$$

Binary Search: Average Case

- Substituting ② in ①:

$$T_a(n) = \frac{2^k(k-1)+1}{2n+1} + \frac{k(n+1)}{2n+1} \quad \rightarrow ③$$

- Recall, n $= 2^k - 1$
- Substituting and simplifying,

$$T_a(n) \approx k - ½ \approx \log_2 n \qquad \text{as n} \rightarrow \infty$$

Homework

2.5. Simplify ③

2.6. $\lfloor x \rfloor \leq x \leq \lceil x \rceil$

$$k = \lceil \log_2(n+1) \rceil = \lceil \log_2(n) \rceil$$

when $n = 2^k - 1$