Question

The number of nodes a heap of height k can hold is

(A) 2^k to $2^{k+1}- 1$

(B) 2^{k+1} to $2^{k+1}- 1$

(C) 2^{k-1} to $2^{k+1}- 1$

(D) 2^k -1to $2^{k+1}- 1$

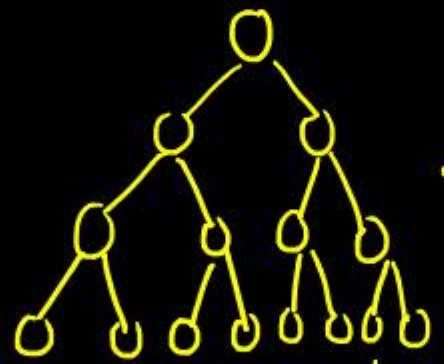
(A)

Binary tree k

Range - Minimum No. of Nodes

Maximum No. of Nodes

k=3



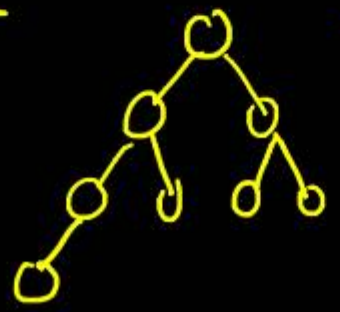
Maximum No. of Nodes

$2^0 + 2^1 + 2^2 + 2^3 = \frac{1(2^4-1)}{2-1}$

$= 2^{3+1} - 1$

$= 2^{k+1} - 1$

$- 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} + 1$

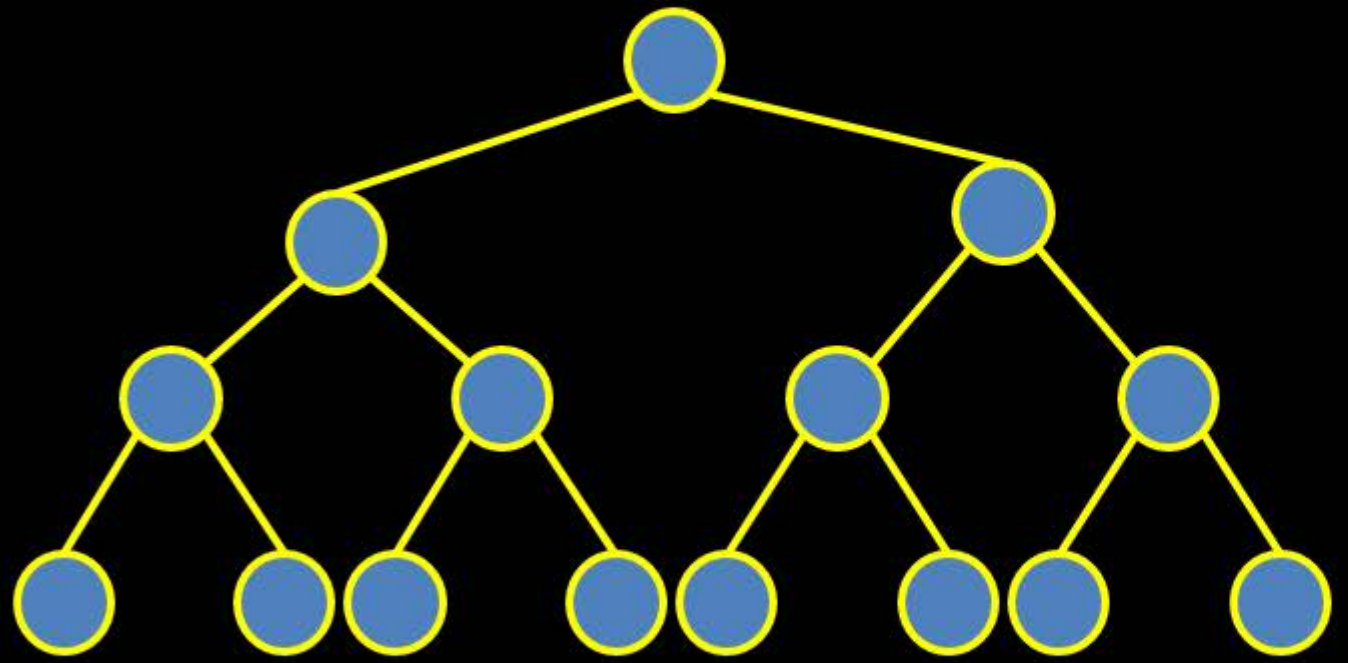


$\frac{1(2^k-1)}{2-1} + 1$

$2^k - 1 + 1 = \boxed{2^k}$

Complete Binary Tree Properties

Maximum Number of Nodes



total No. of Nodes is N

height $2^{h+1} - 1 = N$

$2^{h+1} = N+1$

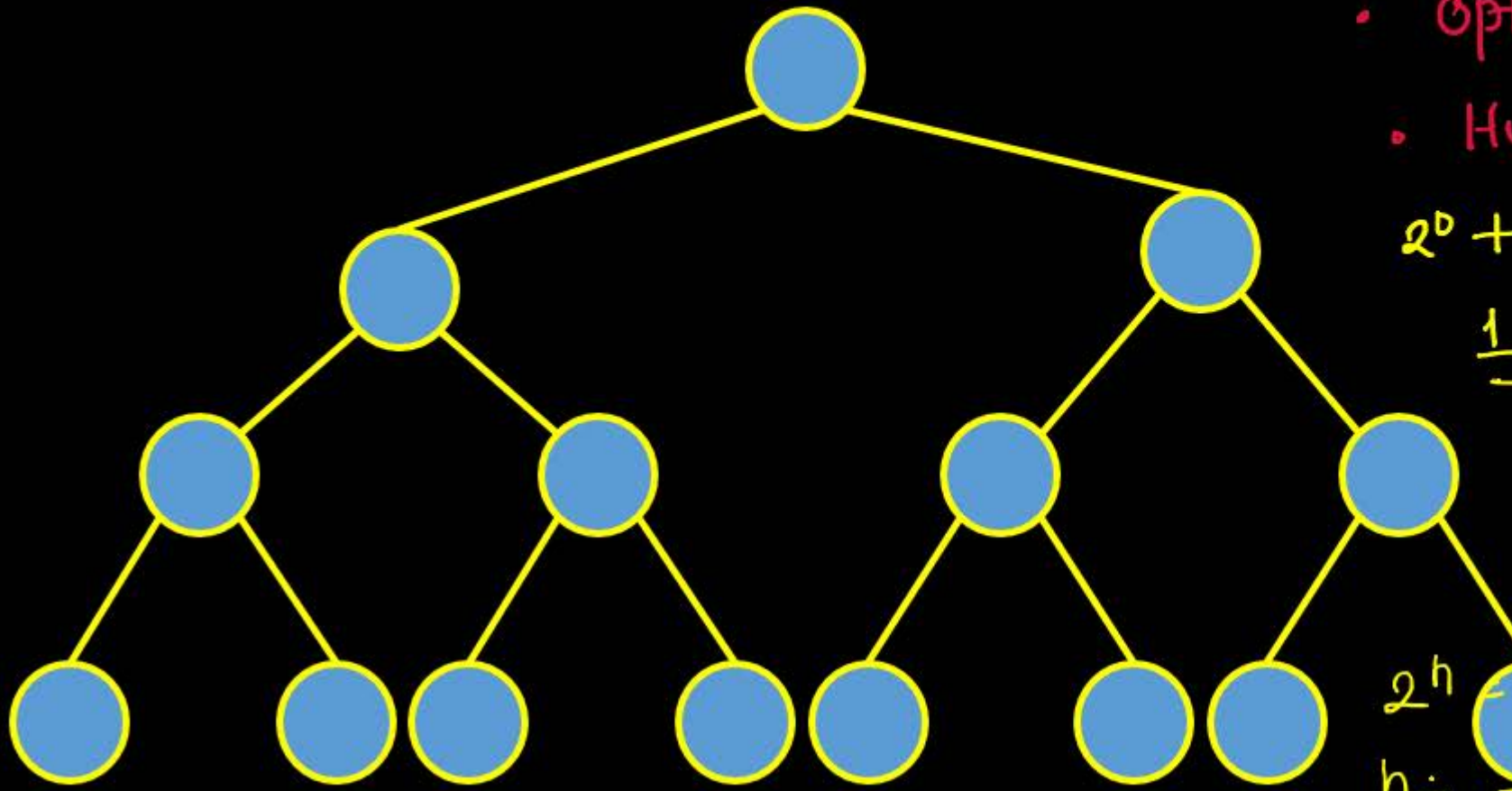
$\Rightarrow 2 \cdot 2^h = N+1$

$\Rightarrow 2^h = \frac{N+1}{2}$

$h = \log_2\left(\frac{N+1}{2}\right)$

Complete Binary Tree Properties

Minimum Number of Nodes



Spanning tree

- Optimal Merge pattern
- Huffman code

$2^0 + 2^1 + 2^2 + \cdots + 2^{h-1} + 1$

$\frac{1(2^h - 1)}{2-1} + 1$

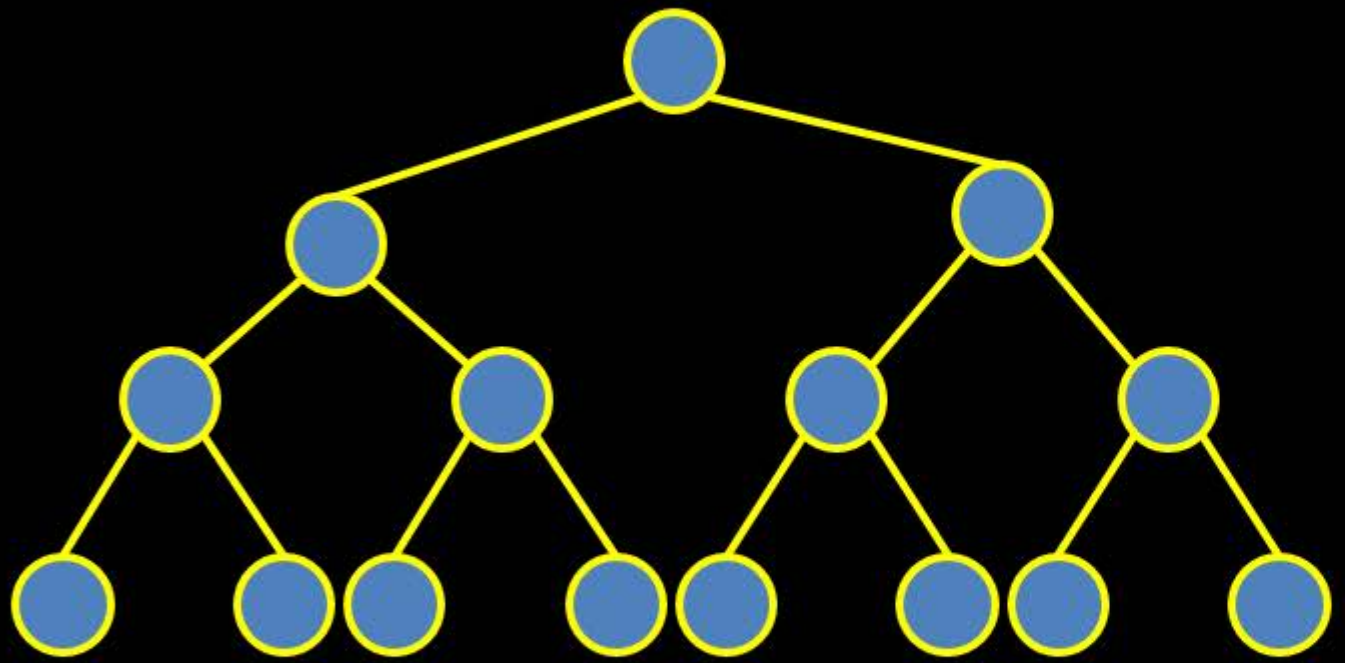
$= 2^h - 1 + 1 = 2^h$

$2^h = N$

$h = \log_2 N$

Complete Binary Tree Properties

Height of complete Binary tree restricted to $\Theta(\log n)$.

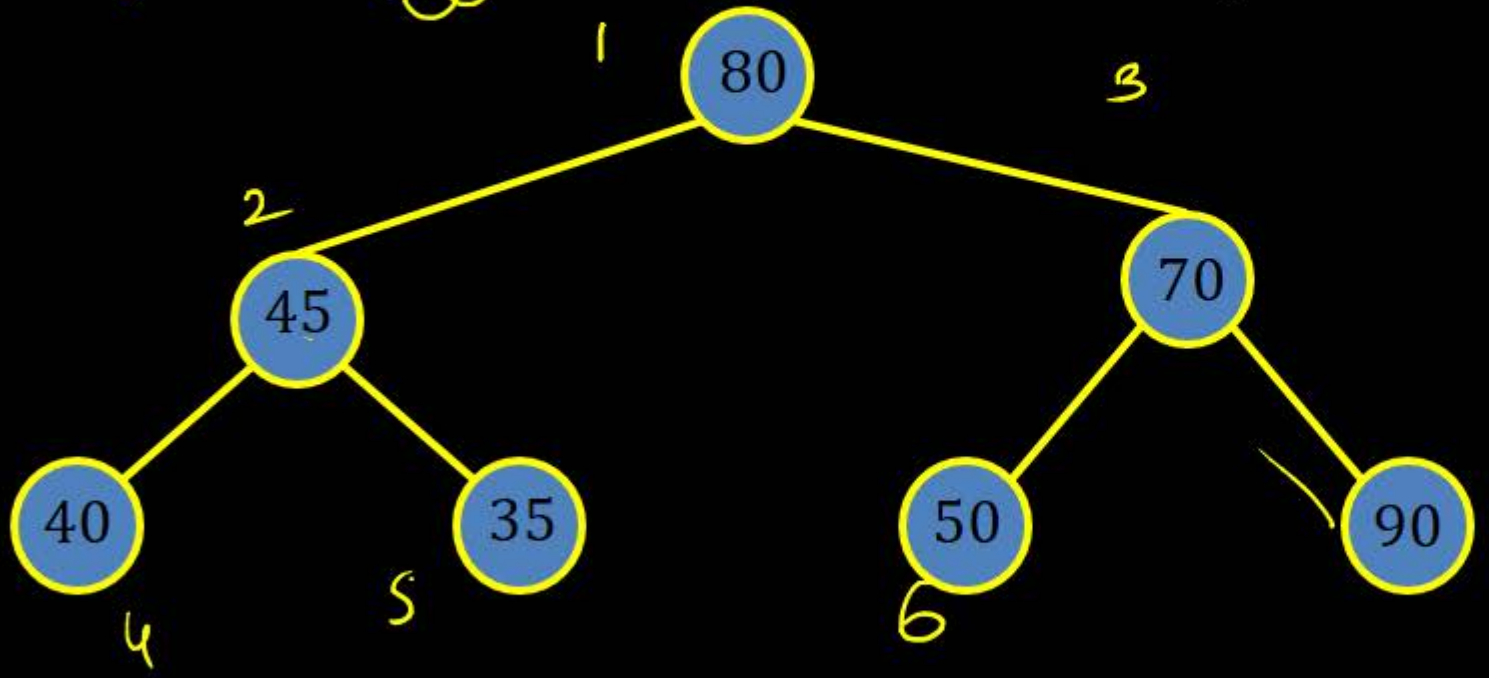


Heap with N Nodes

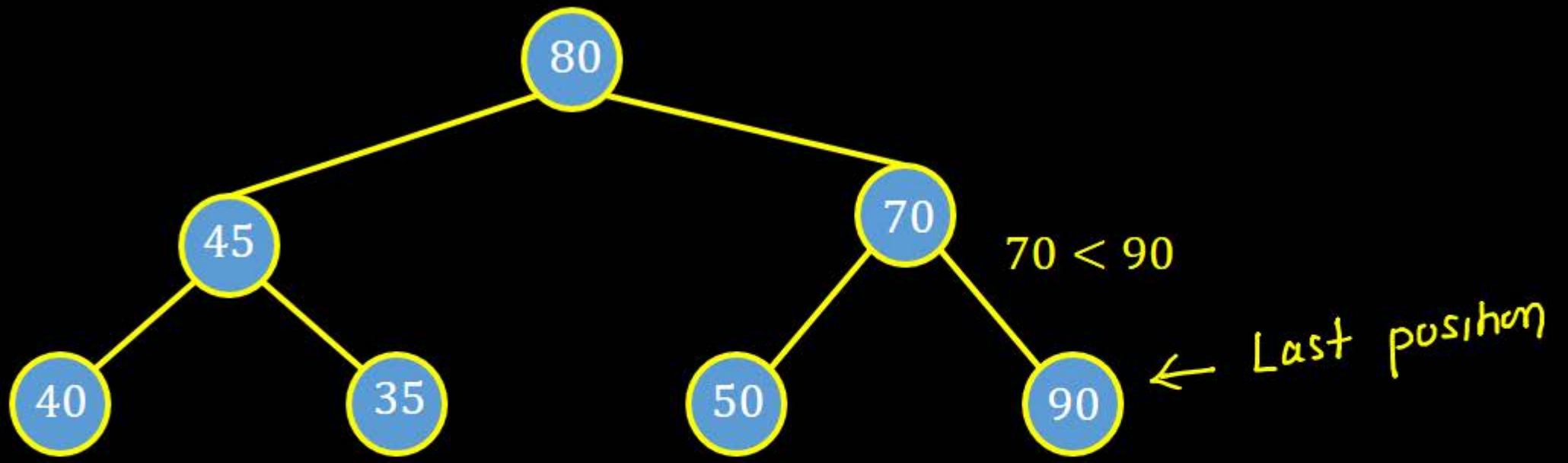
$O(\log_2 N)$

Insert in Binary Heaps

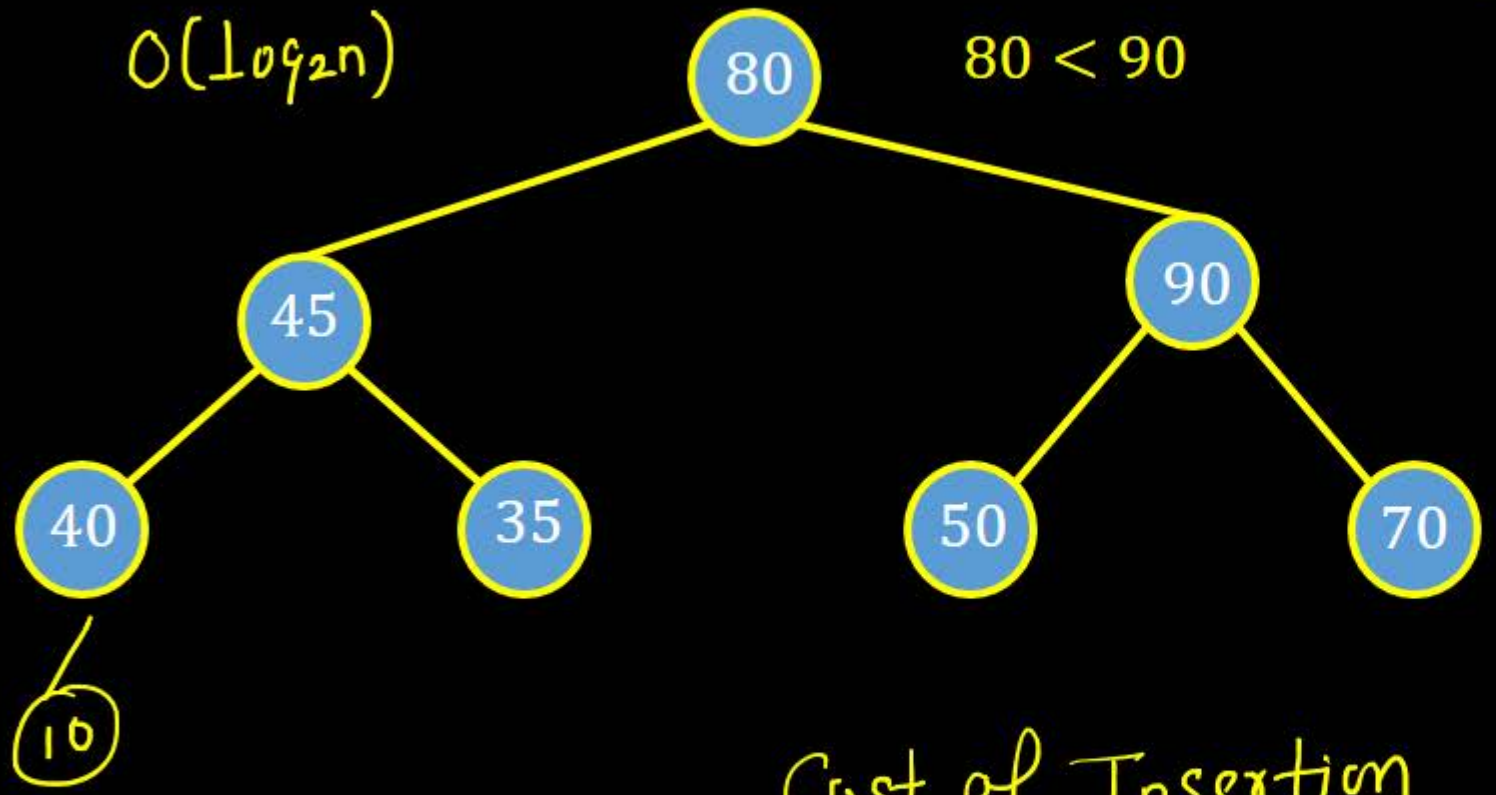
Insert the new element in last position of the array. Insert 90 in the given max heap.



Insert in Binary Heaps



Insert in Binary Heaps



Max heap

Insert in Last position

Compare with parent Node

if key value is less

Swap / Interchange

Repeat the process till the New inserted element settle in its correct position

Cost of Insertion

time complexity of Insertion

worst case it climbs up become the Root

Deletion from Binary Heap

- The maximum element from the max heap a[1:n]can be deleted by deleting the root of the corresponding complete binary tree.
- The last element of the array, that is, a[n], is copied to the root, and finally we call Adjust (a, 1, n - 1).

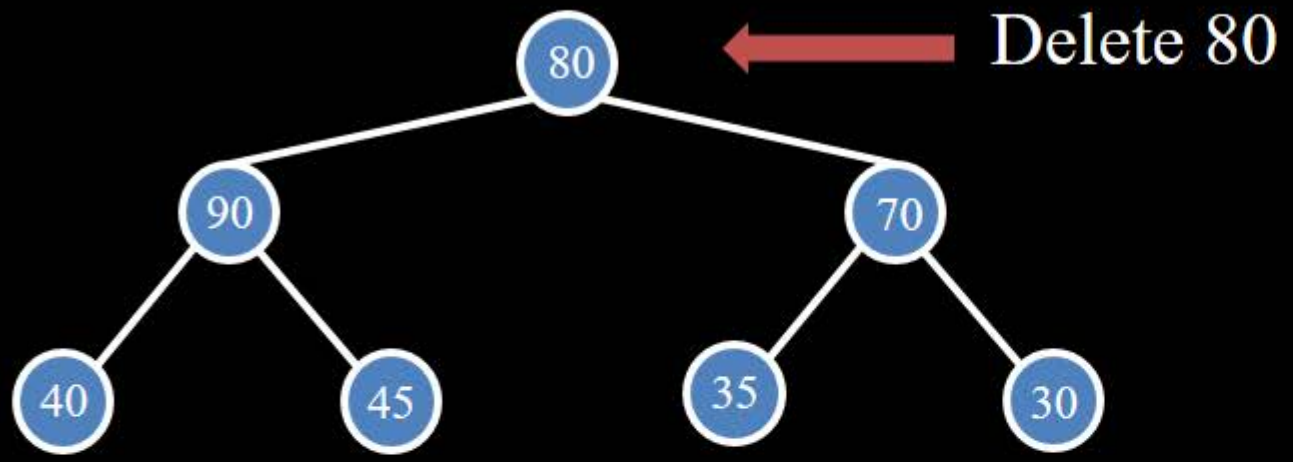
Adjust/Heapify

If the subtrees rooted at 2i(Left Child) and 2i + 1 (right child) are already max heaps, then Adjust (Heapify) will rearrange elements of a[] such that the tree rooted at i is also a max heap.

Deletion from Heap

- To delete the maximum key from the max heap, we use an algorithm called Adjust (Heapify).
- Adjust (Heapify) takes as input the array a[] and the integers i and n. It regards a[1:n] as a complete binary tree.

Example of Deletion



Heap - deletion

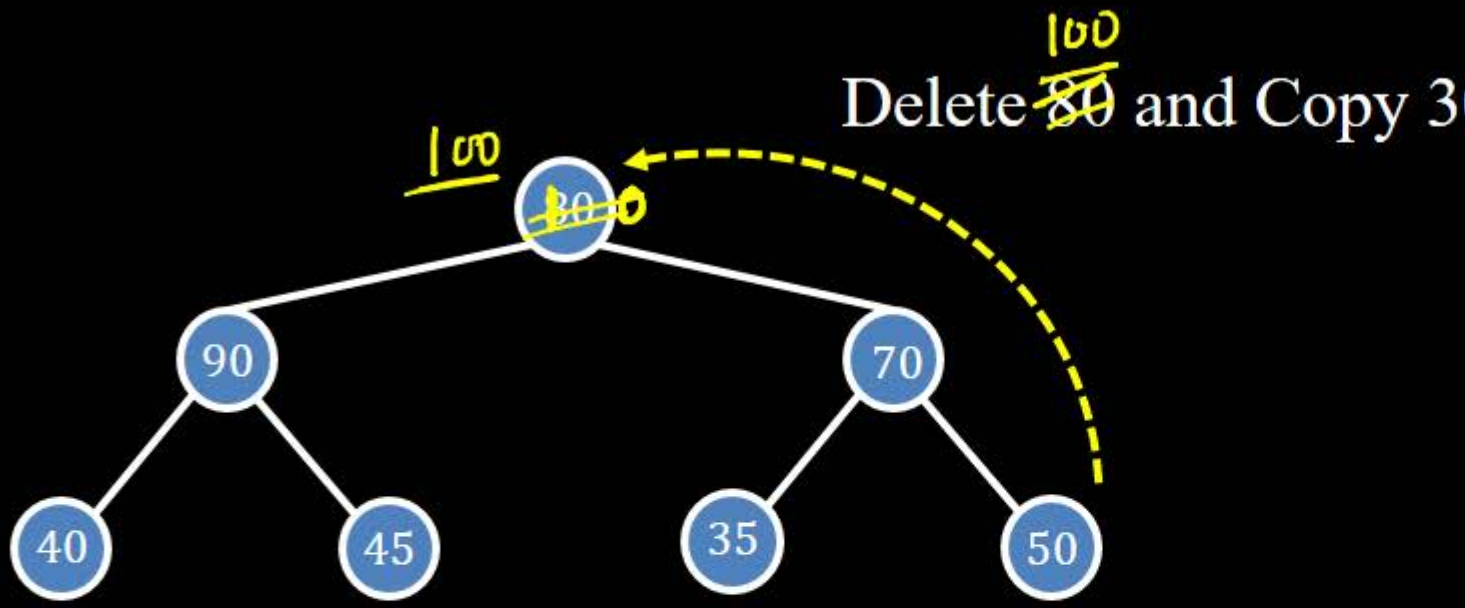
then max Heap delete maximum

min Heap is going to ~~delete~~ delete min

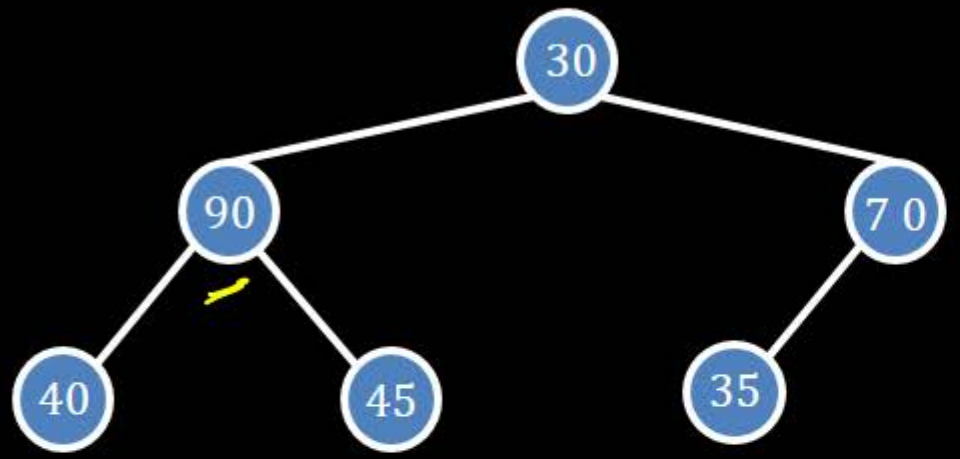
Root will be deleted

1. Delete the Root
2. Copy the Last element as Root (No Longer a max Heap)
3. Adjust the elements (Rearrange the element so that it become Heap again)

Example of Deletion



Example of Deletion



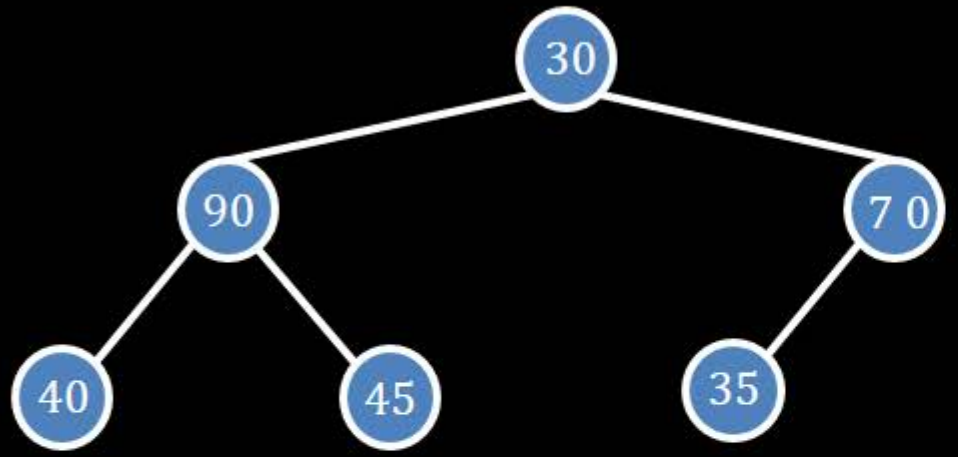
Adjust The Heap

Adjust

Find maximum

Left child or Right

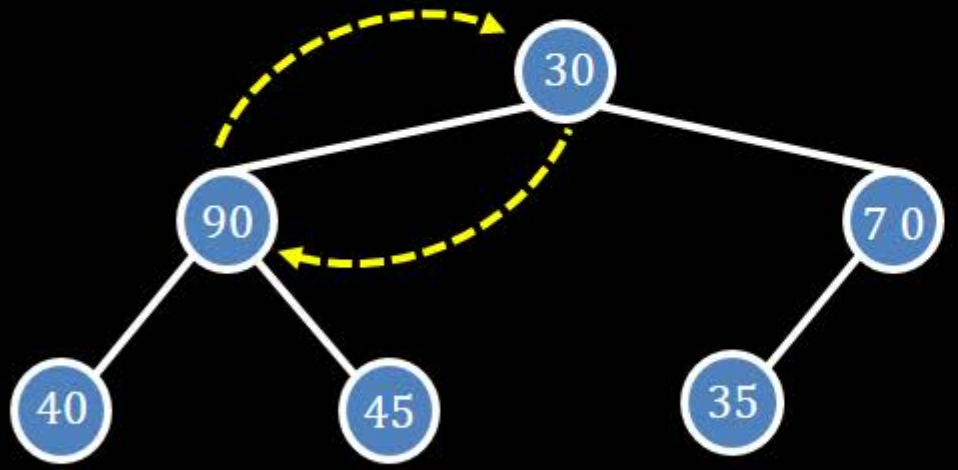
Example of Deletion



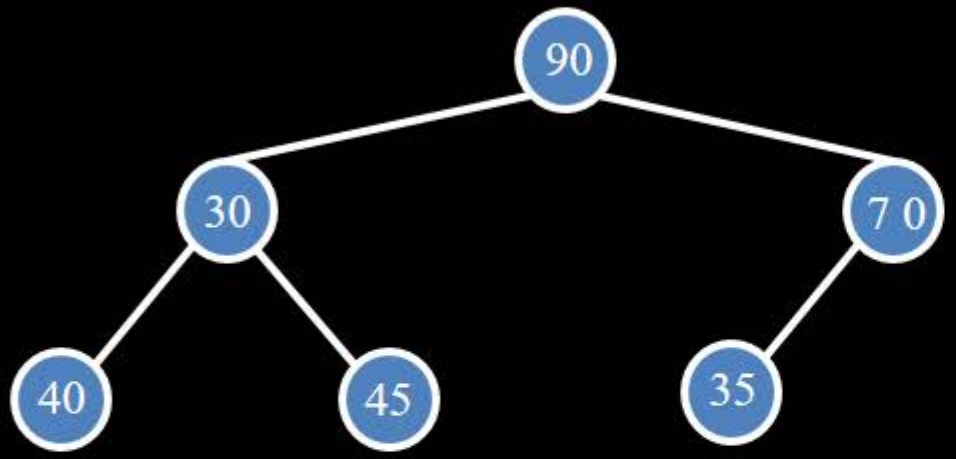
Example of Deletion

Compare the left and right child of node 30 to Adjust and find the greater one.

$$90 > 70$$



Example of Deletion

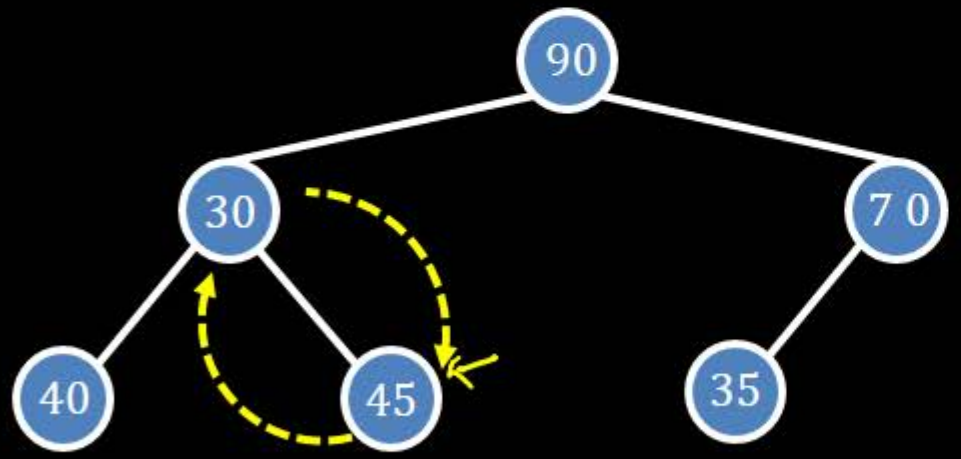


Example of Deletion

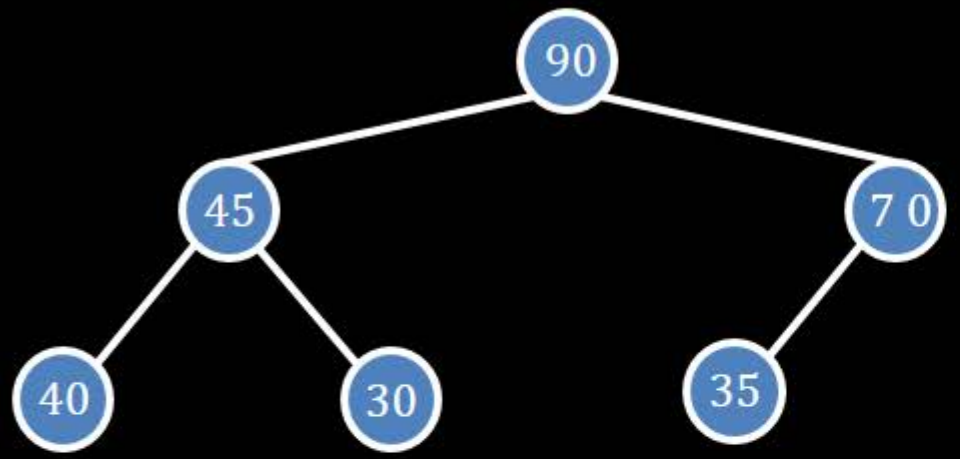
Compare the left and right child of node 30 to Adjust

and find the greater one.

$45 > 40$



Example of Deletion



Adjust or heapify completes

So started as a leaf and after rearranging again settle as leaf only.

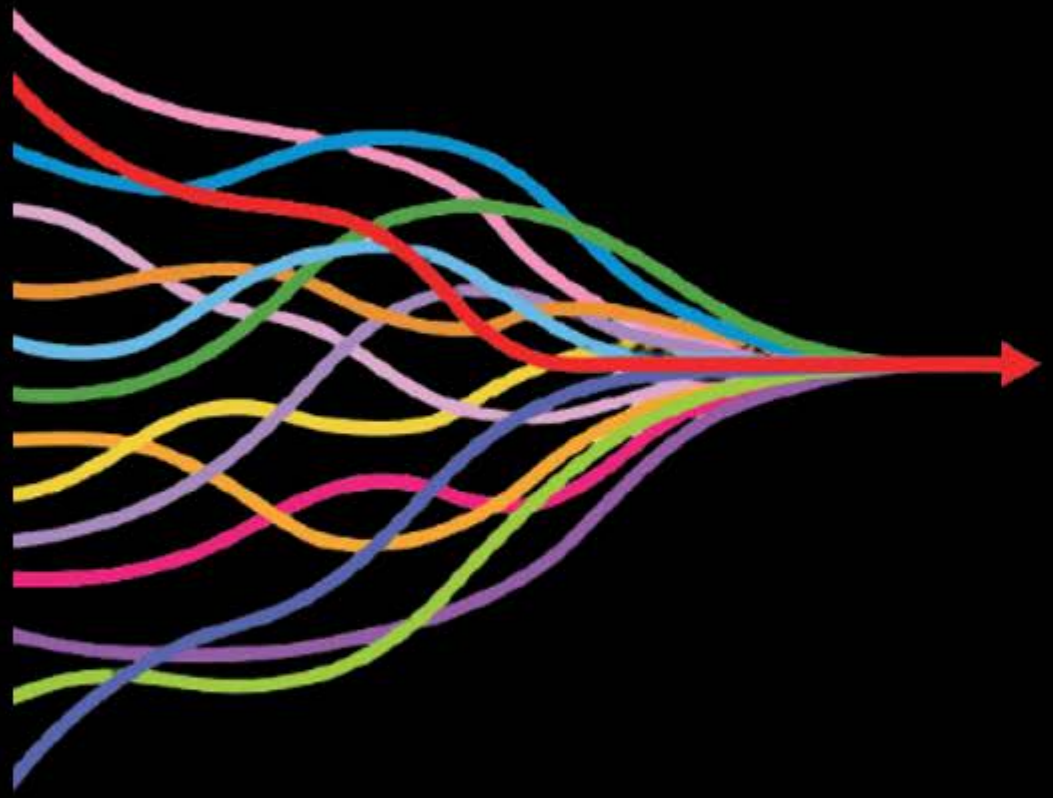
The Newly copied element as Root it may again settle as Leaf.

It has to climbdown all the way from root to Leaf.

$O(\log_2 n)$

Insertion & Deletion in a Heap required $O(\log_2 n)$ time

Min Heap – Insert – $O(\log_2 n)$
Delete – $O(\log_2 n)$



Optimal Merge Pattern

Problem Statement

- Merging of Two Sorted file :

Merging 2 sorted file of size m and n

the No. of comparison Required is O(n+m)

Maximum No. of comparison = n+m-1

Problem Statement

Merge n sorted files:

Given n sorted file we want to merge these sorted file into a single sorted file.

Problem Statement

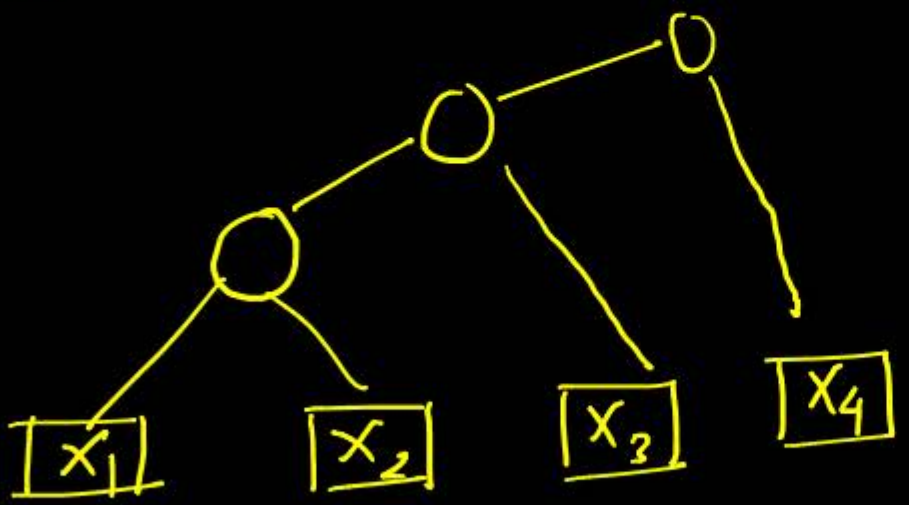
- We have seen that two sorted files containing n and *m* records respectively could be merged together to obtain one sorted file in time O(n + m).
- Pairwise merge n sorted files: When more than two sorted files are to be merged together, the merge can be accomplished by repeatedly merging sorted files in pairs.

Description

- **Example:** Thus, if files x_1, x_2, x_3, and x_4 are to be merged

Is it the only merging pattern possible

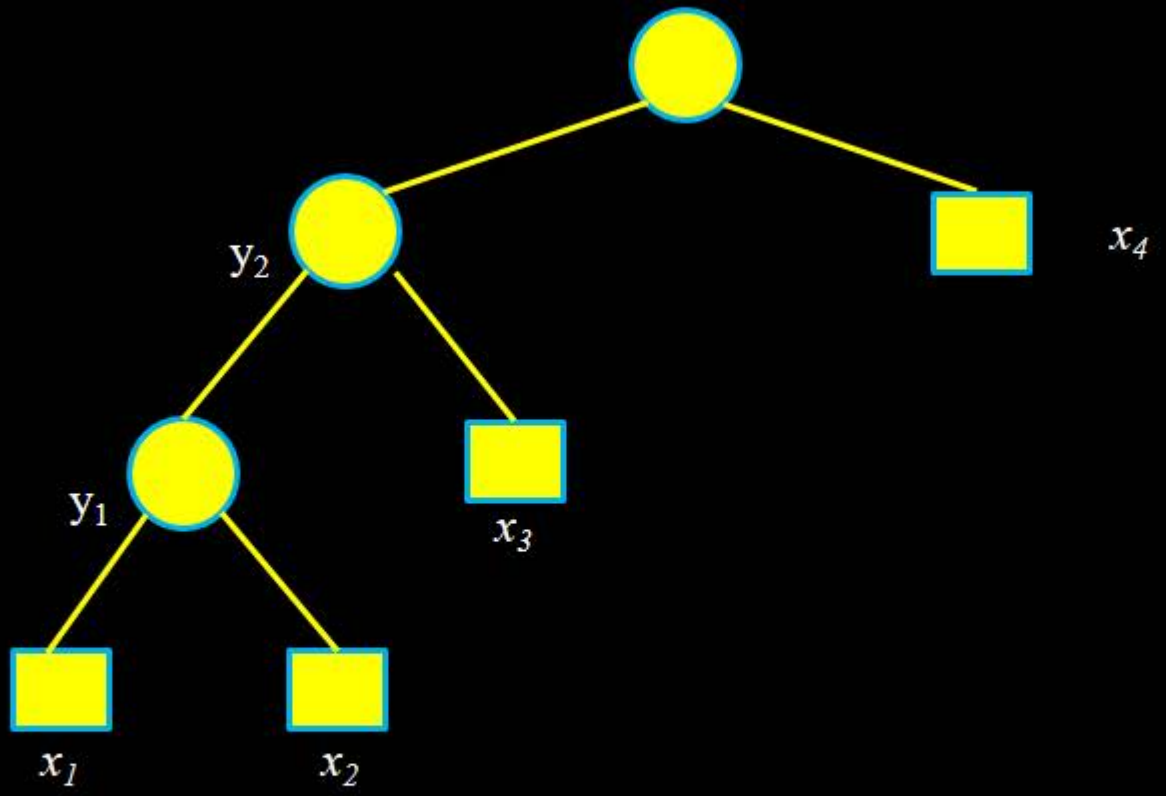
Pair wise merging : at a time only pair of file will be merged.



Description

- **Example:** Thus, if files x_1, x_2, x_3, and x_4 are to be merged, we could first merge x_1 and x_2 to get a file y_1.
- Then we could merge *y1* and *x3* to get y2 .
- Finally, we could merge y2 and x4 to get the desired sorted file.

Description

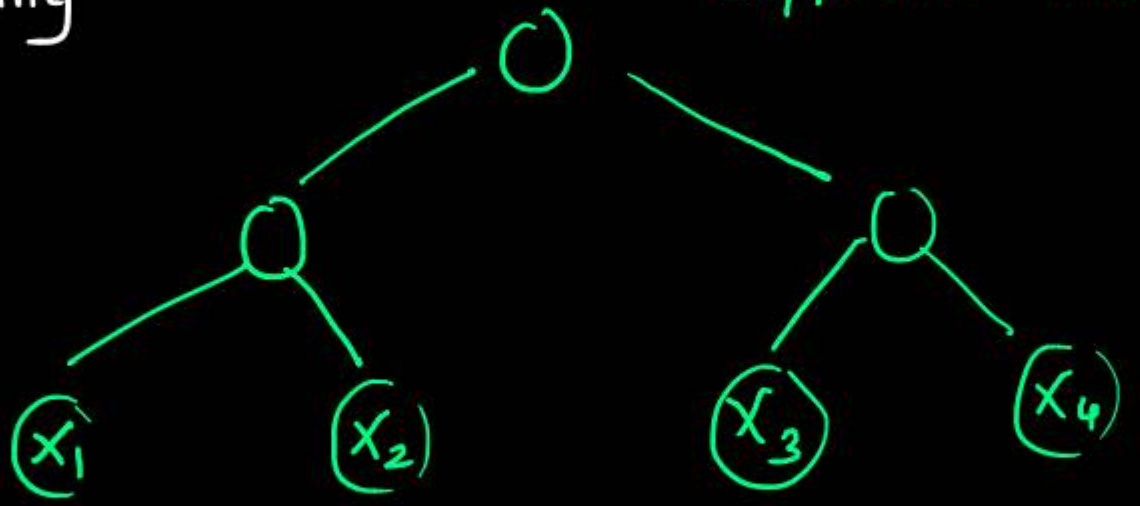


Description

- Alternative Merging:

Cost of merging

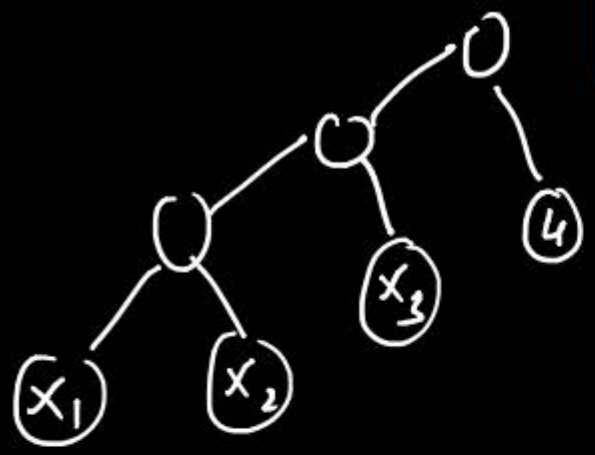
How one merging pattern is different from another

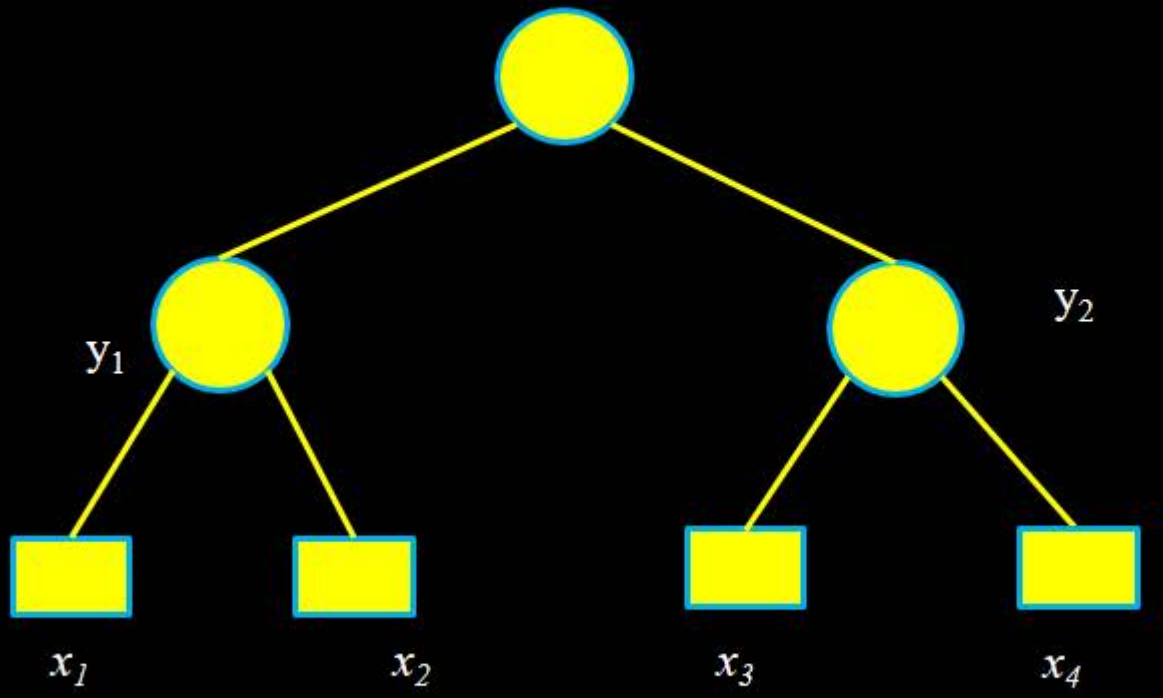


Description

- Alternatively, we could first merge x_1 and x_2 getting y_1, then merge x_3 and x_4 and get y_2 and finally merge y_1 and y_2 and get the desired sorted file.

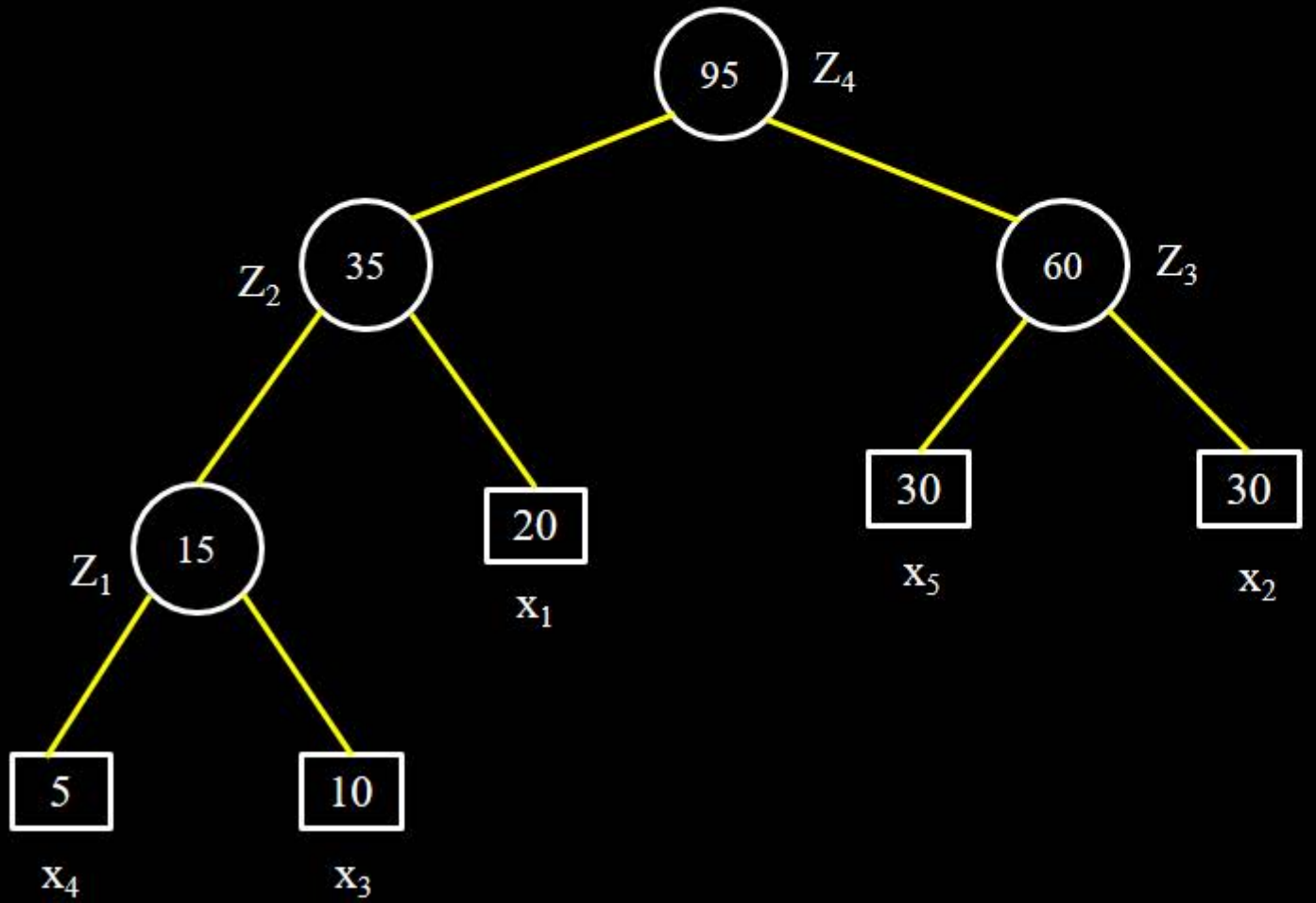
Description





Discreption

- Alternatively, we could first merge *x1* and *x2* getting y1, then merge x3 and X4 and get *y2* and finally merge *y1* and y2 and get the desired sorted file.



Problem Statement

Given n sorted file optimal merge pattern problem is to find an optimal way of ~~mer~~ pair wise merging so that No. of Record movements can be minized.

Because we are merging two files at a time then merging patter looks like a binary tree

Problem Statement

- Given n sorted files, there are many ways in which to pair wise merge them into a single sorted file.

Optima Merge Pattern

- Different pairings require differing amounts of computing time. The problem we address ourselves to now is that of *determining an optimal way (one requiring the fewest comparisons) to pairwise merge n sorted files.*
- Since this problem calls for an ordering among the pairs to be merged, it fits the ordering paradigm.

Two-way Merging

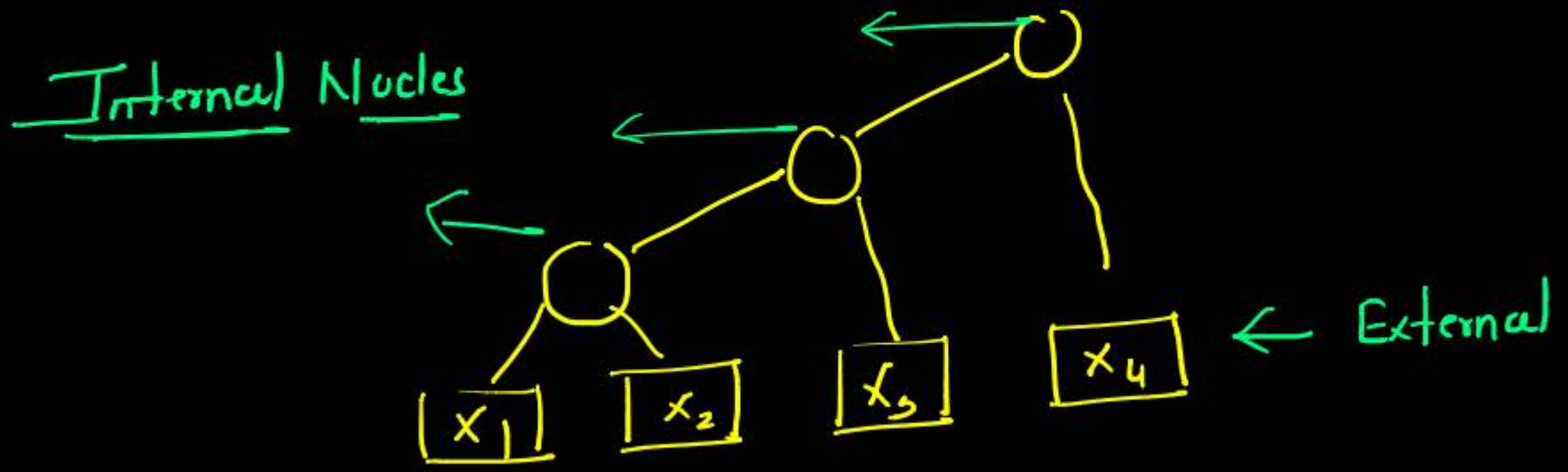
- Two way merging: two file merging together at a time.

Two-way Merging

- Two way merging: The merge pattern such as the one just described will be referred to as a *two-way merge pattern* (each merge step involves the merging of two files). The two-way merge patterns can be represented by binary merge trees.

Two-way Merging

- *External nodes:* the file needed to be merged will be considered as External



Two-way Merging

- *internal nodes.*

Two-way Merging

- The remaining nodes are drawn as circles and are called *internal nodes*.

Weighted External Path Length

Weighted External Path Length:

Represent the file size

is defind as Sum of product of weight (file size) and the Legth (Distance from the Root)

q_i - is Length of

$$\sum_{i=1}^{n} d_i \cdot q_i$$

Weighted External Path Length

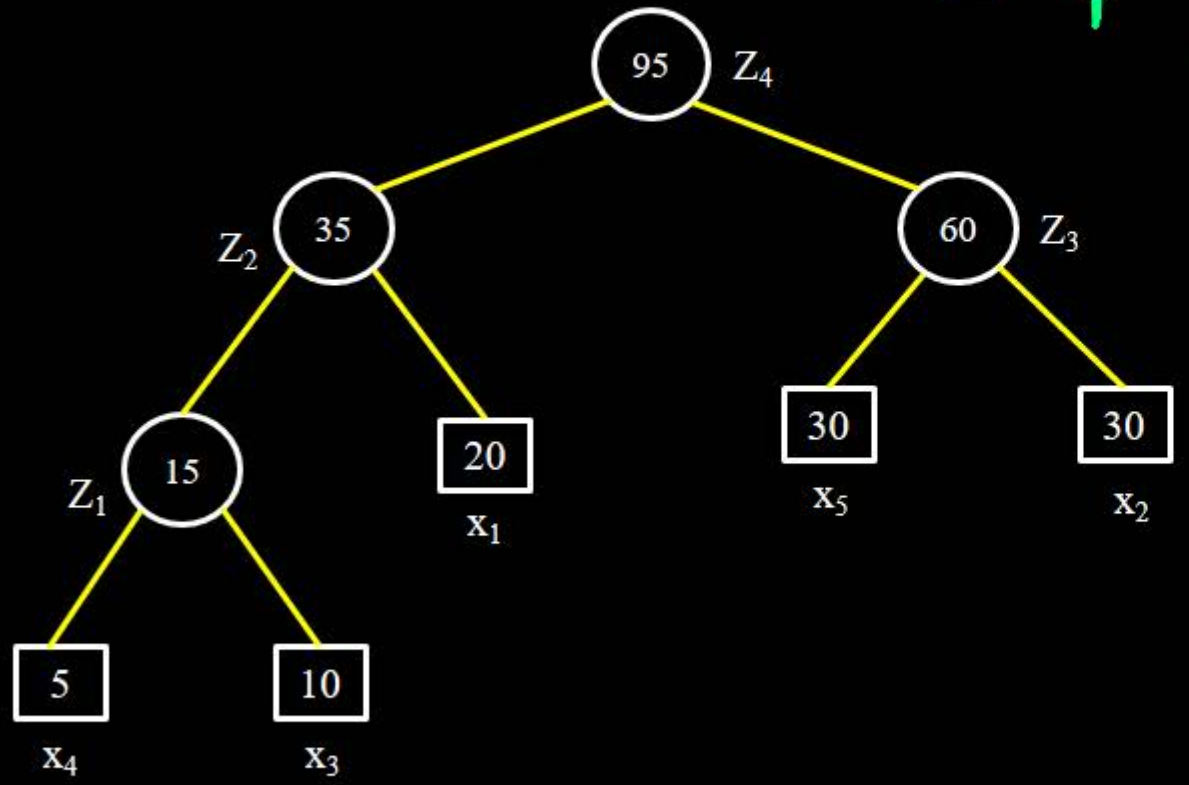
- Given an extended binary tree (that is, simply any complete binary tree, where leafs are denoted as external nodes), associate weights with each external node. ***The weighted path length*** of is the *sum of the product of the weight and path length of each external node*, over all external nodes.

Weighted External Path Length

- If d_i is the distance from the root to the external node for file x_i and q_i the length of x_i is then the total number of record moves for this binary merge tree is
- $\sum_{i=1}^{n} d_i q_i$
- This sum is called the *weighted external path length* of the tree.

Weighted External Path Length

- Example:



$5 \times 3 + 10 \times 3 + 20 \times 2 + 30 \times 2 + 30 \times 2$

$\sum_{i=1}^{n} d_i q_i$

d_i - distance

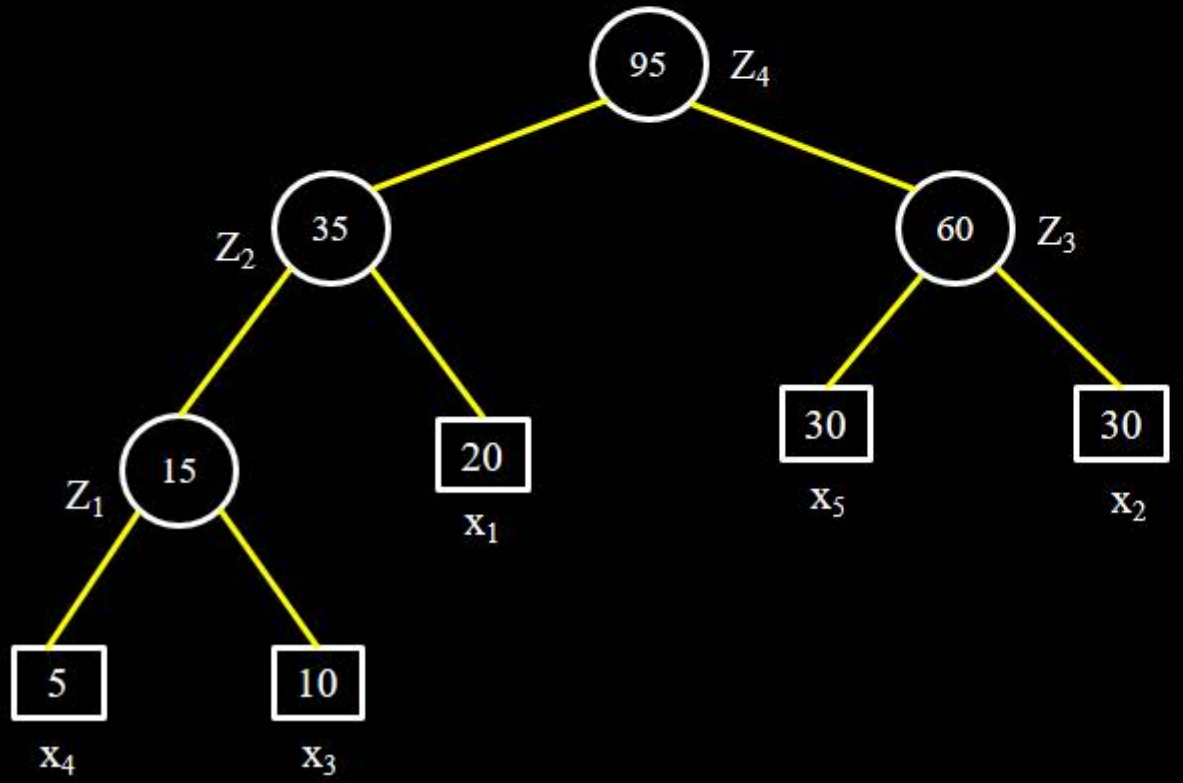
q_i - file size

$15 + 30 + 40 + 120 = 205$

Minimize this cost

Weighted External Path Length

- The external node x4 is at a distance of 3 from the root node z4. Hence, the records of file x4 are moved three times, once to get z1, once again to get z2, and finally one more time to get z4.



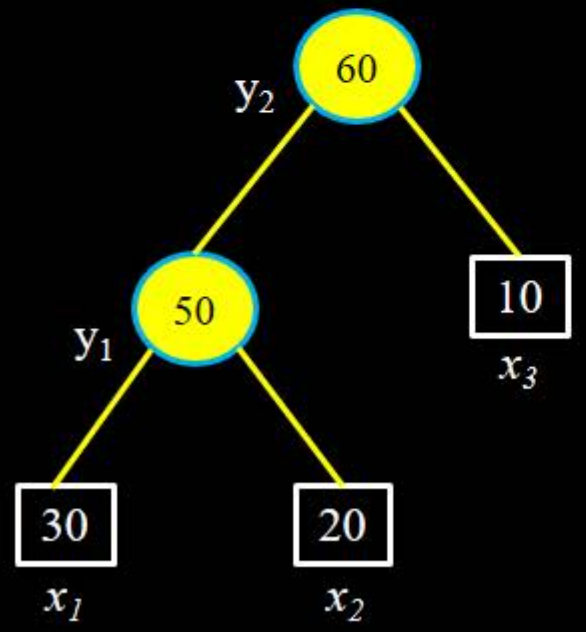
Optimal Merging

- The files x_1, x_2, & x_3 are three sorted files of length 30, 20, and 10 records each.

weighted External path length

$30 \times 2 + 20 \times 2 + 10 \times 1$

$60 + 40 + 10 = 110$

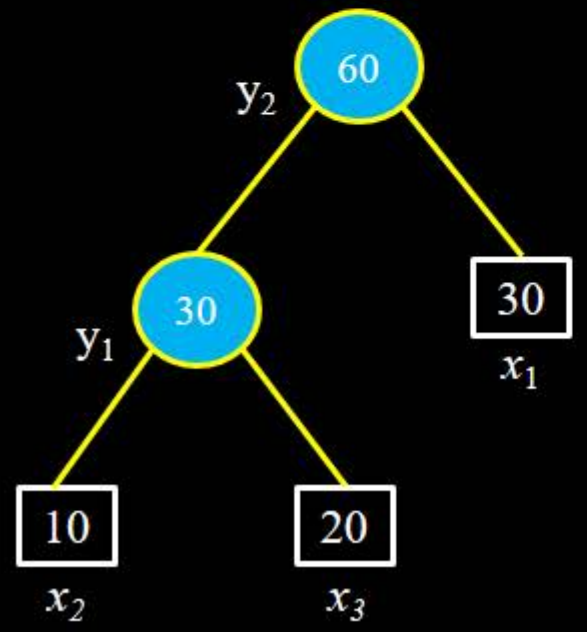


Example

- The files x_1, x_2, & x_3 are three sorted files of length 30, 20, and 10 records each.

optimal way merging

The total is 90.



weighted external path length

$10 \times 2 + 20 \times 2 + 30 \times 1$

$20 + 40 + 30 = 90$

minimum weighted external path length

- Every iteration if two files of minimum size is ~~one~~ merged together then this merge pattern will have minimum weighted External path length

Minimum Weighted External Path Length

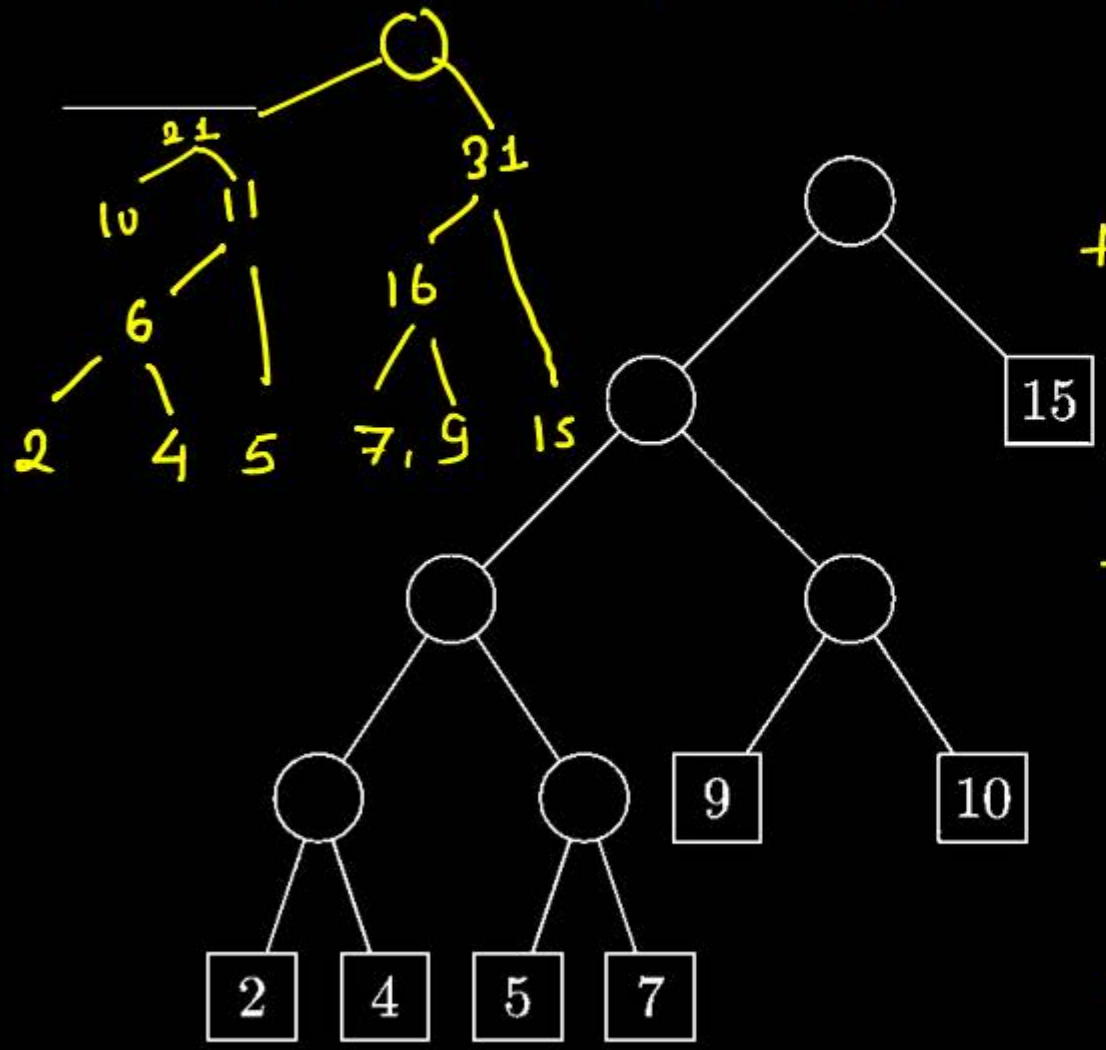
Minimum Weighted External Path Length

If in every step two files of minimum size is merged then it becomes

minimum weighted external path length

GATE 1991| 1 Mark Question

- The weighted external path length of the binary tree in figure is



4 1 x 2 +
4 x 4
+

18 x 4 = 72
19 x 3 57
15
144

2, 4, 5, 7, 9, 10, 15

external path
6, 5, 7, 9, 10, 15
11, 7, 9, 10, 15
10, 11, 15, 16
21 15, 16

minimum weighted

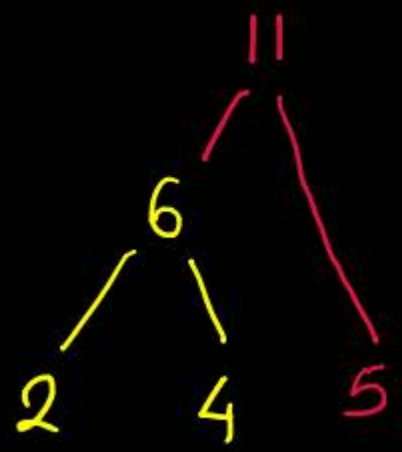
(2+4) x 4 = 24
(9+7+5) x 3 - 63
(10+15) 2 50
137

Greedy Approach

2, 4, 5, 7, 9, 10, 15

6, 5, 7, 9, 10, 15

minimum weighted external path length



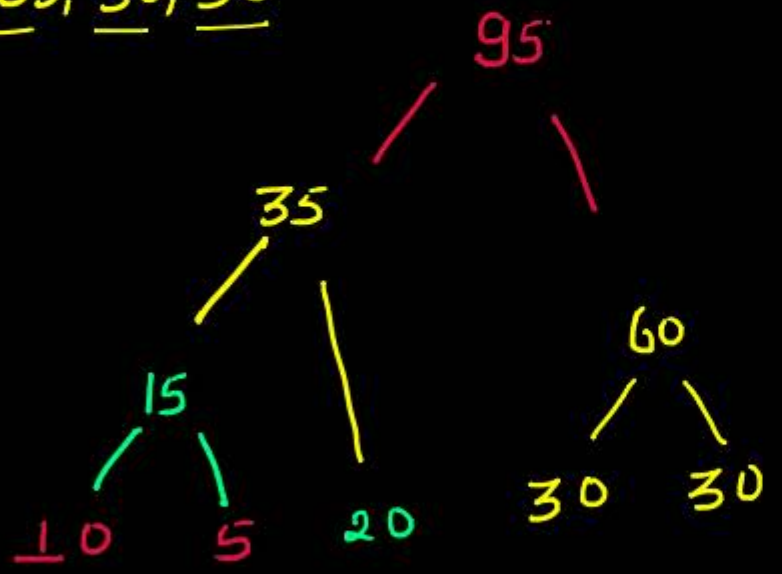
Greedy Approach

- Since merging an n-record file and an m-record file requires possibly n + m record moves, the obvious choice for a selection criterion is: at each step merge the two smallest size files together.

Example

- Thus, if we have five files (x1,……., x5) with sizes (20,30,10,5,30) find optimal merging pattern.

85, 30, 30



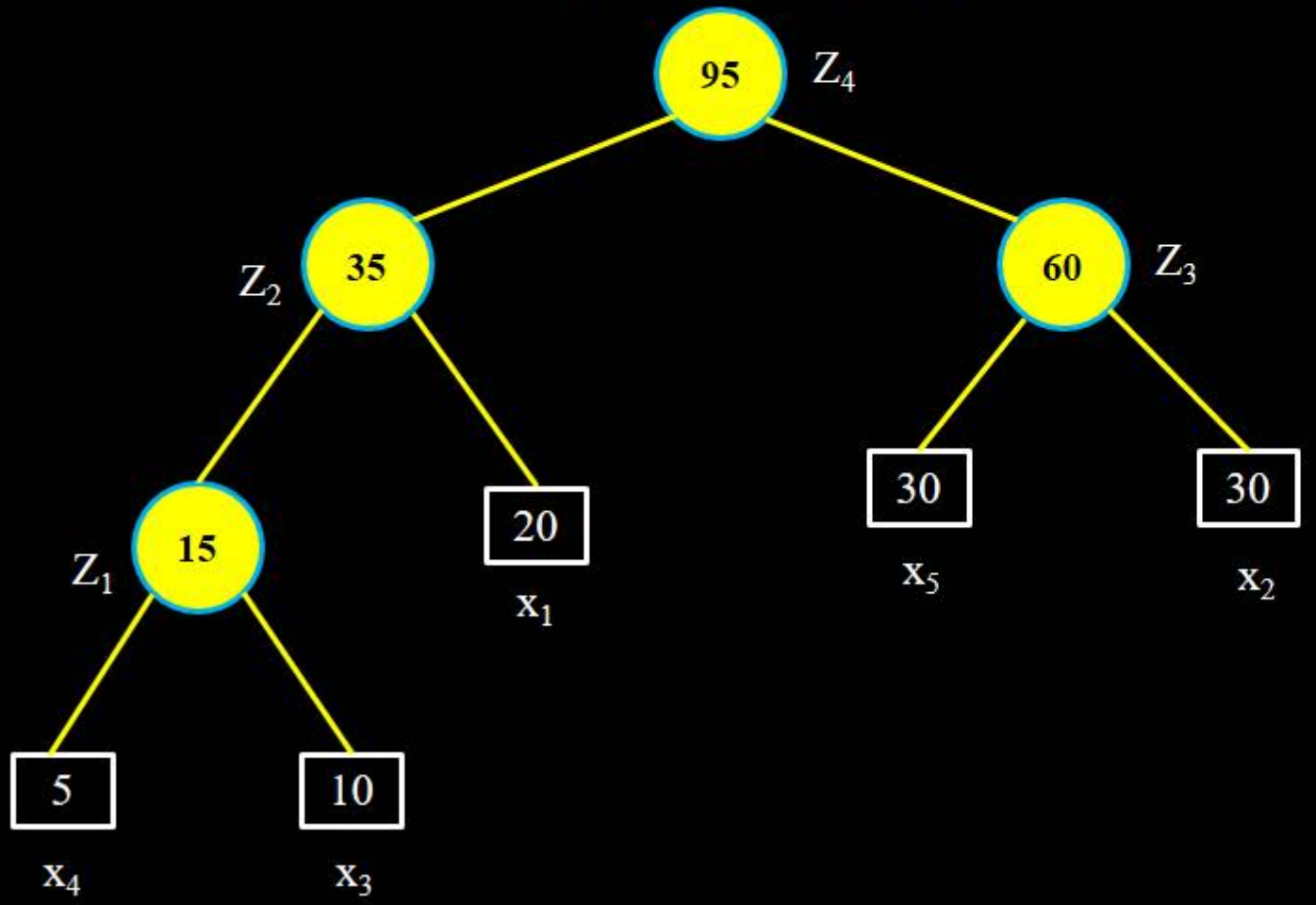
minimum weighted external path length

Simple category

15 x 3 = 45
80 x 2 = 160
205

Example

- Thus, if we have five files $(x_1,\ldots\ldots,x_5)$ with sizes (20,30,10,5,30) find optimal merging pattern.



Solution

- his is straightforward. The nodes of the given tree are given in square boxes. The weights associated with the nodes are the numbers example 15,9,10 etc.
- Weighted path length = sigma(for(each node in the tree) (path length)*(weight of the node)).
- $\sum_{i=1}^{n} d_i q_i$
- Path Lengthi∗Weight Of Nodei
- So answer (written in path_length * weight form) =4∗2+4∗4+4∗5+4∗7+3∗9+3∗10+1∗15=144

GATE 2014 SET-II| 2 Mark Question

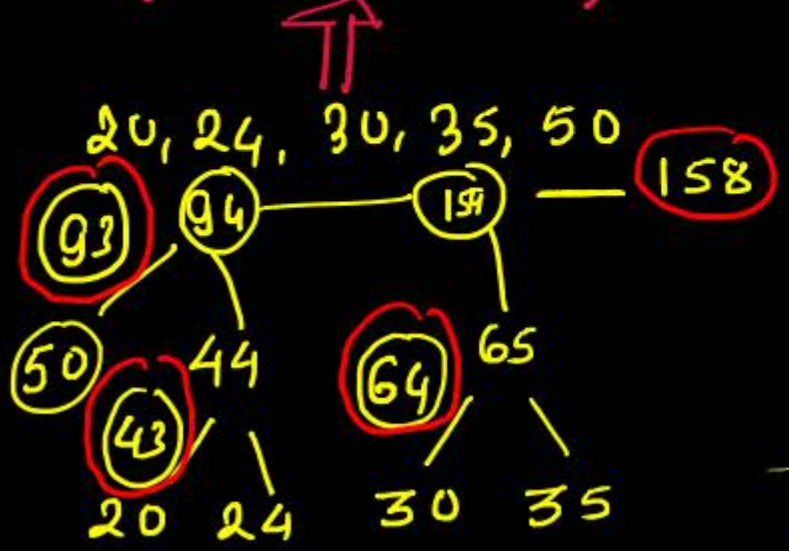
Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is 358 .

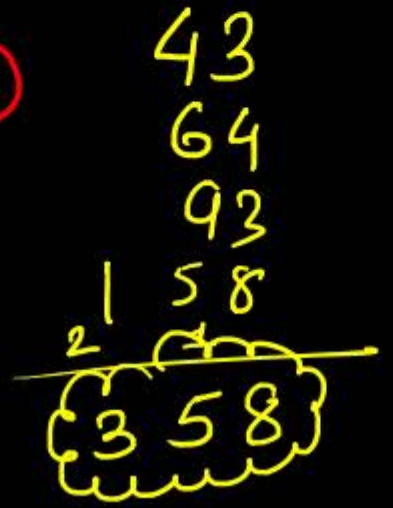
Break
Huffman code

2014 - IIT
2022 - IIT KGP

if we merge 2 files of minimum
the maximum No. comparison
= (m+n-1) worst

Minimum No. of comparison
min(m,n)





Solution

The optimal algorithm always chooses the smallest sequences for merging.

20, 24−44, 43 comparisons

30, 35−65, 64 comparisons

44, 50−94, 93 comparisons

65, 94−159, 158 comparisons

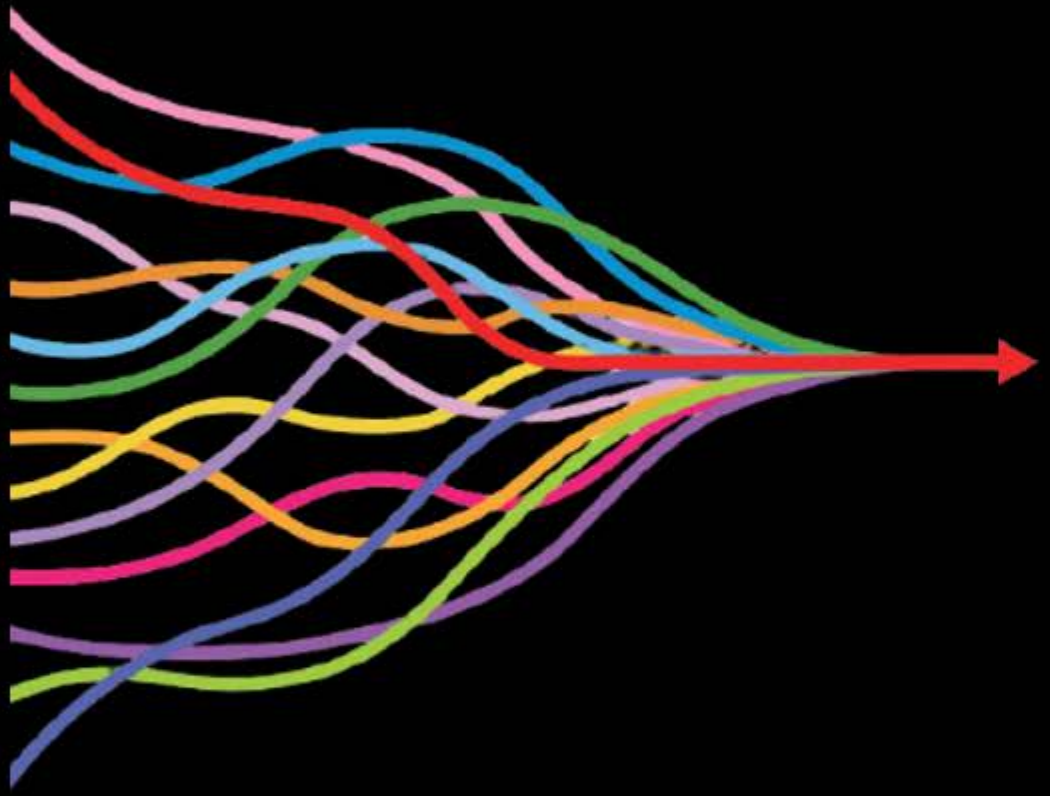
so, totally 43+64+93+158=358 comparisons.

In merge operation we do a comparison of two elements and put one element in the sorted output array. So, every comparison produces one output element. But for the last element we won't need a comparison and we simply insert it to the output array. So for n output elements we need (n−1) comparisons.

Algorithm

Construction of 2way-merge tree.

```
line procedure TREE(L, n)
1 for I = 1 to n - 1 do
2     call GETNODE(T)
3     LCHILD(T) = LEAST(L)
4     RCHILD(T) = LEAST(L)
5     WEIGHT(T)= WEIGHT(LCHILD(T)) + WEIGHT(RCHILD(T))
6     call INSERTS( T, L)
7 end for
8 return (LEAST(L))
9 end TREE
```



Huffman codes

Code

Code : Encryption & decryption

Code . Secrete code — Communication, Information Theory

Set of Rules to convert. data, messages, images

from one form to another

. Communication

. storage (compression)

Reducing the data size

Code

In communications and information processing, code is a system of rules to convert information—such as a letter, word, sound, image, or gesture—into another form, sometimes shortened or secret, for communication through a communication channel or storage in a storage medium.

Uniform & Non Uniform Code

ASCII code to Represent the characters in Computer

Size of ASCII is code

8 bits

Uniform code : The size of each code is same

uniform code

Non Uniform code : The Size (Length) of each code differ.

Prefix Code

If Codes are uniform

e.g. 0000 — a
0001 — b
0010 — c
0011 — d

It will Never happen ~~two~~ one codes Represents 2 values.

001 — code
0011 Not

a ~~pre~~ ~~code~~ satisfying prefix code property

Non uniform code code ∴

No "whole code" will be prefix of another code.

prefix of a code

0110 — 0 / 01 / 011 / 0110 } proper prefix

prefix

Prefix Code

- A prefix code is a type of code system distinguished by its possession of the "prefix property", which requires that there is no whole code word in the system that is a prefix (initial segment) of any other code word in the system.
- It is trivially true for fixed-length code, so only a point of consideration in variable-length code.

Huffman Prefix Code

Huffman code is used for optimal prefix code for data communication and having property of "lossless data compression."

- Some messages are frequent and other messages may ~~set~~ Send less frequently.

- most frequently used alphabet in English e

Huffman Prefix Code

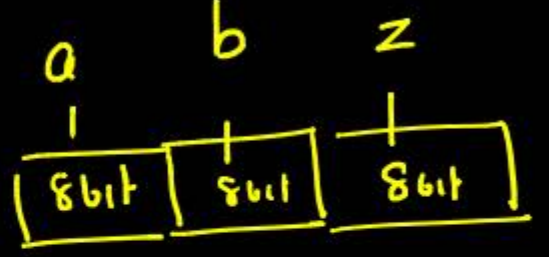
- In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.

Huffman Prefix Code

- The process of finding or using such a code proceeds by means of Huffman coding,
- The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file).

Huffman Prefix Code

- Huffman Code is Non-Uniform code
- data compression is frequently used messages will be encoded in less No. of bits.



Huffman Prefix Code

- Another application of binary trees with minimal weighted external path length is to obtain an optimal set of codes for messages $M_1,...,M_{n+1}$. Each code is a binary string that is used for transmission of the corresponding message

Advantages of Huffman Encoding

- This encoding scheme results in saving lot of storage space, since the binary codes generated are variable in length
- It generates shorter binary codes for encoding symbols/characters that appear more frequently in the input string
- The binary codes generated are prefix-free

Example

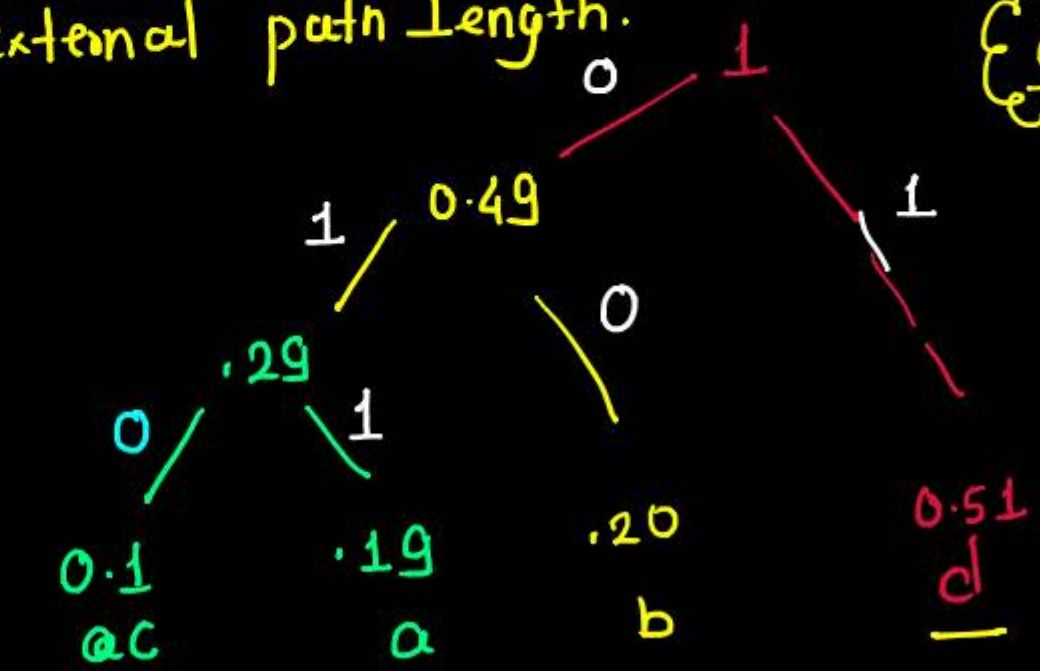
- Example: A message is made up entirely of characters from the set $X = \{a, b, c, d\}$. The table of probabilities for each of the characters is shown below:

Character	Probability
a	0.19
b	0.20
c	0.1
d	0.51

Lower Frequency assigned with Lable-0
Higher frequency assigned with Label-1

Binary tree with minimum weighted external path length.

decode tree



011
00
010
1

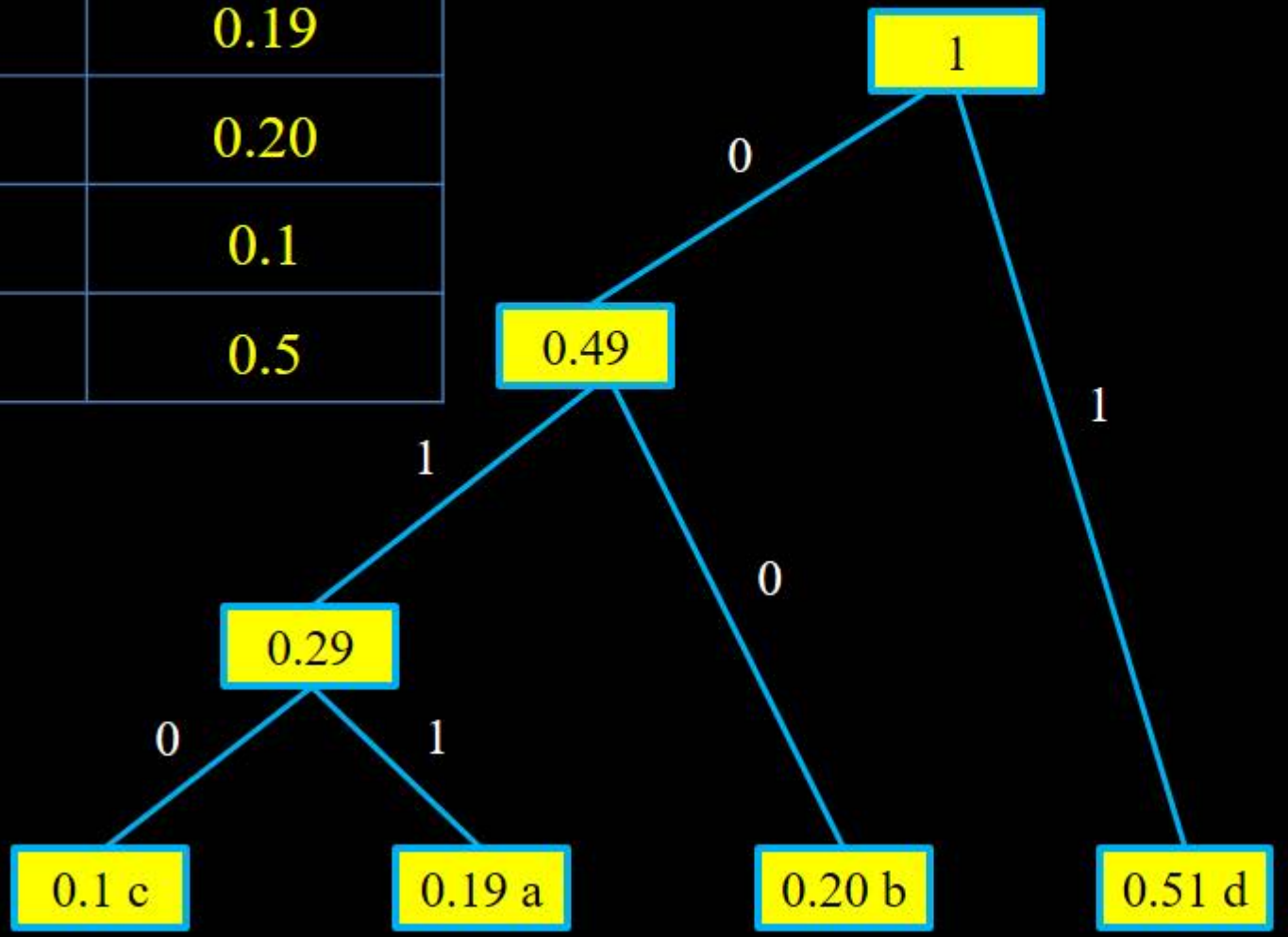
Example

- Example: A message is made up entirely of characters from the set $X = \{a, b, c, d\}$. The table of probabilities for each of the characters is shown below:

Character	Probability
a	0.19
b	0.20
c	0.1
d	0.51

Example

Character	Probability
a	0.19
b	0.20
c	0.1
d	0.5



GATE 2006 (IT)

Q. The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows

a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21

A Huffman code is used to present the characters. What is the sequence of characters corresponding to the following code?

110111100111010

(a) fdheg

(b) ecgdf

(c) dchfg

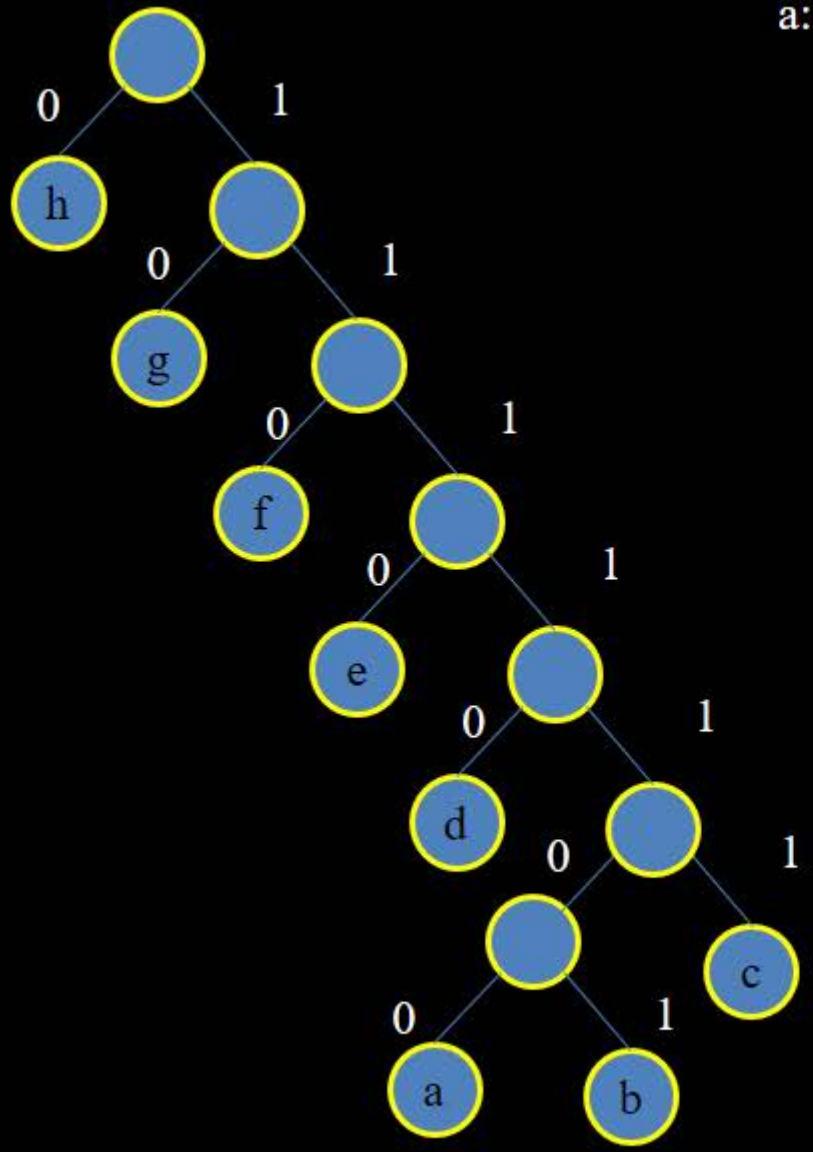
(d) fehdg

GATE 2006 (IT)

a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21

A Huffman code is used to present the characters. What is the sequence of characters corresponding to the following code?

110111100111010



a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21

110111100111010

GATE 2007 | 2 Marks Question

Suppose the letters *a, b, c, d, e, f* have probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}$, respectively.

Which of the following is the Huffman code for the letter *a, b, c, d, e, f*?

(A) 0, 10, 110, 1110, 11110, 11111

(B) 11, 10, 011, 010, 001, 000

(C) 11, 10, 01, 001, 0001, 0000

(D) 110, 100, 010, 000, 001, 111

GATE 2007 | 2 Marks Question

Suppose the letters *a, b, c, d, e, f* have probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}$, respectively.

Which of the following is the Huffman code for the letter *a, b, c, d, e, f*?

(A) 0, 10, 110, 1110, 11110, 11111

(B) 11, 10, 011, 010, 001, 000

(C) 11, 10, 01, 001, 0001, 0000

(D) 110, 100, 010, 000, 001, 111

What is the average length of the correct answer to Q.12?

(A) 3 (B) 2.1875 (C) 2.25 (D) 1.9375

GATE 2017

A message is made up entirely of characters from the set X = {P, Q, R, S, T}. The table of probabilities for each of the characters is shown below:

find optimal Set of prefix code

what is the average length of a code (Expected)

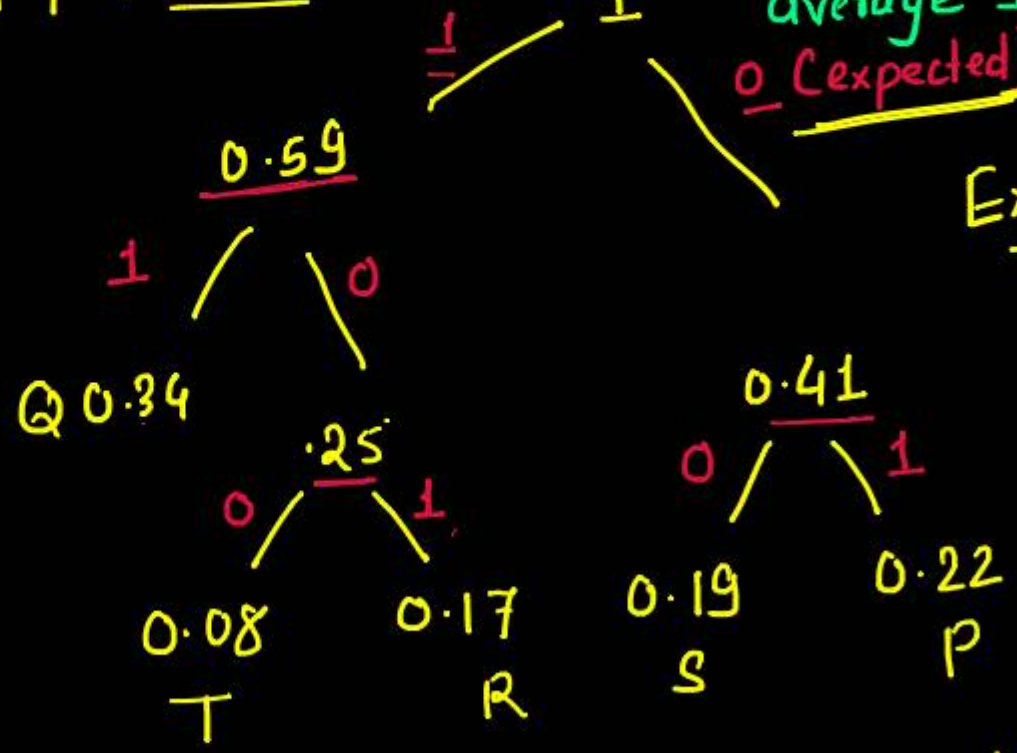
Expected value

12/5 Avg

2.49

Character	Probability
01 P	0.22
11 Q	0.34
101 R	0.17
00 S	0.19
100 T	0.08
Total	1.00

2 2 3 2 3



$= 0.22 \times 2 + 0.34 \times 2 + 0.17 \times 3 + .19 \times 2 + .08 \times 3$

$= 2.25$

GATE 2017

	Character	Probability	
2	8bit P	0.22	x 100 = 22
2	8bit Q	0.34	= 34
3	8bit R	0.17	= 17
2	8bit S	0.19	= 19
3	8bit T	0.08	8
	Total	1.00	

~~Avg len~~

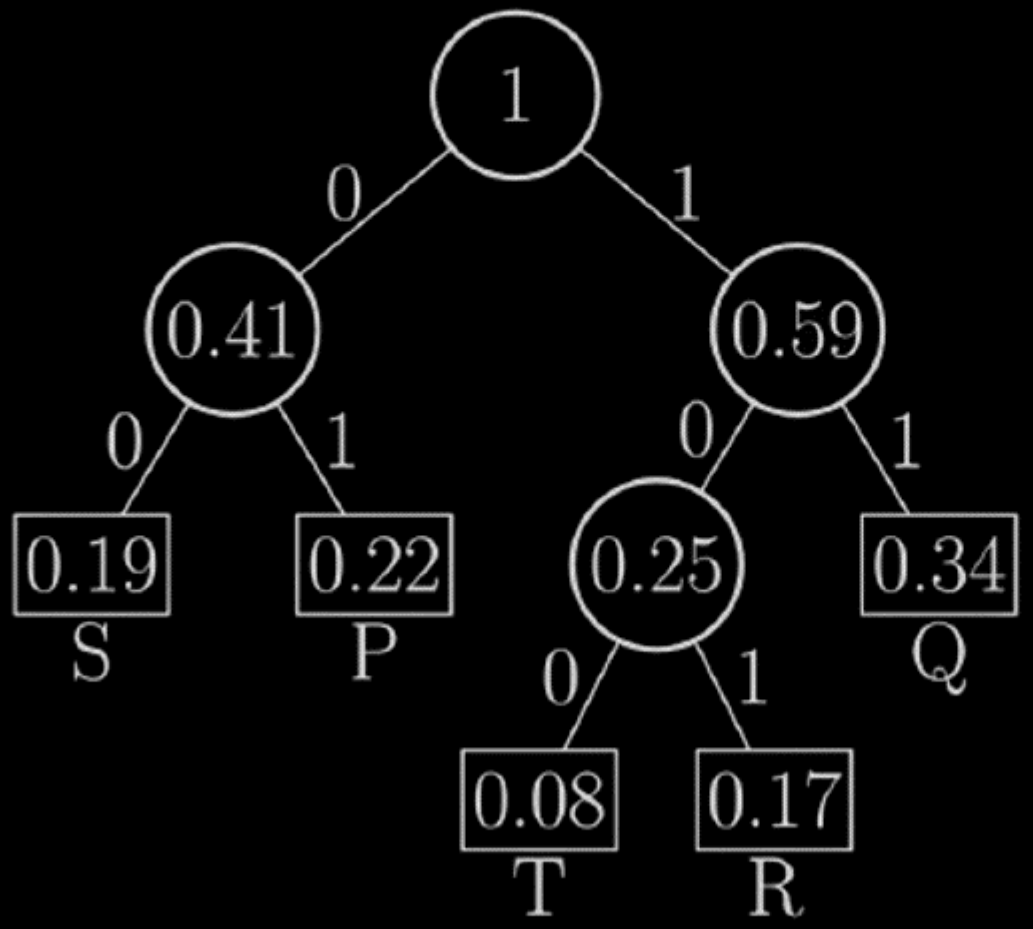
Expected length 2.25

100 x 2.25 = 800 bits

(22x2 + 34x2 + 17x3 + 19x2 + 8x3) = 225 bits

If a message of 100 characters over X is encoded using Huffman coding, then the expected length of the encoded message in bits is 225.

P:01
Q:11
R:101
S:00
T:100



GATE 2021 Set-II | 2 Marks Question

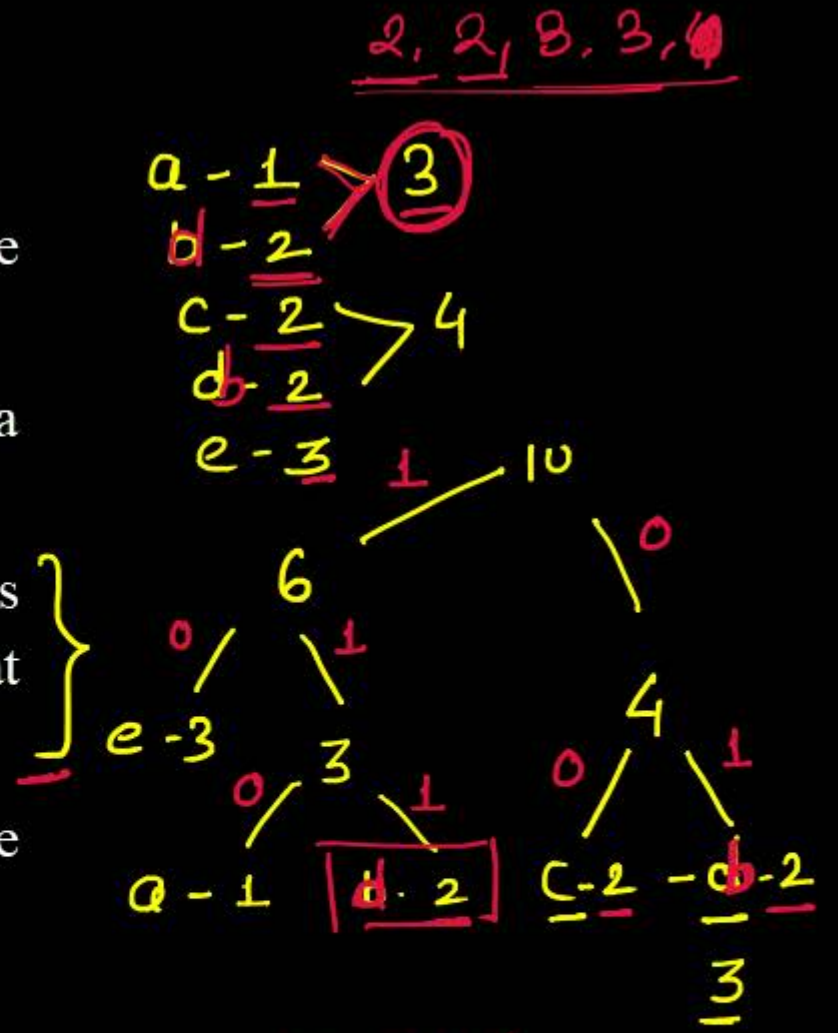
Consider the string *abbccddeee*. Each letter in the string must be assigned a binary code satisfying properties:

1. For any two letters, the code assigned to one letter must not be a prefix of the code assigned to the other letter.
2. For any two letters of the same frequency, the letter which occurs earlier in the dictionary order is assigned a code whose length is at most the length of the code assigned to the other letter.

Among the set of all binary code assignments which satisfy the above two properties , what is the minimum length of the encoded string?

(A) 25 (B) 23 (C) 21 (D) 30

(B)



a - 110 (3)
b - 111 (3)
c - 00 (2)
d - 01 (2)
e - 10 (2)

3+6+4+4+6

= 23