Heap:

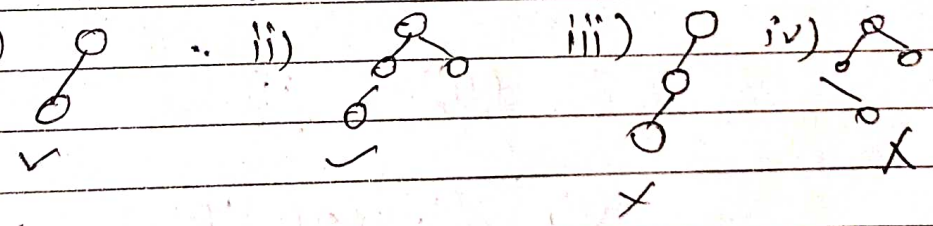
	Insert	Search	Find num	Delete num
Unsorted array	O(1)	O(n)	O(n)	O(n)
Sorted array	O(n)	O(logn) (Binary Search)	O(1)	O(n)
Linked list	O(1)	O(n)	O(n)	O(n)
min Heap	O(logn)		O(1)	O(logn)

Binary Tree:- every node will have at most 2 children.

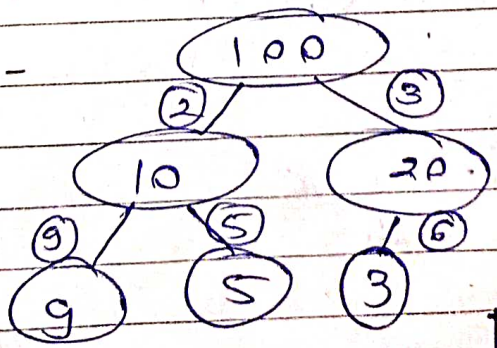
Almost complete Binary Tree:- The leaf should be available only in last and second level.

→ If all leaf is in one level then it will be filled from left to right.

e.g.:- i) ✓ ii) ✓ iii) ✗ iv) ✗



Max Heap:-

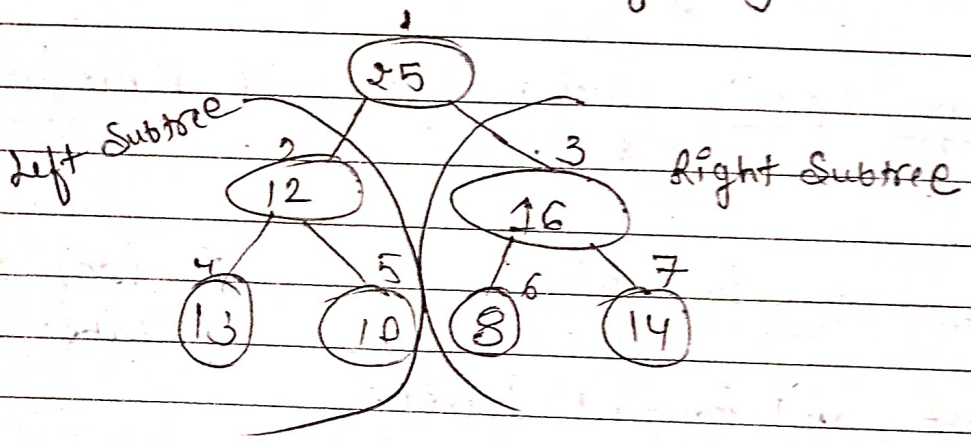


100	10	20	9	5	3

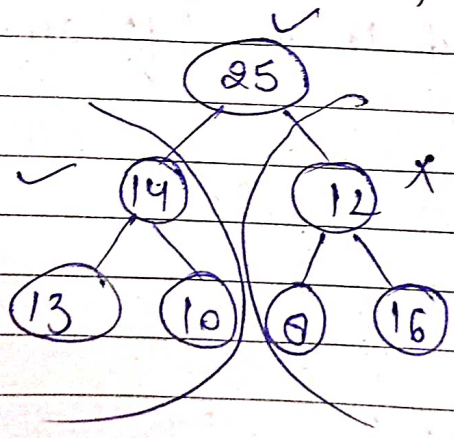
Left child = index of parent × 2
Right child = " " " " × 2 + 1

Parent = [index of Root/2]

A = 25, 12, 16, 13, 10, 8, 14
A. maxHeapSize = 7
A. length = 7 (total no. of keys)

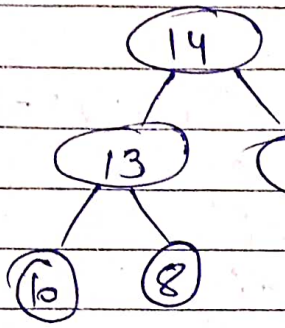


A = 25, 14, 12, 13, 10, 8, 16



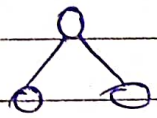
A. maxHeapSize = 2
A. length = 7.

(A) = 14, 13, 12, 10, 8

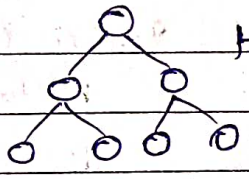


=> Single node max ya min dono ho skta hai.

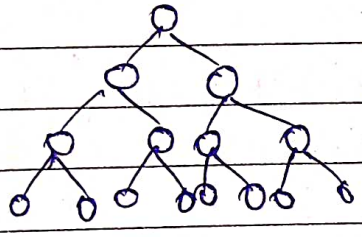
A.maxHeapsize = 2
A.length = 5



H(T) = 1



H(T) = 2 repeat to the right



H(T) = 3

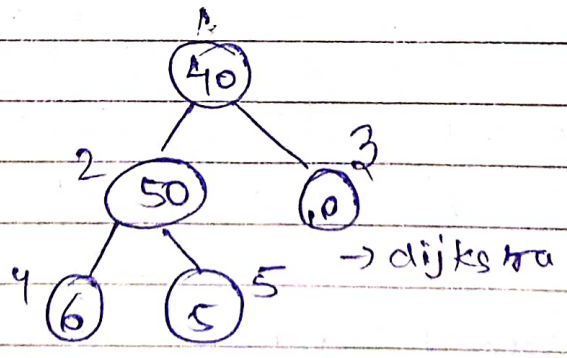
H	1	2	3	4
max^m no. of	3	7	15	31

Max no. of elements = $(2^{h+1}-1)$
Height of Binary tree = $O(\log n)$

```
MAX_HEAPIFY (A,i){
    l = 2i
    r = 2i+1
    if ( l ≤ A.heapsize and A[l] > A[i])
        largest = l;
    else largest = i;
    if (largest ≠ i)
        exchange A[i] with A[largest]
        MAX_HEAPIFY (A, largest)
}
Build_HEAP
Build_MAX_HEAP(A){
    A.HeapSize = A.length
    for (i = [A.length[2] down to 1]
        MAX_HEAPIFY(A,i);
```

Ques: from where the leaf is starting

$\left(\frac{n}{2}\right)+1$ to n

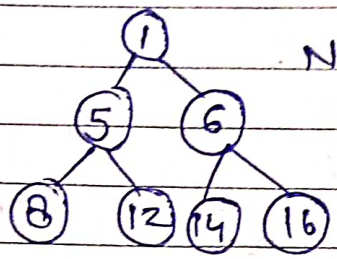


$\left(\frac{5}{2}\right)+1 = 3$ leaf node

12/09/23

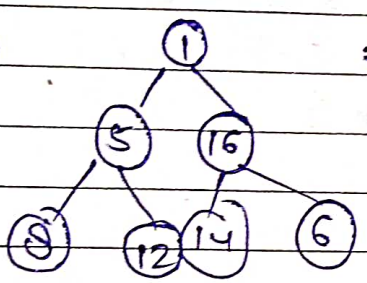
We should know about the leaf and non leaf node to convert your tree into min or max heap.

Convert it into a tree → 1 5 6 8 12 14 16

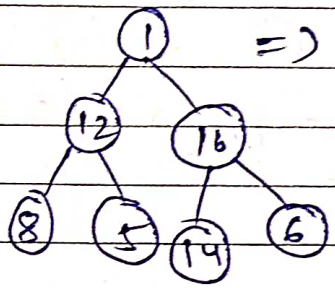


Non leaf node =

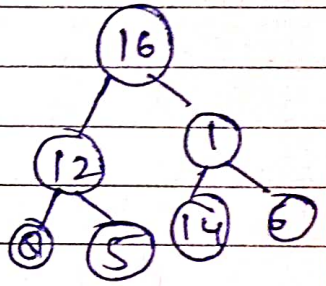
=>



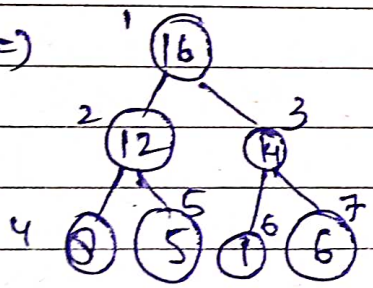
=>



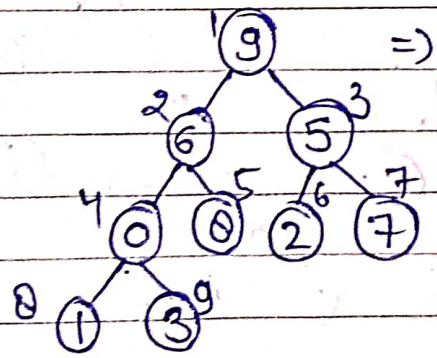
=>



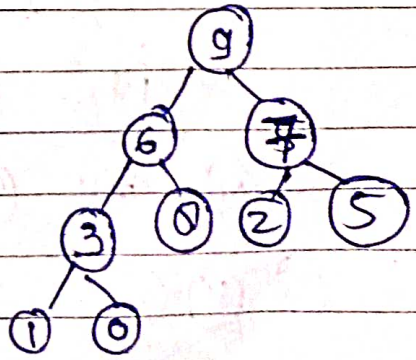
=>



9 6 5 0 8 2 7 1 3 =>



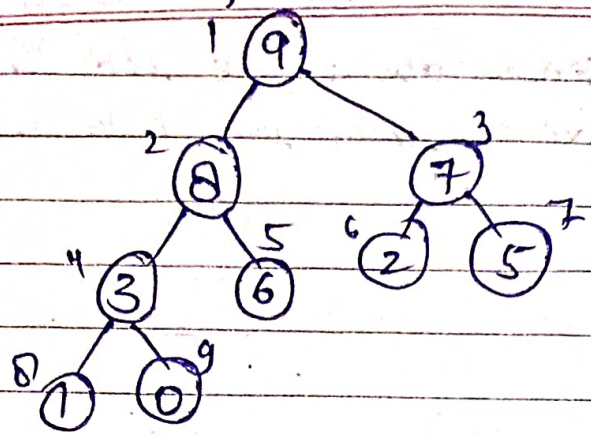
=>

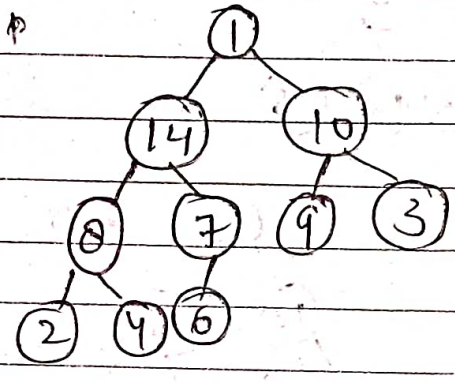


Non Leaf Node = 1, 2, 3, 4

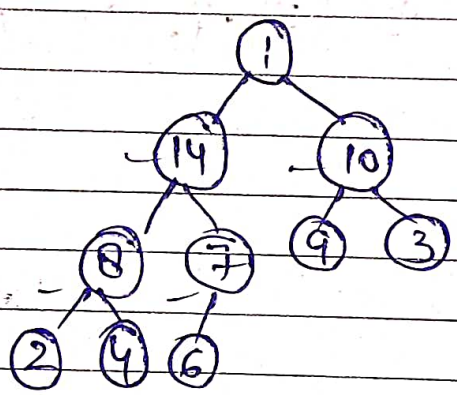
Leaf Node = 8, 9, 5, 6, 7

=>

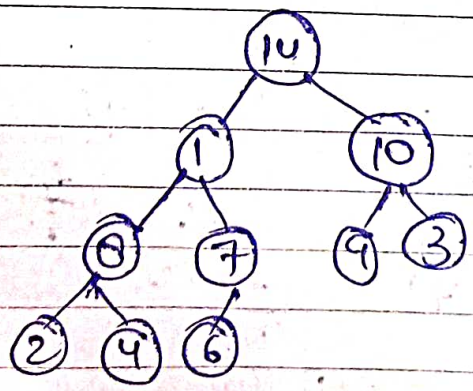




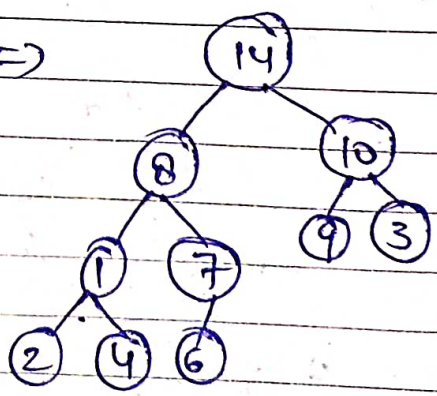
Soln



=>

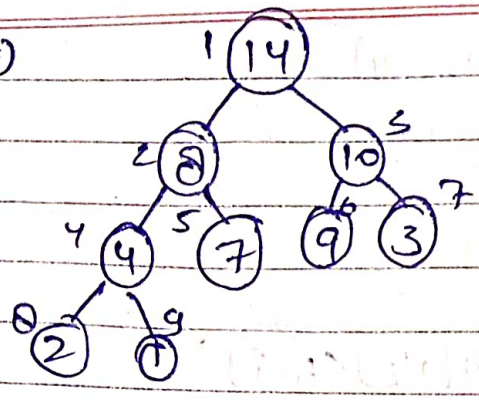


=>



Date :
Page :

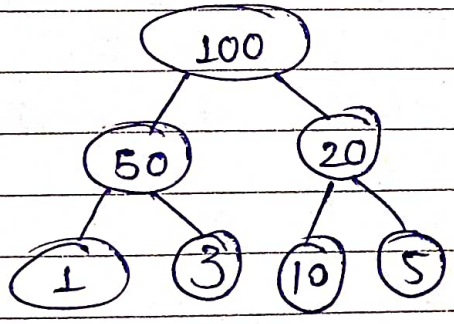
=)



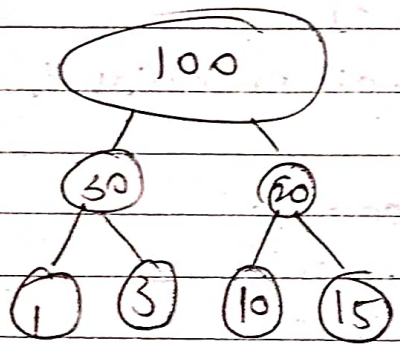
T.C.= O(logn)

```
HEAP_EXTRACT_MAX(A)
{
    if A.heapsize < 1
        error "heap Underflow".
    max = A[1]
    A[1] = A[A.heapsize]
    A.heapsize = A.heapsize - 1
    MAX_HEAPIFY (A, 1)
    return max
}
```

100 50 20 1 3 10 5



```
HEAP_increase_Key ( A, i, Key)
{
    if ( Key < A[i])
            error!
    A[i] = Key
    while i > 1 and A[i/2] < A[i])
            exchange A[i] and A[i/2]
            i = i/2
    }
```



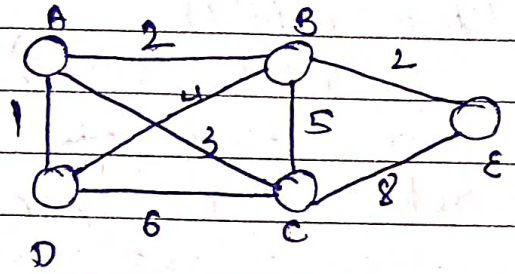
Find max
Delete max
Inserting new node
Increase key
Decrease key

Two ways → Insert new element, change any existing value.

Greedy Algorithm

Optimization Problems:-

Optimization means decrease cost and min. risk results in maximum profit and maximum reliability.



Optimization

Greedy
We cannot solve all optimization problems using greedy approach

Dynamic Programming
with dynamic we can solve all optimization problem but submits a exponential time which is not preferable

$O(2^n)$ ✗

I) Knapsack Problem:-

	obj 1	obj 2	obj 3	m=20
Profit	25	24	15	obj(n)=3
weight	18	15	10	

$\frac{25}{18} = 1.3$ $\frac{24}{15} = 1.6$ $\frac{15}{10} = 1.5$

16
15 ⟌ 240
15
90

3.2
15 ⟌ 48
45
30

18/15

Date :
Page :

(maximum profit)
greedy about weight

	2×24/15 = 3.2
obj1	~~18~~ 18

20

~~Total = 21.2~~

25 + 2×24/15 = 20.4

(least weight)
(greedy about profit)

	24/15 ×10 = 16
obj3	10

~~Total = 10+16 = 26~~

Total = 15 + 24/15 × 10
= 31

16
15+16
= 1

240
15
90

Third approach :-

Ratio Approach

	15×5
	15

= ~~15 + 22.5 = 18.33~~
15 + 7.5 = 22.5
24 + 15 7.5 ×15
10
31.5 Ans

13/09/23

Algo of Knapsack –

```
Knapsack                    (m = size of bag)
{
    for i=1 to n
        compute Pi/wi
    Sort objects in non increasing order of p/w
    for i=1 to n
    {
        if (m>0 && wi<=m)
            m = m-wi
            P = P+Pi
        else break
    }
```

if (m > 0)

$$P = P + P_i \left(\frac{m}{w_i}\right)$$

{

T.C := $O(n \log n) + O(n) + O(n) + O(1)$ (sort: $O(n \log n)$; $i \to n$: $O(n)$; $i \to n$: $O(n)$; if: $O(1)$)

= $O(n \log n)$

m = 15, n = 7

objects	1 ✓	2	3 ✓	4	5 ✓	6 ✓	7 ✓
profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
P/w	5	1-6	3	1	6	4.5	3

Sorting objects according to ratio P/w :-

(5, 1, 6, 3, 7, 2, 4)

2
7
3
6
1
5

Profit = 6 + 10 + 18 + 15 + 3 + 5 × 2/3

= 52 + 5 × 1/3

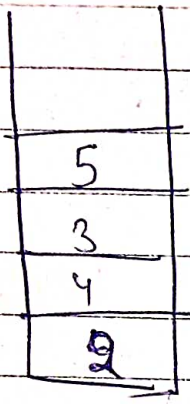
= 55.3

18
15
33
15
46
52

m=15, n=5

Obj	1	2	3	4	5
P	2	28	25	18	9
W	1	4	5	3	3
P/W	2	7	5	6	3

2, 4, 3, 5, 1



~~15 11 8 3 1~~

16
16
0

Profit = 28 + 18 + 25 + 9
= 80 (A).

Huffman Coding

abcd

100

a	00	a - 50 — 0
b	01	b - 40 — 10
c	10	c - 5 — 110
d	11	d - 5 — 111

Without compression = 2 × 100 = 200

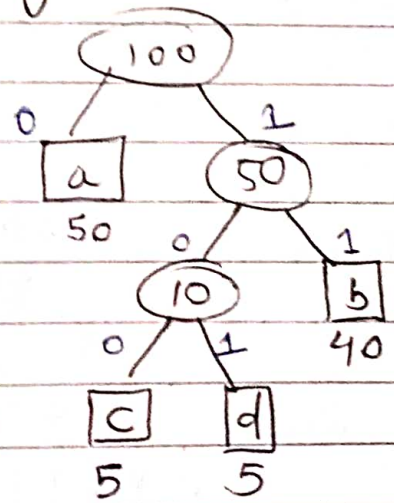
After compression
50 × 1 + 40 × 2 + 5 × 3 + 5 × 3
= 160 bits

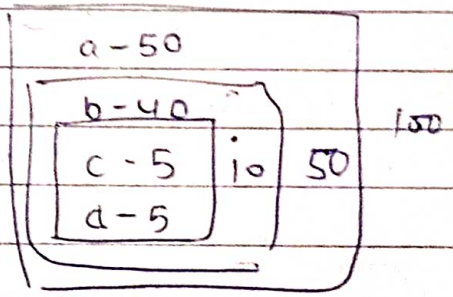
no. of occurences of abcd in the file.

Element with higher frequency should be near to root node and element with lower frequency should be far from root.

```
Huffman(c) {
    n = |c|
    make a minheap 'Q' with c
    for i = 1 to n-1
    {
        {allocate a new node z}
        z.left = x = Extract_min(Q)
        z.right = y = Extract_min(Q)
        z.freq = x.freq + y.freq
    }
    return (Extract_min(Q))
}
```

when there is variation in occurance of letters in dec., then only Huffmann coding is applicable.

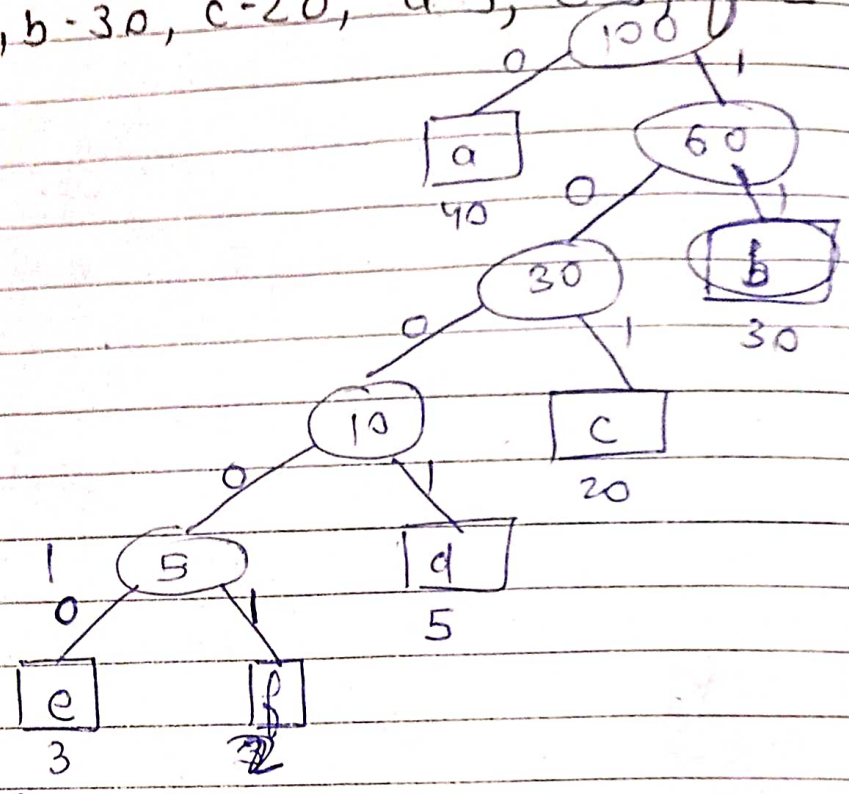




a - 0
b - 11
c - 100
d - 101

if add a in RHS instead of LHS
a - 1
b - 01
c - 000
d - 001

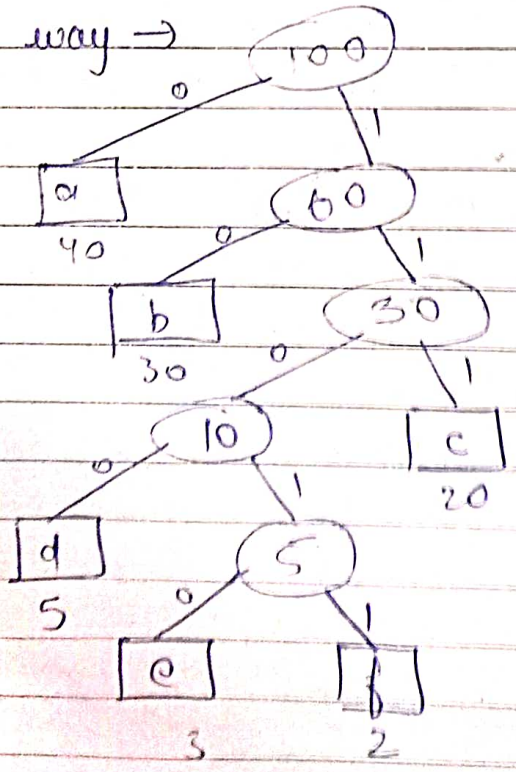
a-40 , b-30, c-20, d-5, e-3, f-2



First way →

a - 0 , b → 11, c → 101, d → 1001, e → 10000, f → 1000

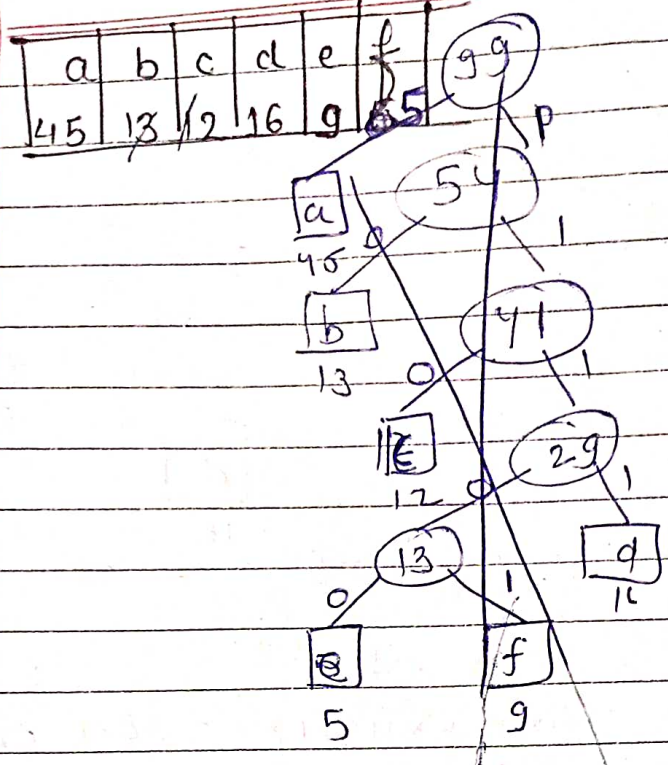
Second way →



a → 0
b → 10
c → 111
d → 1100
e → 11010
f → 11011

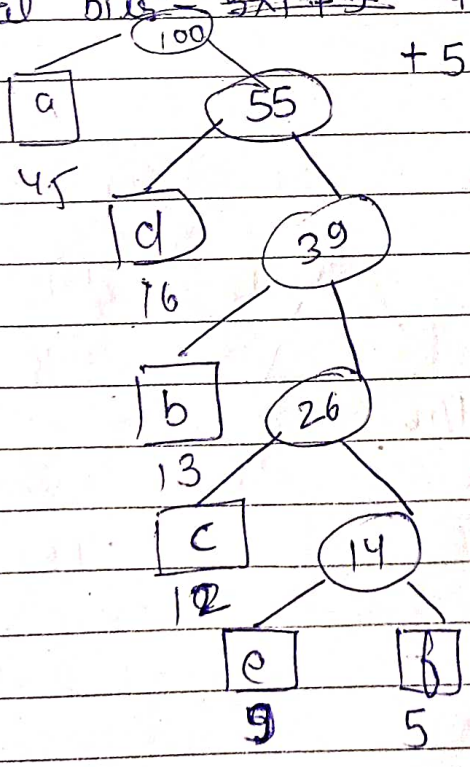
14/09/23

a	b	c	d	e	f
45	13	12	16	9	5

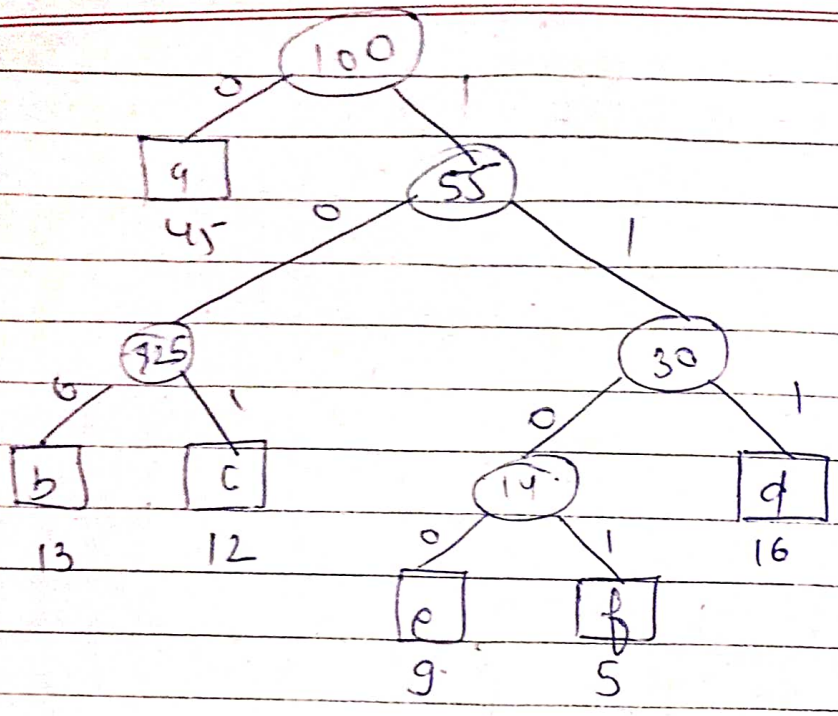


a → 0, b = 10, c = 110, d = 1111, e = 11100, f = 11101

Total bits = 45X1 + 13X2 + 12X3 + 16X4 + 5X5 + 9X5



45X1 + 16X2 + 13X3 + 12X3 + 9X1 + 5X1 = 234 Ans.

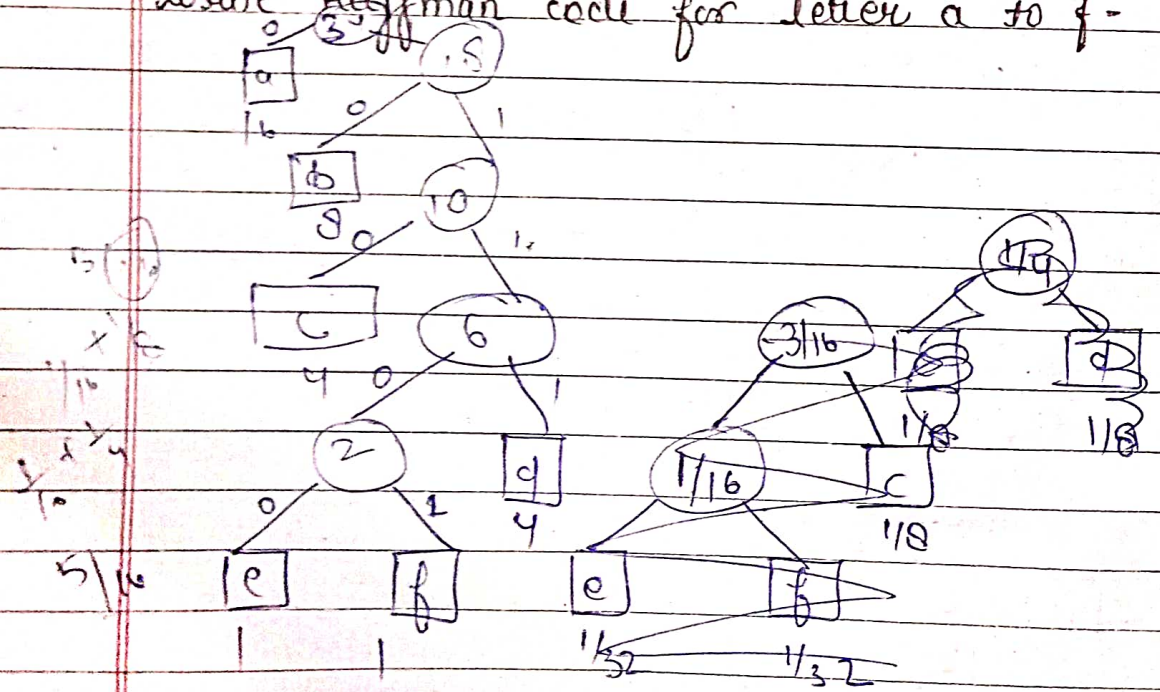


(forest)

$45\times1+3\times13+12\times3+9\times4+5\times4+16\times3=224$ Ans

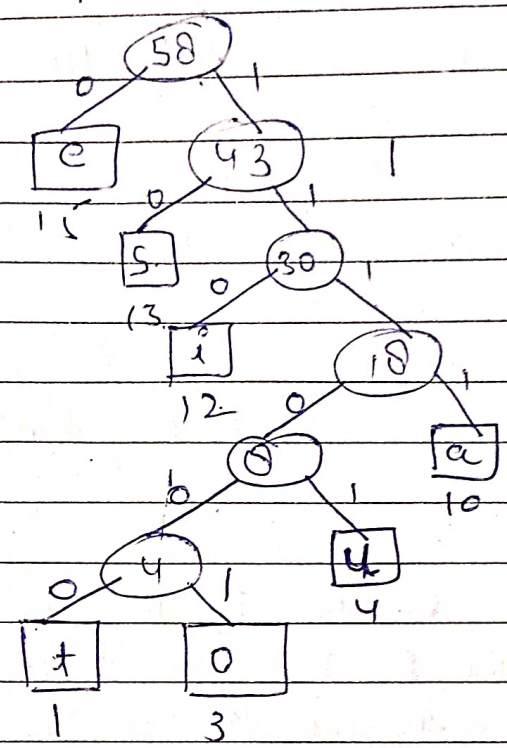
a	b	c	d	e	f
1/2	1/4	1/8	1/8	1/32	1/32
16	8	4	4	1	1

Write Huffman code for letter a to f-



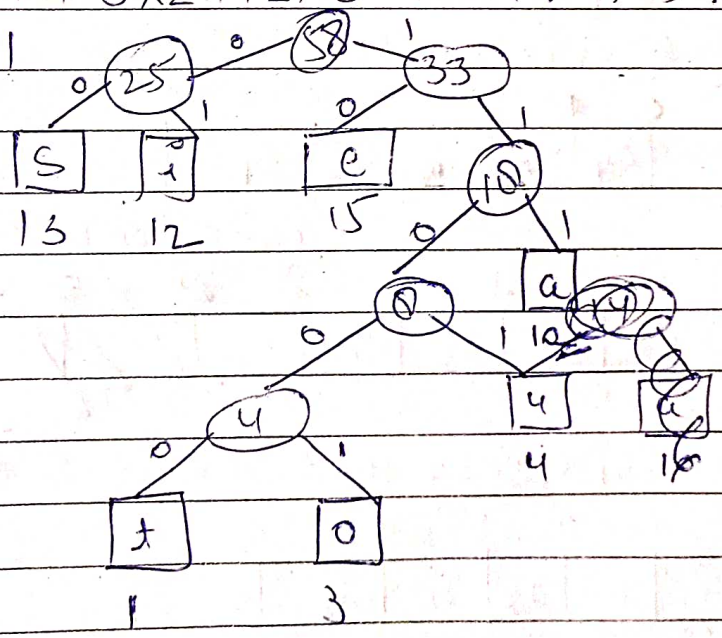
d	e	i	o	u	s	t
10	15	12	3	4	13	1

e	s	i	a	u	o	t
15	13	12	10	4	3	1



$15\times1 + 13\times2 + 12\times3 + 10\times4 + 4\times5 + 1\times6 + 3\times6$

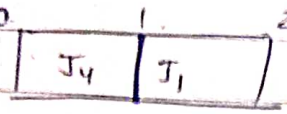
$= 161$



$13\times2 + 12\times2 + 15\times2 + 10\times3 + 4\times5 + 1\times5 + 3\times5$

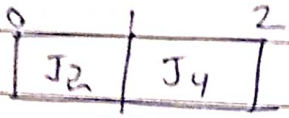
Job sequencing for deadlines :-

	J_1	J_2	J_3	J_4
Deadline	2	1	1	2
Profit	6	8	5	10

Soln :- i) 

(maximum profit is placed first)

$10 + 6 = 16$

ii) 

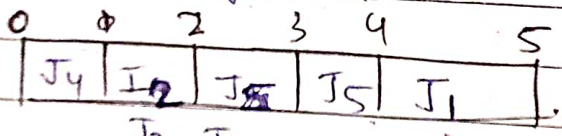
(a process which have given deadline is placed at last)

$8 + 10 = 18$

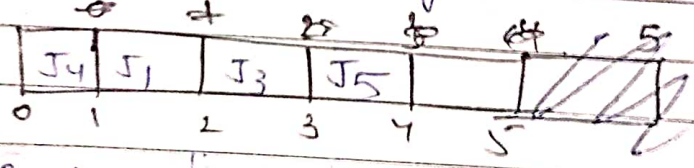
Qu :-

	J_1	J_2	J_3	J_4	J_5	J_6
Deadline	5	3	3	2	4	2
Profit	200	180	190	300	120	100

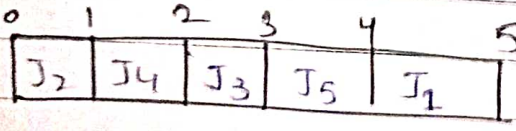
Max time = 5 no. of blocks = 5

i). 

J_3 J_2 Profit = 300 + 190 + 180 + 120 + 200 = 990

ii) 

Profit = 300 + 200 + 190 + 120 = 810

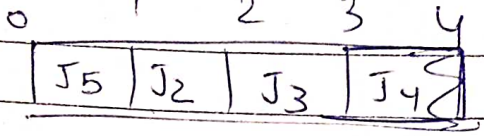
iii) 

T.C. = $O(n^2)$ + $O(n \log n)$ = $O(n^2 \log n)$

placing at farthest place — sorting

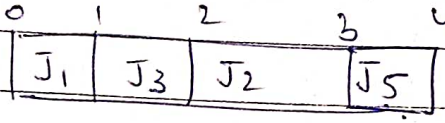
	J_1	J_2	J_3	J_4	J_5
Profit	2	4	3	1	10
Deadline	3	3	3	4	4

i)



Profit = 10 + 4 + 3 = 17 + 1 = 18

ii)



Ques:-

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

Solns:- Sort →

	J_3	J_9	J_7	J_5	J_4	J_8	J_1	J_6
Profit	30	25	23	18	18	16	15	10
Deadline								