| | ① find max | Delete Max | Insert |
|---|---|---|---|
| Max heap | O(1) | O(logn) ② | O(logn) ③ |

| ④ Increase key | ⑤ Decrease key | ⑥ find min. |
|---|---|---|
| Max heap O(logn) | O(logn) | O(n) |

⑦ Search O(n)

⑧ deletion $O(n + \log n) = O(n)$
(n: for deletion, log n: for rearrangement)

# If you want to Implement operation from 1-5. It's better to go for heap otherwise choose any other operation.

Stirling Approximation:-
$\lfloor n = O(n^n)$
$\lfloor n = \omega(2^n)$
$\log(\lfloor n) = \Theta(n \log n)$

## Heap Sort and Analysis.

```
Heap_Sort (A){
 Build-MAX-HEAP (A)
 for ( i = A.length down to 2) {
  exchange A[1] with A[i]
   A. heap-size = A. heap-size-1
   MAX_HEPIFY (A,1) }
}
```

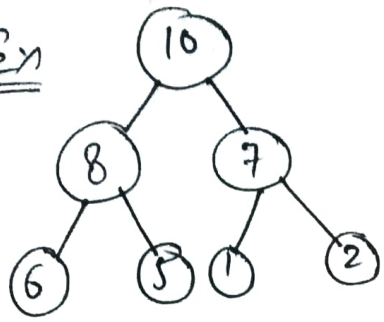
$T(n) = O(n \log n)$ → due to MAX heapify its log n and due to for loop its n so,

$\boxed{T(n) = O(n \log n)}$

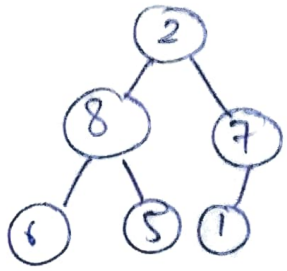
⇒ On $\frac{n}{2}$ times calling T(n) for MAX_HEAPIFY is O(logn), so in average case it will take max.

So, $O(n/2 \log n) = O(n \log n)$.

Ex




⇒




Apply max Heapify.

10

On the continue apply Heap Sort Algo we will get sorted array at last.

| 10 | 8 | 7 | 6 | 5 | 1 | 2 |
|---|---|---|---|---|---|---|

→ Start

| 1 | 2 | 5 | 6 | 7 | 8 | 10 |
|---|---|---|---|---|---|---|

→ After sorting.

## Problems on heap and heap sort.

Q In a heap with 'n' elements with the smallest element at root, the 7th smallest element can be find in time?

After delete a min at root, next min come at top.

del 1st min → O(log n)

— 2 — →
— 3 — →
— 4 — →
— 5 — →
— 6 — →

↓ O(log n)

find 7th — O(1)

Insert del from 1–6 again

= 6 * O(log n)

So,

$T(n) = 6 * O(\log n) + 6 * O(\log n) + O(1)$

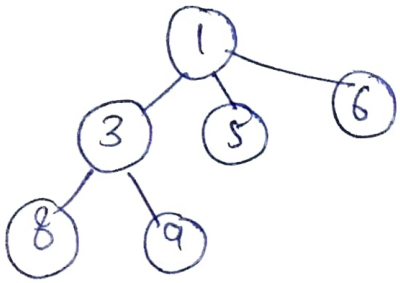
$= O(\log n)$

Q In a binary max heap contain n numbers, the smallest element can be found in time

Soln min. will found at the leaf and for leaf will lie btw $\lceil n/2 \rceil + 1$ to n. So total no. of comparison will be $O(n/2) = O(n)$.

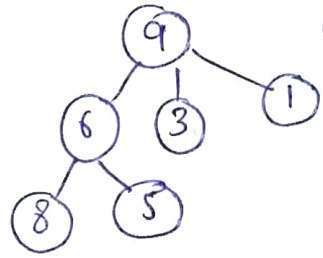
Following array sequence is given. which can be used to form a ternary binary tree. find max. heap.

① 1 3 5 6 8 9




(min heap)

② 9 6 3 1 8 5

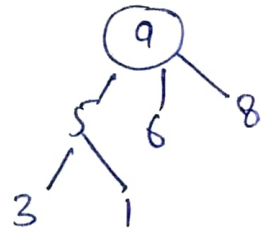



(Not a max heap)

③ 9 3 6 8 5 1

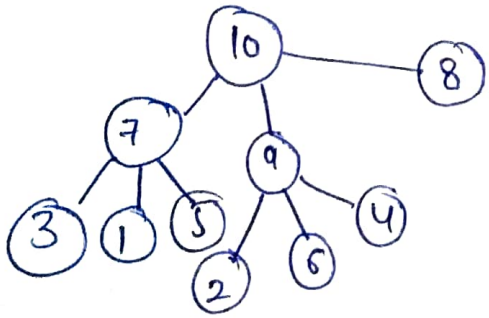
(Not a max heap)

④ 9 5 6 8 3 1




→ ternary max heap

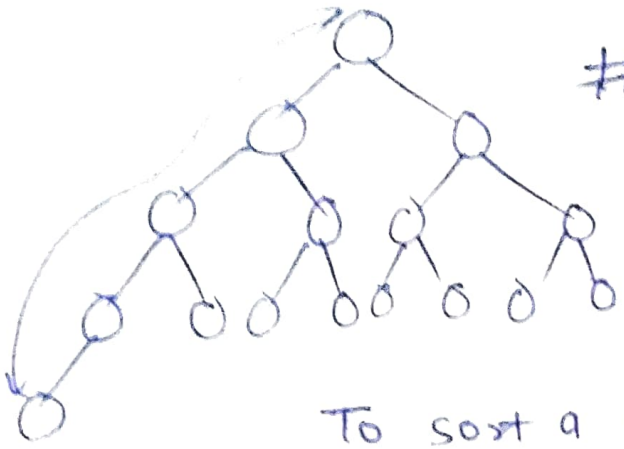
then add 7, 2, 10, 4 to ternary max heap.




10 7 9 8 3 1 5 2 6 4

final o/p.

Consider the process of inserting an element into a max heap. If we perform a binary search on the path from new leaf to root to find the position of newly inserted element, the no. of comparison performed are:-



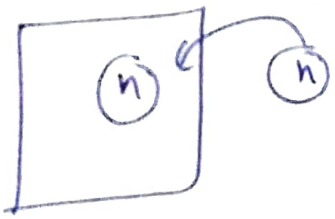
# In worst case total no. of elements in sequence so search
⇒ $\log n$ = (height of tree)

To sort a no. in binary search
no. of comparison required = $O(\log n)$

So, total no. of comparison = $O(\log \log n)$.

Q we have a binary heap of 'n' elements and wish to insert 'n' more elements (not insert necessarily one after another) into this heap. The total time required for this is-

Case-1 Insert one after another




Assume n elements are in heap and you want to insert more in heap one by one.

So for one element it will take $\log n$
for n element = $O(n \log n)$.

Case-2 Consider nothing in Heap and you added n more elements to array first so that array contains 2n elements. The apply Build_Heap Algorithm.

for n elements Build_Heap will take $O(n)$
for 2n ——— $O(2n)$

not inserting one after other but inserting together.

So, Time required = $O(n)$.

Gate! 2016

Assume that algorithm consider to sort a sequence in ascending order and the given input is already in ascending order. which of the following is true

① Quick sort runs in $O(n^2)$ ✓

② Bubble ——— $O(n^2)$ ✗ Error

③ Merge ——— $O(n)$ ✗ $O(n \log n)$

④ Insertion ——— $O(n)$ ✓

| | worst case |
|---|---|
| Insertion sort | $O(n^2)$ |
| Quick sort | $O(n \log n)$ |
| Merge sort | $O(n^2)$ |

## Bubble Sort Algorithm

```
BS ( int a[], n){
  int i, j, swap;
for (i=0 ; i<n ; i++) {
    swap=0;
for (j=0 ; j<(n-i-1) ; j++){   # for pairwise comparison
   if (a[j] > a[j+1]) {
   swap (a[j] , a[j+1]);
    swap=1; } }
 if (swap==0)
    break;
 }
}
```

Complexity:- In worst Case no. of comparison required

$= (n-1) + (n-2) + \cdots + 2 + 1.$

$= \frac{n(n-1)}{2} = O(n^2)$. [ A reverse sorted array.

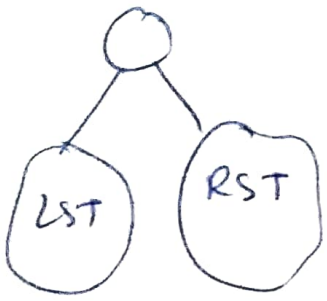
Best Case:

After 1st iteration array got sorted so total no comparison $= n-1$

$= O(n)$.

## Questions on heap

Consider a complete binary tree where the left and right subtree are max-heap. The lower bound for the no. of operation to covert a tree to heap.

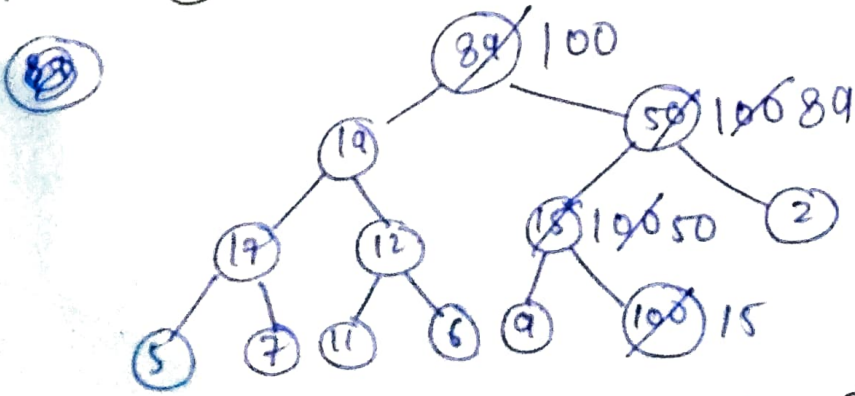



LST and RST are already max heap but not root
So apply Max_Heapify at root node.
In worst case root node may be transfer to leaf so $\log n$ Comparison or in best case it may be maximum so no. transfer.

So, no. of transfer will lie between (0–$\log n$)
In worst case → $O(\log n)$
Here, $\boxed{\Omega(\log n)}$.

Q Consider the following array of elements
< 89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100 >. The min. number of interchanges needed to convert it into a max heap.

① 4 ② 5 ③ 2 ④ 3 Ans




1 + 1 + 1 = 3

Gate-2016 → Max. height of node of 9 | 9mp Question

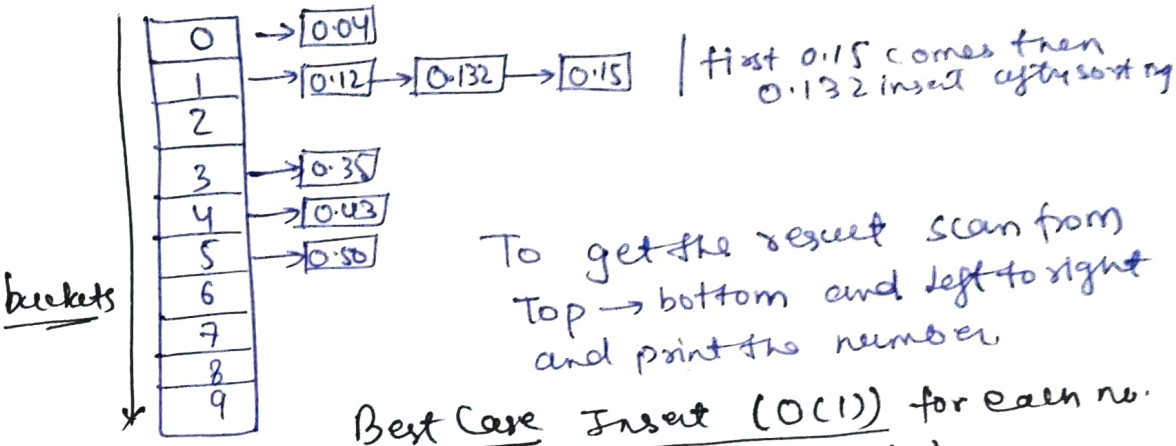
Gate-2018

## Bucket Sort :-

Sort a large set of floating point numbers which are in range 0.0 to 1.0 and are uniformly distributed accross the range ?

0.35 , 0.12 , 0.43 , 0.15 , 0.04 , 0.50 , 0.132 | Given Sequence

Consider a collection of bucket have index (0-9) as




To get the result scan from Top → bottom and left to right and print the number.

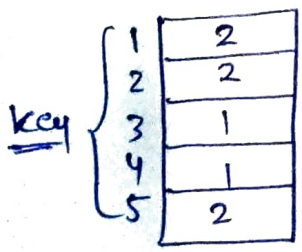
Best Case Insert (O(1)) for each no. for n no. O(n).

Space complexity = O(n+k).
- for no.
- for bucket index.

In worst Case O(n²).

**Counting Sort** :- Only apply when the range of the number is known.

say Range is (1-5)
and numbers are 2 2 3 4 1 5 1 5




Insert the no. of count of number its appear in the sequence.
And after insert all numbers print the table according to count.

O/p → 1 1 2 2 3 4 5 5

Range is use to give keys to table.
Let the no. of keys K and no. of inputs are n.
then, time complexity is —

$T(n) = O(n+k)$

$O(n)$ for scanning all no. once and $O(k)$ to scan the keys of table.

So, $\boxed{T(n) = O(n+k)}$

Space Complexity = Extra space = table of k slots

$\boxed{S(n) = O(k)}$.

Disadvantage

{ You can't have any key beyond range. and the space in table increase with range.

Ex → 1 , 1000, 2, 3

There are only 4 numbers but we need 1000 slots bcoz of 1000.

**Radix_Sort** → First decide the base of number.

```
Radix_sort (A, d)
// Each key in A[1...n] is a d-digit number.
// Digits are numbered 1 to d from right to left
for(i = 1 to d) do
Use a stable sorting algorithm to sort A on digit i.
```

↳ Use counting sort if range is known.

Ex Let the numbers are

804 , 26 , 5 , 64 , 52 , 1

Step-1 Largest no. 804 is of 3 digits ($d=3$) so first make all no. of 3 digits as

804 , 026 , 005 , 064 , 052 , 001.

Ex In

| 3 | 2 | 1 (i) |
|---|---|---|
| 8 | 0 | 4 |
| 0 | 2 | 6 |
| 0 | 0 | 5 |
| 0 | 6 | 4 |
| 0 | 5 | 2 |
| 0 | 0 | 1 |

Step-2
Sort the values on value at ($i=1$)

| | |
|---|---|
| 0 | 0 1 |
| 0 | 5 2 |
| 8 | 0 4 |
| 0 | 6 4 |
| 0 | 0 5 |
| 0 | 2 6 |

⇔ no. comes first appear first

Step-3 then apply apply sorting according to index i=2 on already sorted array in Step 2 we get

0 0 1
8 0 4
0 0 5
0 2 6
0 5 2
0 6 4

Step-3 Again apply same procedure on Step-2 on i=3

0 0 1
0 0 5
0 2 6
0 5 2
0 6 4
8 0 4

} we get the sorted Array.

Analysis of Time Complexity of radix-sort (AID)

$T(n) = d\,(O(n+b))$

- $d$: no. of digits (Due to Loop)
- $O(n+b)$: Due to sorting Algo (Here counting sort)

And $\boxed{d = \log_b l}$ → b = base of no. system. l = largest no.

$T(n) = \log_b l \; O(n+b)$

$= \boxed{O(n \log_b l)}$

If $l = O(n^k)$ Assume

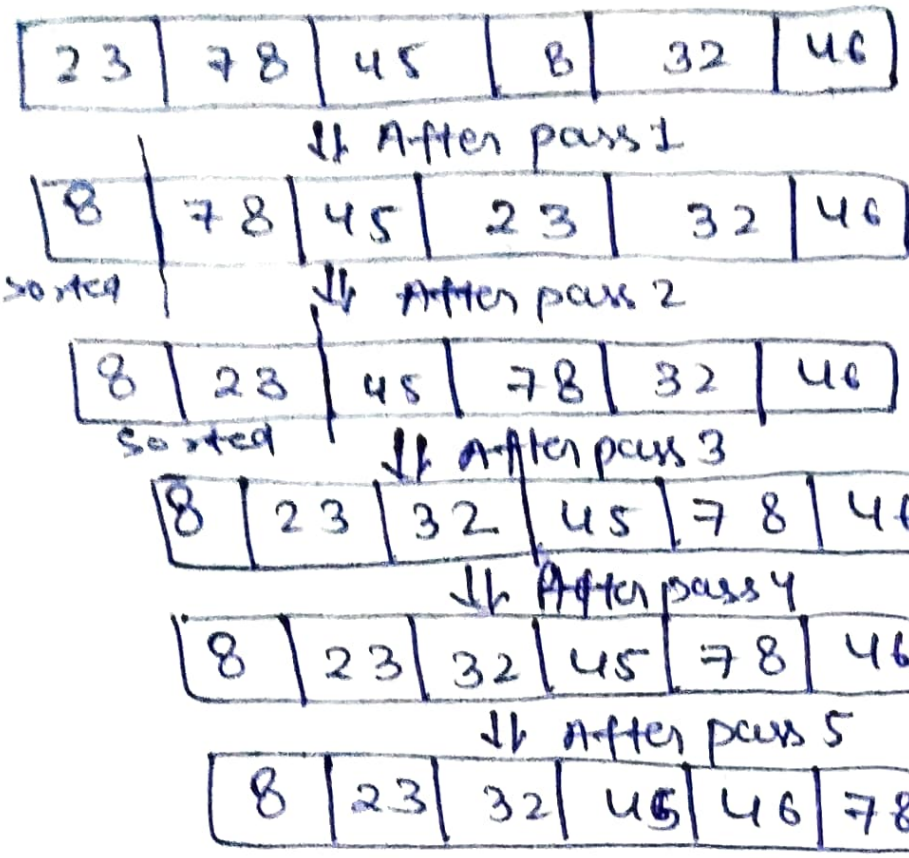
$T(n) = O(n \log_b n) \rightarrow \boxed{O(n \log n)}$

$\boxed{S(n) = O(b)}$ → Required for sorting Algorithms

**Selection Sort**

```
void selectionSort (A, n){
  int i, j, min, temp;
for (i=0; i<n; i++){
   min = i;
for (j=i+1; j<n; j++){
  if (a[j] < a[min])
    min=j; }

temp = a[i];
a[i] = a[min];
a[min] = temp; }}
```




No. of pass required = (n-1)

$T(n) = T(n-1) + O(n)$ → To decrease the problem size by 1.

$= O(n^2)$

$S(n) = O(1)$.

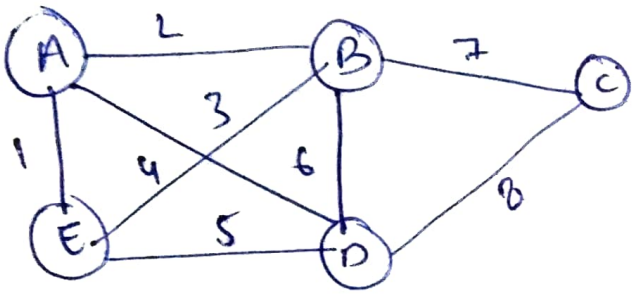
# Greedy Algorithms

## ① Introduction to Greedy Algorithms:-

Use to resolve optimization problems.

1. Minimize the cost
2. Maximise the profit
3. Max Reliability
4. Min Risk

Consider a graph with given weight.




To get min. path btw A and C we have $O(2^n)$ approaches

$n \rightarrow$ no. of nodes.

by using exhaustive search techniques.

But we want to solve it get polynomial solution $O(n^k)$. So we have two type of optimization techniques.

1. Greedy → Applicable to only some problems with specific properties Gives. $O(n^k)$ solutions.
2. Dynamic Progamming → Applicable to all problems. Mostly give $O(n^k)$ solution but sometime give $O(2^n)$ solution. If gives $O(2^n)$ solution then it is better to use exhaustive search techniques.

# Greedy Knap Sack Algorithm

(Knapsack) → Bag

**Problem.** m = 20 (Bag of size 20 units) → capacity
n = 3 (objects type)

| | Ob1 | Ob2 | Ob3 |
|---|---|---|---|
| Profit | 25 | 24 | 15 |
| weight | 18 | 15 | 10 |

On selling 18 units of ob1 we get profit of 25.

What are the objects that you want to place in bag to get maximum profit.

**Soln** Assume we are greedy about profit

**Case-1**

| | Weight | Profit |
|---|---|---|
| Ob1: | 18 | 25 |
| Ob2: | 2 | (24)(2/15) |
| | 20 | 28.2 |

Not the answer just an assumed case.

Take one with max profit first.

**Case2**: Assume we are greedy about weight
Take one with least weight first.

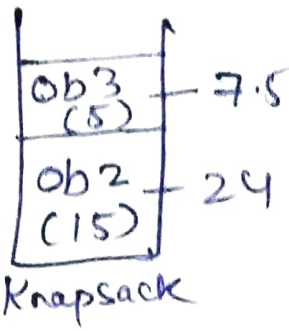
| | W | P |
|---|---|---|
| Ob3: | 10 | 15 |
| Ob2: | 10 | (24x10/15) |
| | 20 | 31 |

Another assumed case

In this particular problem we got that if we are greedy about weight, then we are earning more profits. But we can't predict the answer till now bcoz we don't consider all the cases.

Case-3 → Don't go either by profit or weight go with ratio of Profit/weight

| | ob1 | ob2 | ob3 |
|---|---|---|---|
| Profit | 25 | 24 | 15 |
| weight | 18 | 15 | 10 |
| Profit/weight | 1.4 | 1.6 | 1.5 |




| | w | p |
|---|---|---|
| ob-2 | 15 | 24 |
| ob-3 | 5 | 7.5 |
| | 20 | 31.5 |

In this case we always get the maximum profit.

Algo → Greedy knapsack

```
GK {
for i = 1 to n;                    ] T(n) = O(n)
Compute pi/wi;                     ]
Sort objects in non-increasing order of p/w.   ] T(n) = O(nlogn)
                                                  ↓ In best Algo.
for i = 1 to n
  if (m > 0 && wi <= m)            } T(n) = O(n) in worst case
      m = m - wi;
      P = P + Pi;
  else
      break;
if (m > 0)                         } = O(1)
    P = P + Pi (m/wi);
}
```

T(n) = O(n) + O(nlogn) + O(n) + O(1)

T(n) = O(nlogn)

Explaination using Examples ⟶ Contn.

$m = 15$ , $n = 7$

| 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 |

After sorting in order of max P/w ratio we get sequence of order.

At the end Object ( 5, 1, 6, 3, 7, 2, 4)

$m = 0$

$P = 52 + 5(2/3)$

$= 55.33$.

Max profit

$52 + 5(2/3)$

| | | | |
|---|---|---|---|
| $m = 0$ | $P =$ | | |
| $m = 2$ | $P = 52$ | 2 | obj 2 |
| $m = 3$ | $P = 49$ | 1 | obj 7 |
| $m = 8$ | $P = 38$ | 5 | obj 3 |
| $m = 12$, | $P = 16$ | 4 | obj 6 |
| $m = 14$, | $P = 6$ | 2 | obj 1 |
| $m = 15$, | $P = 0$ | 1 | obj 5 |

If you want to do the problem using max heap instead of any sorting algorithm, then Build the max heap of $(P_i/w_i)$ list and then apply delete node.

So it will take $O(n) + O(n \log n) = O(n \log n)$

$O(n)$ ↓ Building Heap

$O(n \log n)$ ↓ Delete all n nodes (in worst case)

[ ये संभव है सारे element insert करने के बाद knapsack में ]

# Case-3rd

If weights are same then we can solve the problem using profit values only

# HUFFMAN CODES (Greedy)

Used to compress the files Ex → Zip file.

Scene. Assume a file with 100 characters with different frequence of a, b, c, d (मतलब abcd ही उस 100 characters में कई बार शामिल है)

If represent char in binary as

a = 00
b = 01
c = 10
d = 11

} means two 2 bits for each character

then 100 characters will take total 200 bits to send encoded message.

But in other method assume

| Represent | Character | frequence |
|---|---|---|
| 0 | a | 50 |
| 10 | b | 40 |
| 110 | c | 5 |
| 111 | d | 5 |

It means a → 1 bit
b → 2 bit, c, d → 3 bit

Total no. of bits required
= 50 × 1 + 40 × 2 + 10 × 3
= 160 bits

So we save 40 bits here.

I scene → Uniform encoding
II scene → Non-uniform encoding.

In Non-uniform encoding we have to take care of a problem that a binary code that represent any character should not be the prefix of any other binary code represents another character.

↳ Prefix Code

→ Contn.

```
Huffman (C)
{
    n = |C|
    make a min heap 'Q' with C }  O(n)
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
    Insert (Q, z)
return (Extract_min (Q))
}
```

Build Heap

2*(n-1) log n
for extract min
O(n log n)
for insert
(n-1) log n
= O(n log n)

→ O(1)

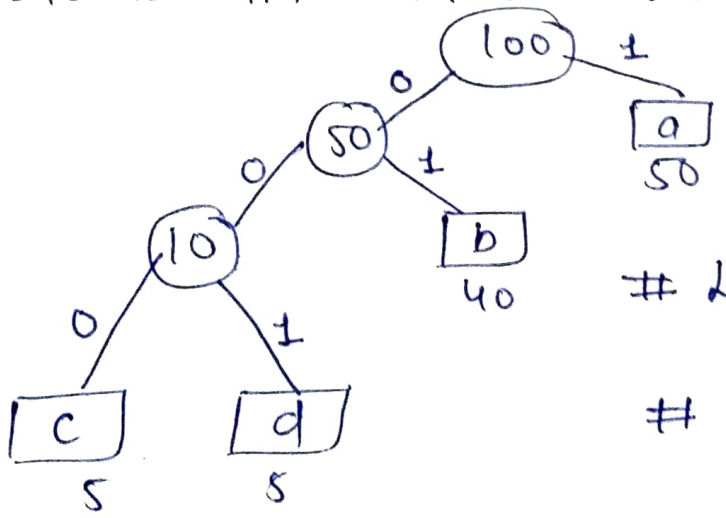
$T(n) = O(n \log n)$

$S(n) = O(n)$ → for trees only

**Ex** Assume n = 4

a - 50
b - 40
c - 5
d - 5
} character with freq.

Lets make a tree with given frequency.
Starts with min. and reach to max.




# Lesser child should be at left

# Root node value must be equal to total size.

So,

a = 1
b = 0 1
c = 0 0 0
d = 0 0 1


# Give value '0' to left child edge and '1' to right child edge and it can also be change.

→ Also the length of node

Total no. of bits required to encode the 100 char are = $(50\times1)+(40\times2)+(5)\times3+(5\times3)$

$= 160$ bits.

So, bits per character (b/c) $= \frac{160}{100} = 1.6$ b/c.

→ Weighted external path length (minimum).

Ex→

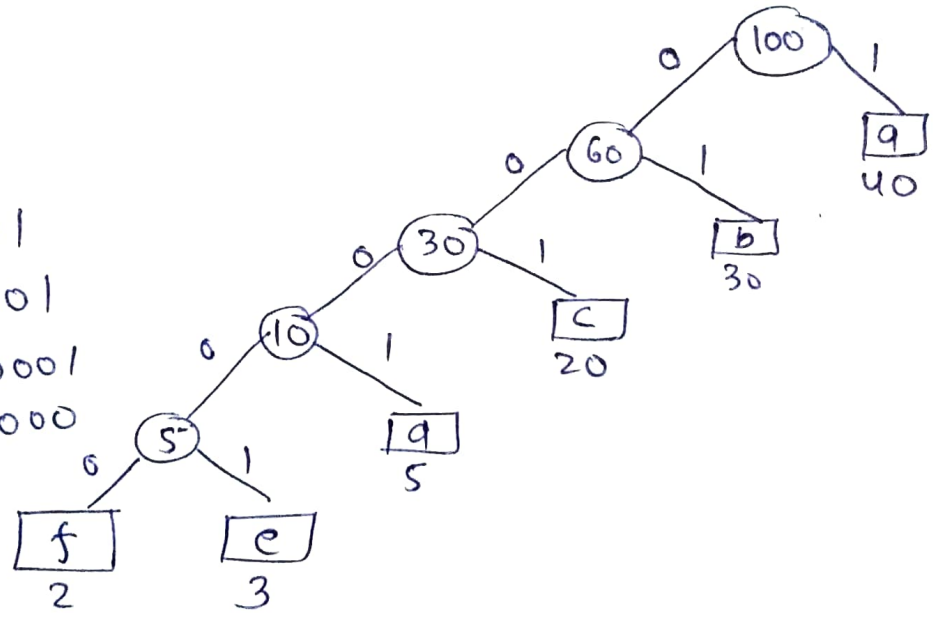
| | (freq.) |
|---|---|
| a | 40 |
| b | 30 |
| c | 20 |
| d | 5 |
| e | 3 |
| f | 2 |

Every node have value equals to frequency.

40 30 20 5 3 2

**I st way**

a → 1
b → 01
c → 001
d → 0001
e → 00001
f → 00000




So min. weighted external path length =

$40(1) + 30(2) + 20(3) + 5\times4 + 3\times5 + 2\times5$

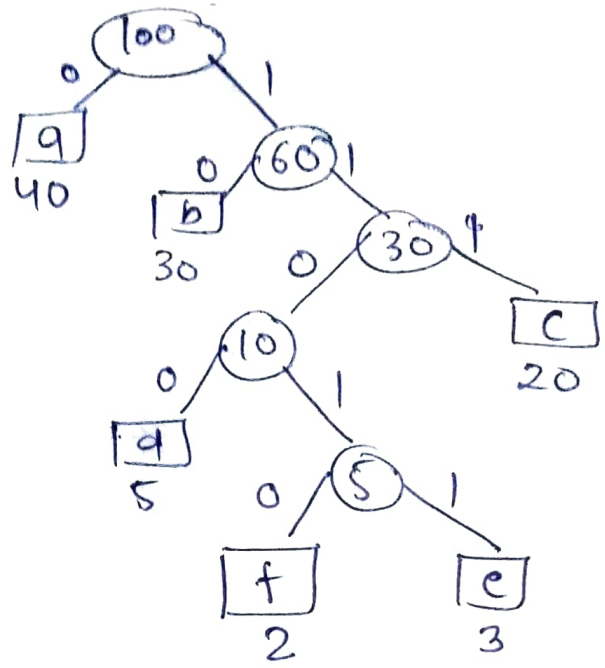
$= 40 + 60 + 60 + 20 + 15 + 10$

$= 205$ bits. $= 2.05$ b/c

To represent 6 no. we need min. 3 bits So total bits required = $100\times3 = 300$ bits

So we have bits gain of = 95 bits here.

II way




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

Again the output will same but the implementation is different

Algorithm Explaination

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

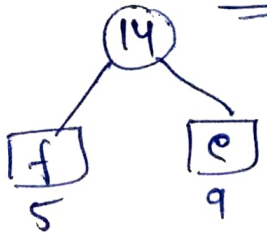
$c \in C$ → set of all characters

① Make a min-heap will all characters (Q)

Q → [45, 13, 12, 16, 9, 5] → not in order but on calling extract min it will give the minimum.

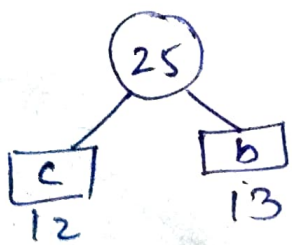
②

new Q (45, 13, 12, 16, 14)

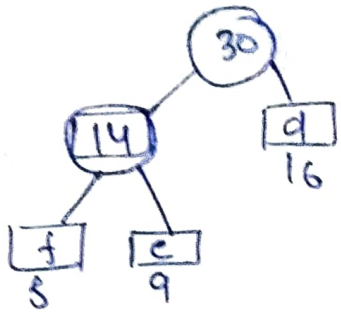



③

new Q (45, 16, 14, 25)

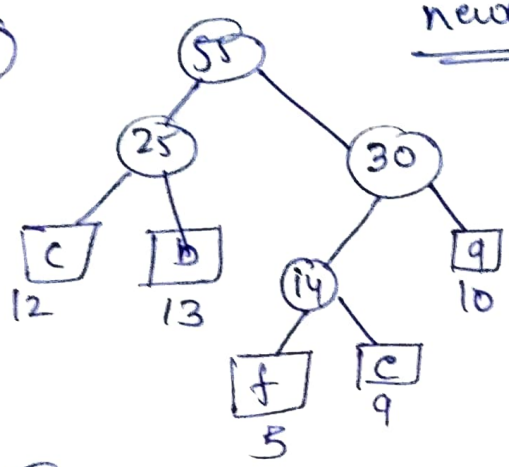


(3)




newQ (45, 25, 30)

(4)



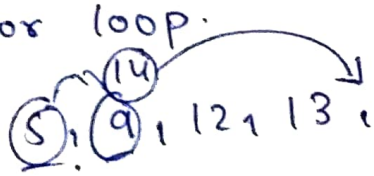

newQ(45, 55)

(5) Add node (55) and (45) → to new node 100 in tree of 4th step.

Ext. path weight = (45×1) + (3×12) + (3×13) + (16×3) + (4×5) + (9×4)

If you apply sorting algorithm instead of heap then problem will be -

① In best case sorting will take (nlogn) but in for loop.




You del, 5, 9 and then insert it into the correct position it will take O(n) time and since loop will execute (n-1) times. so it will take $O(n^2)$.

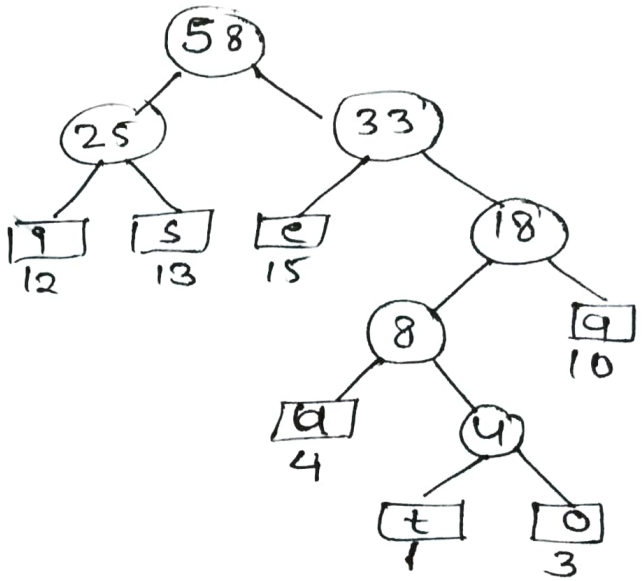
So, We are not using loop here

If you have to extract the no. and need to place it back to list use heap and if you have to extract the no. only use sorting.

Ex → Do the solution by self and try to verify with the given example soln.

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

Soln tree



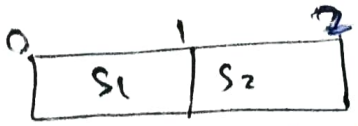

Hint:- हर बार same tree में Node, Add हों ये जरूरी नहीं है।

## Job Sequencing with deadlines

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

If you complete the job under deadline you will get the profit.

size of array = max. deadline



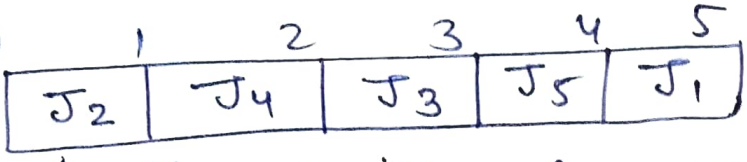

Case 1 for max profit
take $S_1 = J_4$
$S_2 = J_2$ (Can't take above deadline)

Case-2 $S_1 = J_4$, $S_2 = J_1$ } Profit = 16. (may or may not be max.

Case 3 $S_1 = J_2$, $S_2 = J_4$ | Profit = 18 (max.)

Ex-2

| | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ |
|---|---|---|---|---|---|---|
| D: | 5 | 3 | 3 | 2 | 4 | 2 |
| P: | 200 | 180 | 190 | 300 | 120 | 100 |




Start from max. profit and reaches to min. profit.

↳ Place the job as far as possible in deadline limit → gives the max. optimize solution.

Time complexity

① for sorting all Jobs (O(nlogn))

② Time for placing jobs in the slots in worst case is = O(n*n) = O(n²)

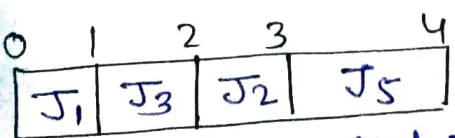
O(n) → for search
O(n) → for n Jobs
Total O(n²).

Start from deadline and traverse till first in worst case (n).

So, T(n) = O(nlogn) + O(n²)
= O(n²).

Ex-3

| Jobs | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ |
|---|---|---|---|---|---|
| P | 2 | 4 | 3 | 1 | 10 |
| D | 3 | 3 | 3 | 4 | 4 |




→ Solution

Net profit → 2+4+3+10 = 19.

Algo →

① Take the array O(1)
② Sort the array O(nlogn)
③ Insert the job under dealline as far as possible O(n²)

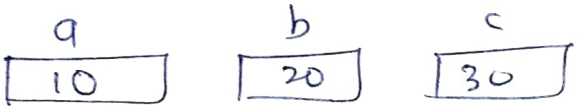
In example 3

① Deadline should be very close to No. of Jobs.

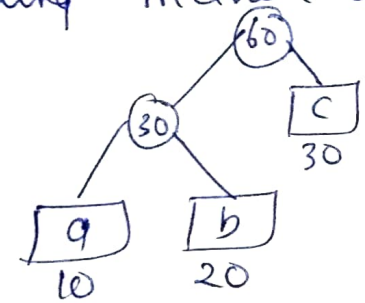
② If any Job have deadline 100 from 5 Jobs. It doesn't mean you create an array of 100 slots. Make array of size nearly equal to no. of jobs

## Optimal Merge Pattern :-

Take three files of different size as




then the above can be use to get the merged solution in min. time using Huffman coding method as-




Make the tree with min. external path length

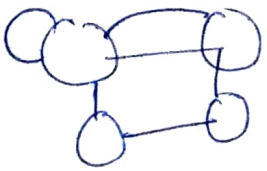
# File with least no. of record should be as far as possible from root and have no. of movement as max.

## Introduction to spanning tree and Kirchoff theorem.

Graphs: Collection of nodes and edges (V, E)

Simple Graph → Almost 1 edge btw two nodes.

Multi Graph → More than 1 edge between 2 nodes



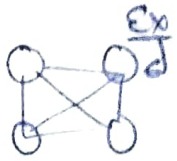
③ Null Graph :- No edges. Only Nodes.

In this topic we only consider Simple Graph.

Min. no. edge in simple Graph = 0

Max ——————— = $nC_2$

$= \left(\frac{n^2 - n}{2}\right)$




In worst case edge = $O(n^2)$ ← n = no. of vertices

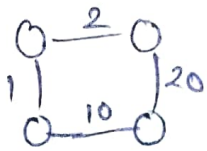
$= O(V^2)$

In worst case → $E = O(V^2)$

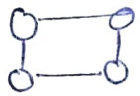
→ $V = O(\log E)$

## Spanning Trees

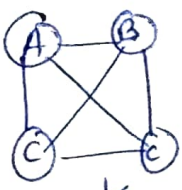
Weighted Graph → weights on graph edge




Unweighted Graph —

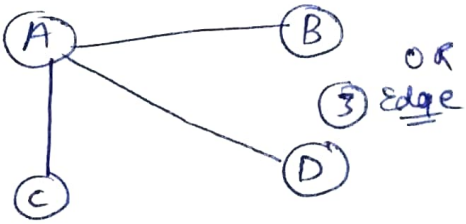


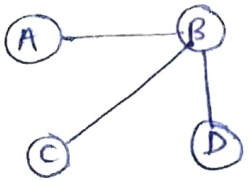
# min. of edges in graph so that we can cover all the nodes → spanning trees




$K_4$

Spanning Tree



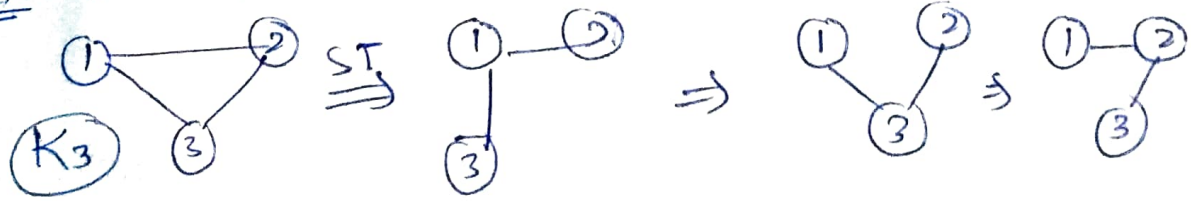




3 Edge

min. no. of edges required to connect n nodes = n-1.

# Spanning tree is also a sub-graph of given graph.

Ex → No. of spanning trees possible




# In case of non-labelled all are isomorphic because we can't distinguish.

## Complete Graph

## No of spanning tree possible

$K_4 \longrightarrow 16$

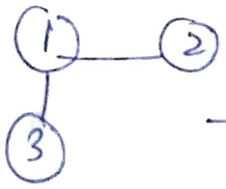
$K_3 \longrightarrow 3$

$K_n \longrightarrow \boxed{(n^{n-2})}$

Only for complete graph (all the min possible edges are there)

If graph is not complete then we will get less no. of spanning trees.

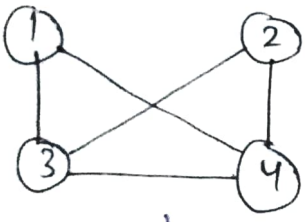
Ex




→ Here we will get only single spanning tree

No. of spanning trees possible for not-complete graph →

Ex




Graph is undirected so (1,4) and (4,1) both have one.

### Kirchoff Theorem.

① Make an adjacency matrix

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 |

Due to undirected graph

## In simple graph diagonals are always going to be zero, due to no self loops.

② Replace all the diagonal zeroes with degree of nodes.

③ Replace all the non-diagonal '1' with (-1).

④ Non diagonal zeroes leave as it is.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 0 | -1 | -1 |
| 2 | 0 | 2 | -1 | -1 |
| 3 | -1 | -1 | 3 | -1 |
| 4 | -1 | -1 | -1 | 3 |

→ Degree of nodes.

No. of Spanning Tree = Cofactor of any element.

Cofactor of (1,1) = $|[2(3\times3-1)-(-1)(-3-1)+(-1)(1+3)]|$

$= |[2(8)+1(-4)+(-4)]|$

$= |16-8|$

$= |8|$

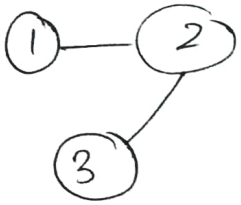
→ Always use |x| → to give +ve value

Remove the column and row corresponding to element you want to find the cofactor and find the determinant

In complete graph degree of each node is → $(n-1)$

$n$ = no. of nodes in graph.

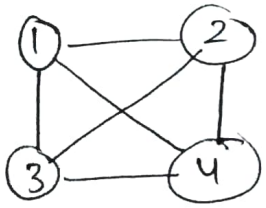
Ex -




$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \end{array}$$

Cofactor = $|2-1| = 1$ Ans

Ex →




$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array} = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Cofactor = $3(9-1)+1(-3-1)-1(1+3)$

$= 24-4-4$

$= 16$

and $n^{n-2} = 4^{4-2} = 4^2 = 16$

#

So, Kirchoff's Theorem is also applicable on Complete graph.