Analysis of Algorithm

1. Sorting. (8)
2. Searching. (6)
3. graphs. (4)
4. Type of Algorithms. (4)
5. Algorithm analysis.

1. Sorting

1) Bubble Sort :-

n = 5

a

0	5
1	2
2	-1
3	0
4	3

i	j	
3	0	
3	1	
3	2	pass (1)
3	3	
2	0	
2	1	(2)
2	2	
1	0	(3)
-1	1	
0	0	(4)

increasing order :-

```
for i -> n-2 down to 0
    for j -> 0 to i
        if (a[j] > a[j+1]):
            swap a[j], a[j+1]
```

This algorithm requires n-1 passes to sort an array of n element. The algorithm uses 2 counters i & j such that i varies from n-2 down to 0. For each value of i, j varies from 0 to i. For each value of j following condition is check.

if (a[j] > a[j+1])

if the if condition is true

then a[j] & a[j+1] are swaped

→ program for bubble sort.

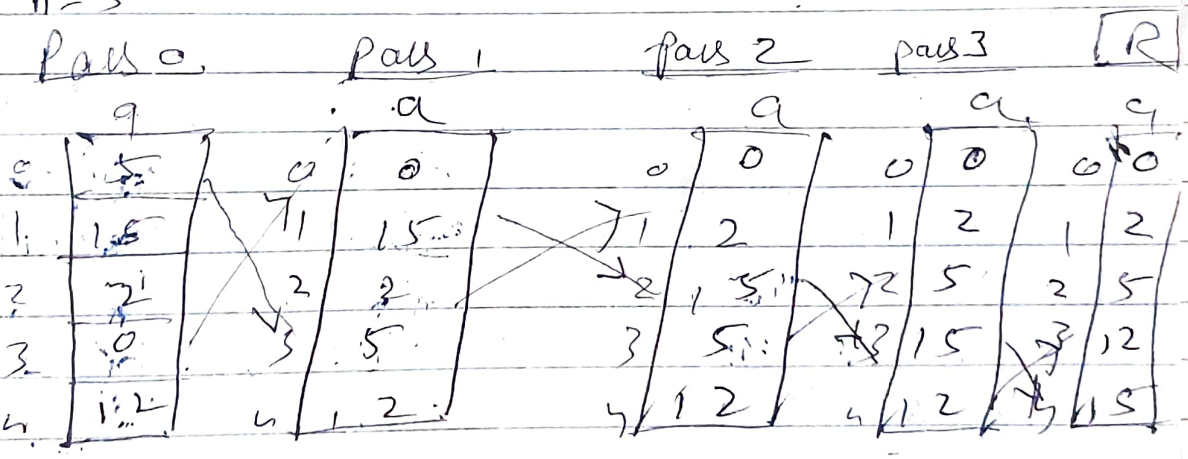
```
#include<stdio.h>
#include<conio.h>

void bubble(int a[], int n)
{
  int i, j, t;
  for(i=n-2; i>=0; i--)
    for(j=0; j<=i; j++)
      if(a[j] > a[j+1])
      {
        t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}

void main()
{ int a[100], n, i;
  printf("enter no of elements");
  scanf("%d", &n);
/* scan value of array */
for(i=0; i<=n-1; i++)
{ printf("enter element %d", i+1);
  scanf("%d", &a[i]);
}
bubble(a, n);
```

```
/* print sorted array */
for (i=0; i<=n-1; i++)
printf ("%d", a[i]);
}
```

Selection sort :-

This algorithm require n-1 passes to sort array of n elements in pass number i we select minimum element in a[i] to a[n-1]. This minimum value is exchange with a[i]

n = 5




```
Void selection (int a[], int n).
{
  int i, j, min, p, t;
for (i=0; i<=n-2; i++).
{ min = a[i]; p=i.
  for (j=i+1; j<=n-1; j++)
{ if (a[j] < min)
  { min = a[j];
```

```
}p=j;
t = a[i];
a[i] = a[p];
a[p] = t;
}. // end sort
} // end selection.
```

⇒ **insertion sort** :- This algorithm need $n-1$ passes to sort array of n values the algorithm sorts with assumption that array is partition into sorted & unsorted parts. only 1 element a[0] is in sorted part & remaining element are in unsorted part. In each pass 1st element of unsorted part is taken & it is inserted in the sorted part so that sorted part remains sorted.

n=5

Pass 1

	a	
0	3	sorted
1	5	unsorted
2	-1	
3	8	
4	2	

Pass 2

	a
0	3
1	5
2	-1
3	8
4	2

Pass 3

0	-1
1	3
2	5
3	8
4	2

Pass 4

-1
3
5
8
2

x = 3

j >= 1 x = -1 3 > -1

i >= 1 5 > -1

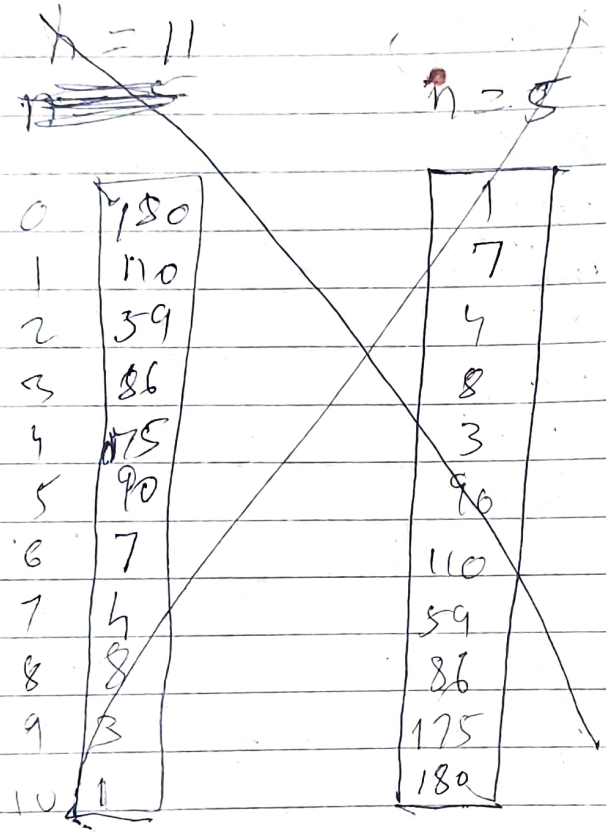
```
void insertion (int a[], int n)
{
  int i, j, x;
  for (i = 1; i <= n-1; i++)
  { x = a[i]; j = i;
  while (a[j-1] > x && j >= 0)
  {
    a[j] = a[j-1];
    j = j-1;
  } /* end while */
  a[j] = x;
  } // end for
  } // end insertion.
```

Insertion Sort gives good performance when array is sorted or partly sorted because in such case the inner while loop will not execute even once because all values are in right place

-> Shell Sort :- (Diminishing Increment Sort) :-

```
void shell (int a[], int n)
{ int i, j, x, h;
  h = n/2;
while (h != 0).
 { for (i = h; i <= n-1; i++)
  { j = i; x = a[i];
    while (a[j-h] > x && j >= h-1)
    { a[j] = a[j-h];
      j = j-h; }
```

```
a[j] = x;
}
h = h/2;
} // end while
} // Shell.
```



This Sorting technique compares element of array that are at some distance from each other. the distance is define with variable h Eg h=5 means these element will be compare which at a distance of 5 by doing this we would get 5 sorted array! 5 sorted array means all elements at a distance of 5 are in increasing order. Eg:- Show the step in which 5 array is sorted

using Shell sort.

n = 11

	a
0	180
1	110
2	59
3	86
4	175
5	90
6	7
7	4
8	8
9	3
10	1

1) let h = n/2 ∴ h = 5

0	180	→	0	90
5	90		5	180
1	110	→	1	7
6	7		6	110
2	59		2	4
7	4		7	59
3	86		3	8
8	8		8	86
4	175		4	3
9	3		9	175
0	90		0	1
5	180		5	90
10	1		10	180

1) we compare all elements at di's 0/5 like a0 compare a0. if a0 > a5 then pull down a0 by 5 places.

a5, a0 — a0 > a5 — push down

a0, a1 — a1 > a6 — " —

a7, a2 — a2 > a7 — " —

a8, a3 — a3 > a8 — " ←

a9, a4 — a4 > a9 — " —

a10, a5, a0 — a5 > a10 } push down.
a0 > a10 } push down.

h = 2

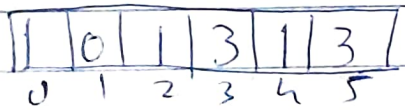
0	1
1	7
2	4
3	8
4	3
5	90
6	110
7	59
8	56
9	175
10	180

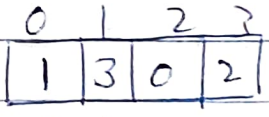
Now compare elements which are at distance = 2.
Eg compare a_2 with a_0
if $a_0 > a_2$ push down a_0 by 2 places.

3) when h = 1. All elements that are at distance = 1 are compared i.e insertion sort

(h-1 half left) done in next by's.

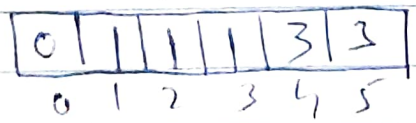
Counting Sort

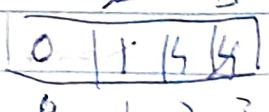




1 4 4 6 → shift

2^3 5 6





Ch (2) Searching

1) Searching

Linear search :-

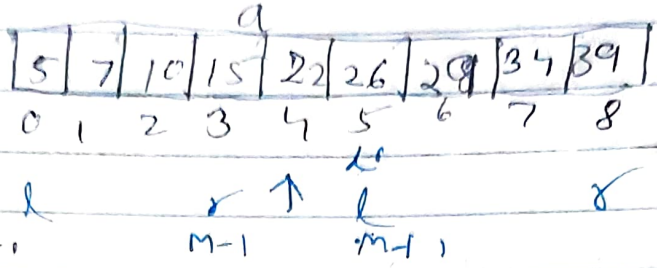

```
void linear (int a[], int n).
 {
  int i, x;
  printf ("enter value to search");
  scanf ("%d", &x);
  i=0.
  while (a[i]!=x && i<=n-1).
{ i++; }.
  if ( i >n-1)
  printf ("no such value \n");
  else
 }. printf ("element found at %d \n", i+1);
```

This searching technique will start searching an element x in the array from location 0. If location 0 does not store x then search proceeds to location 1. If location 1 is again not storing x then proceed to loc 2. This continues till x is found at some location or we have check all locations.

If all location are check then we declare x is absent. This search increases the location number by 1 every time hence called linear search or sequential search.

Searching :

Binary Search.



This Searching technique require array to be in sorted order. The array is partitioned into left & right parts by finding midpt. If value x is present at midpt then search terminates. But if value is not present at midpt; we either search left part or right part.

① x = 22

l	r	mid = (l+r)/2
0	8	4

Jerend.

② x = 7

l	r	Mid
0	8	4
0	3	1

Jerno

③ x = 10

l	r	Mid
0	8	4
0	3	1
2	3	2

found.

(4) $x = 39$

l	r	Mid
0	8	4
5	8	6
7	8	7
8	8	8

found.

(5) $x = 16$.

l	r	Mid
0	8	4
0	3	1
2	3	2
3	3	3
4	3	

Not found.

6) $x = 8$

l	r	mid
0	8	4
0	3	1
0	1	0
2	3	2
2	1	

Not found.

$x = 30$

l	r	mid
0	8	4
5	8	6
7	8	7
7	6	

Not found.

```
void binsearch (int a[], int n).
{
int l, r, mid, x;
printf ("enter value to search");
scanf ("%d", &x);
l=0; r=n-1; mid=(l+r)/2;
while (l<=r && a[mid] != x).
  { if (a[mid] < x).
    l= mid+1;
    else
    r = mid-1;
    mid = (l+r)/2. ;
  }.
if (l>r).
 printf ("no such data\n")
 else
 printf ("value found at place %d\n",
          mid+1);
}.
```

Hash Search:- Hashing is searching technique in which every key is mapped to some address of another array called hash array or hash table. The addr for a key is found by using some arithmetic function called hash function Eg hash func = Key % 10 so remainder of this will be used as address.

What is hash function : It is an arithmetic func which transforms a key into an address. The key is then stored at the addr found in the hash table. properties of hash function are

-> It should give min collision.
-> It should be easyly computable.

what is collision :- If 2 keys K_1 & K_2 are mapped to the same addr X & if K_1 is already stored at that addr then K_2 cannot be stored that is called collision.

Collision handling thenique :-

① Linear probing :- If 2 keys K_1 & K_2 collied at addrs X & if K_1 is already stored at addrs X then K_2 cannot be stored so new addrs for K_2 is searched ~~linar~~ linearly (X+1, X+2, X+3 ....) till we get a vacant plate where K_2 can be stored. This method of handling collision is called linear probing.

Original array a

55	72	78	65	96	77	85	109	98
0	1	2	3	4	5	6	7	8

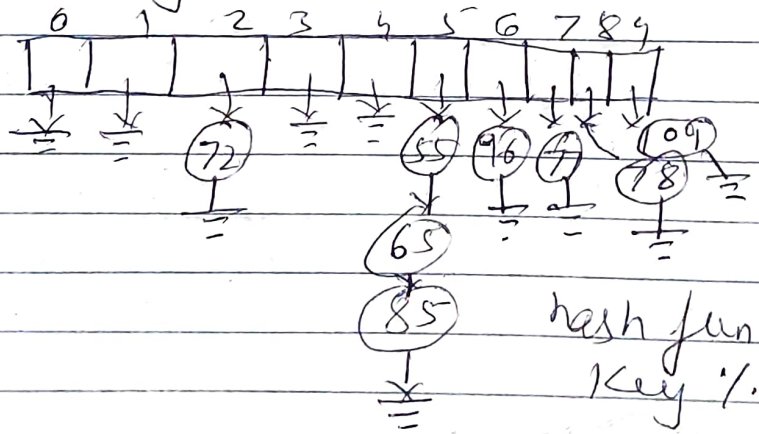
		72			55	65	96	78	77	85	109	98
0	1	2	3	4	5	6	7	8	9	10	11	12

85 65 96 77 98 109

hash func = Key % 10

If linear probing technique is used for handling collision then the hash array size must be larger than the size of original array this is because some empty slots (vacant positions) are needed in the hash array to confirm that element is absent.

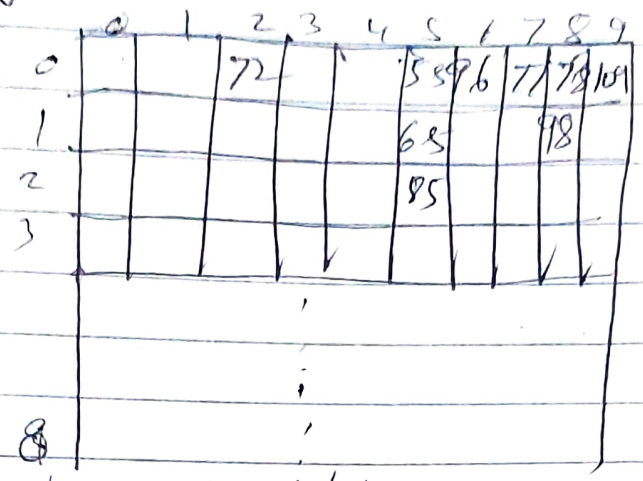
(2) Direct chaining:- ~~Di~~ In this method the collided keys are stored in the form of link list. This link list can be later searched to see wheather an element is present or absent. Size of hash array is decieded by the hash func
Eg:- If key % 10 is hash function then min & max addr is 0 & 9 so hash array should have 0 to 9 location.



hash func = key % 10.

(3) Using Buckets with more slots:- In this a double dimension array or matrix is use for representing hash array. No of columns of mat are decided by hash function. So if hash func is key % 10 then column will be from 0 to 9.
No of rows should be at least =

No of elements



hash func = key % 10

-> hashing methods :-

① Hashing by division :- This method divide every key by some const the remainder obtained acts as an addrs Eg : key % m will give addrs in the range of 0 to m-1 hence hash table size should be atleast m.

② mid-Square hashing :- Every key is Squared & some fix no of digits are calculated from mid pt of square these digits will act as addrs. Eg :- let 55 be key whose square is 3025 obtain 2 digits from mid pt i.e 02 hence 02 is addr of 55. If 2 digit are picked from mid-pt then addr is in the range 0-99 hence hash array should have at least 100 elements

③ folding :- This method is suitable for keys of larger length every key

is folded such that each fold contains equal no of digits. all folds are added & carry is ignored from sum this sum is use as addr. Eg:-
if key is 123456789 then folding

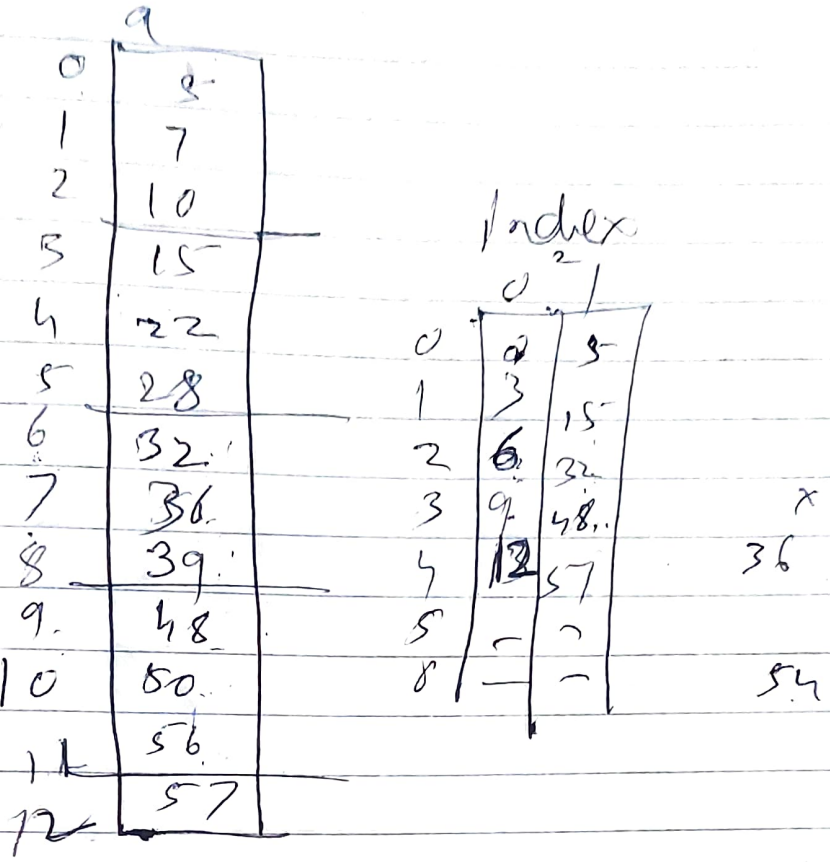
12
34
56
78
09

189
↳ Cary ignore

so addr = 89.

Density of hashing:- $\frac{\text{No of collision}}{\text{No of elements}}$

Index Sequential Search:- This searching the needs the array to be sorted in increasing order the array is partition such that every portion contain equal no of elements one extra array called index array is use by index search this array has exactly 2 columns & several rows column 0 of index stores position at which portion begins column 1 of index stores value at that position Eg:-



→ Steps for searching 1 element like x=36 are as follows :-

- use column 1 of the index & search for 1st value which is larger than 36. (it is 48 in row 3).
- Go to the previous row (it is row 2
- obtain addrs from column 0 of row 2. (addr is 6.).
- Now start searching array a from addr 6. & stop at 1st value which is larger than 36. (it is 39 at position 8).
- Go to previous element ~~that is~~ i.e 7 & Now check it with 36. (X).

```
#include<stdio.h>
int a[100], n, index[35][2], row;
void create index()
{
 int i;
 row = -1;
 for(i=0; i<=n-1; i=i+3)
 { row++;
   index[row][0] = i;
   index[row][1] = a[i];
 }
 } /* end createindex */

void indexSearch()
 {
  int x, i, addr;
 printf("Enter value to Search");
 scanf("%d", &x);

 if (x < a[0] || x > a[n-1])
 printf("No Such value \n");
 else
 {
  i=0;
 while(index[i][1] <= x &&
       i <= row)
& i++;
  i--;
 addr = index[i][0];
 while(a[addr] <= x && addr <=
       n-1);
 addr++;
```

```
addr = -1;
if (a[addr] == x)
printf ("value found \n");
else
printf ("not found \n");
}
}

void main()
{ int i;
printf ("\n enter n");
scanf ("%d", &n);
/* Is can value */

for (i=0; i<=n-1; i++)
{
printf ("enter element");
scanf ("%d", &a[i]);
}
bubble (a, n);
create index ();
indexsearch();
}
```

B. Tree :-

B tree is called as balanced multiway tree in which every node can store n keys & will have n+1 pointers.

- Max. no of keys that a node can store is divided by order of tree. Order is define as no of subtrees

Or children that a node can have.
Eg: If order of B tree is 5 then every node can have 5 child nodes & can store max 4 key.

All leaf node are always at same level ∴ this tree is called balanced tree.

→ Let us Eg of B tree of order 5

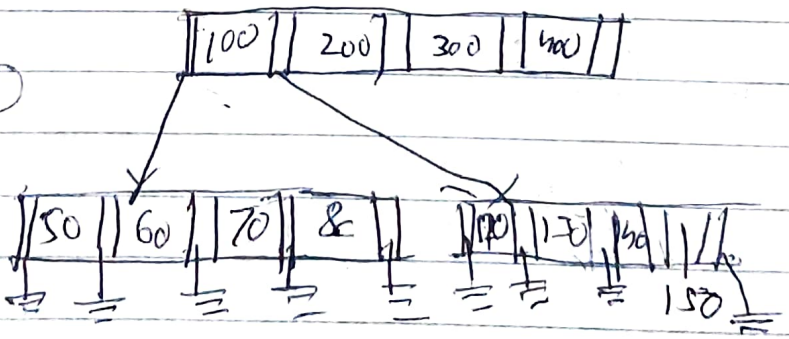
As shown in tree below root node is storing keys 100, 200, 300, 400

All key is store to left of 100 are less than 100 (50, 60, 70, 80).

All key store to right of 100 but left of 200 are greater than 100 & less than 200. (120, 130, 140, 150).

Binary tree

B tree.
multiway tree.



Ceiling symbol

Rule of B-tree :

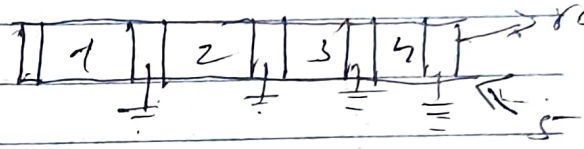
$\lceil order/2 \rceil - 1$ key.

- Every node must contain atleast ~~order~~ divided by 2 keys. Root node is exception to this rule.

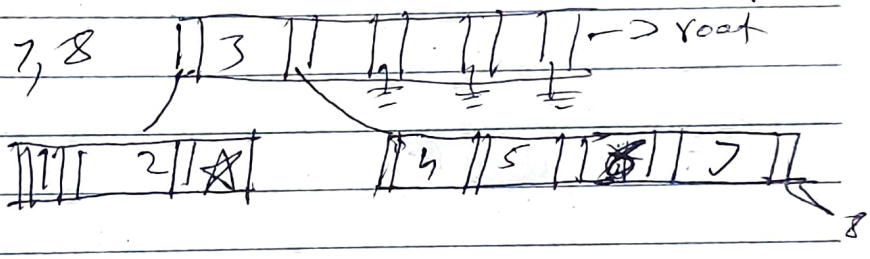
-> Construction of B-tree :-

- Construct a B tree of order 5 using following data : 1 - 17 :

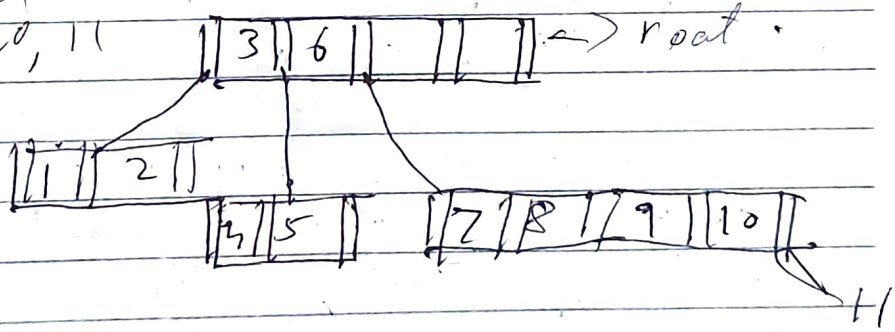
(1) add : 1, 2, 3, 4, 5



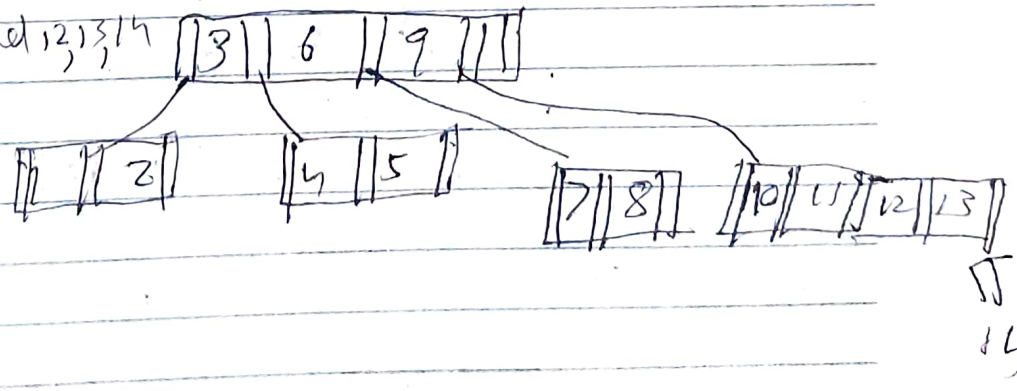
(2) add 6, 7, 8



(3) add 9, 10, 11

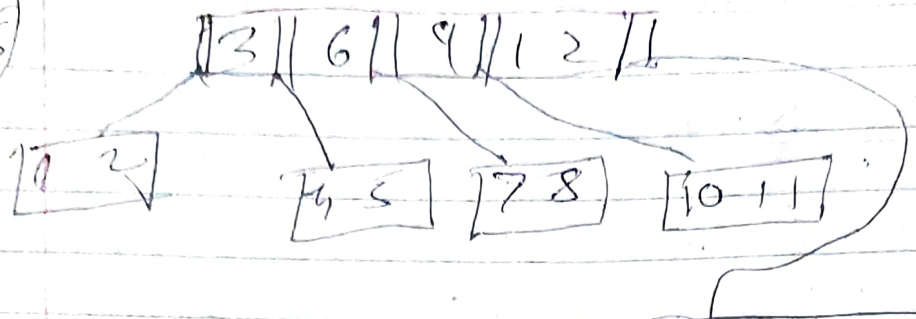


(4) add 12, 13, 14



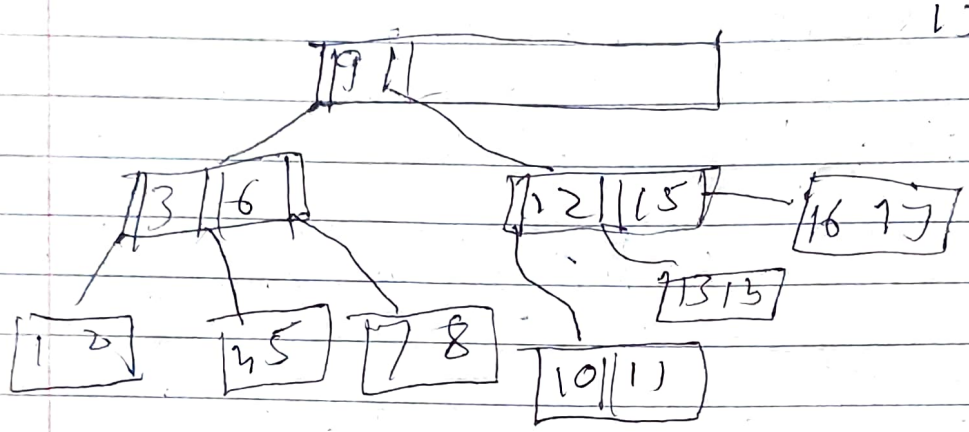
add 15, 16, 17

(5)



17

(6)



Split & propagate :-

-> while inserting a new key in a node which is already full split & propagate process occur

=> If a key can't be inserted in a node because node is full then node is split such that 2 new nodes are created each containing order by 2 keys.

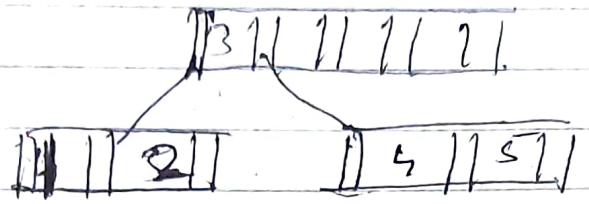
=> The median of node is propagated to the parent node.

Eg: Consider a B tree of order 5 which contains following key.

1 2 3 9 12

while inserting a new key such as 5 in the above node split occurs creating 2 sub nodes the median 3 is propagated to become new root

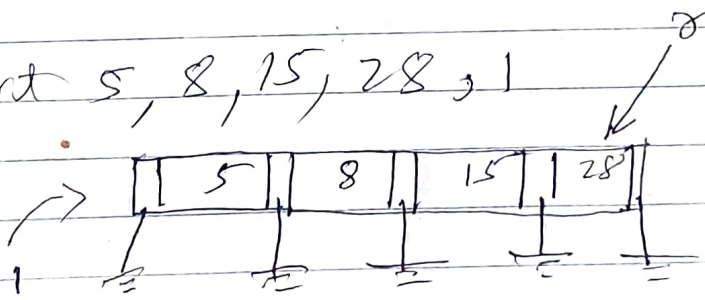
∴ resultant tree is



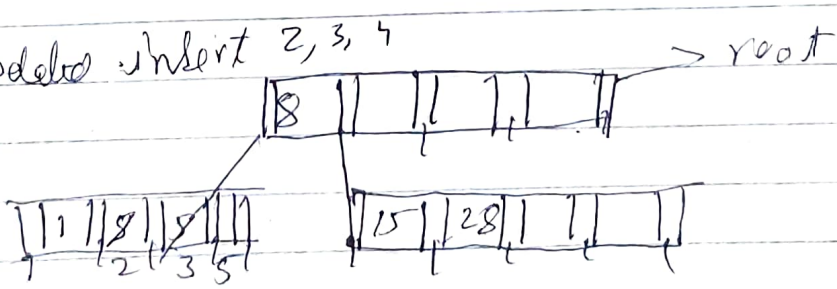
Q. Construct a B tree of order 5 by using following data.

Data → 5, 8, 15, 28
1 2 3 4
30 36 42 48.

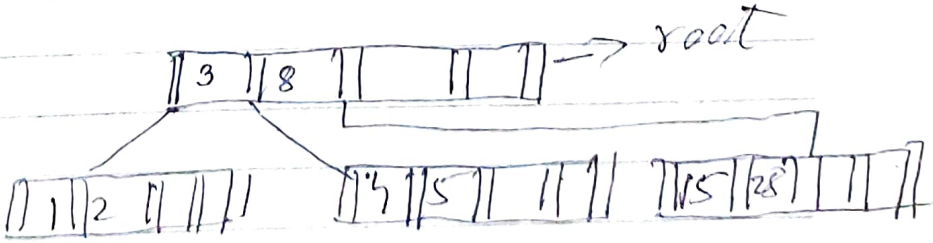
1) insert 5, 8, 15, 28,



2) updated insert 2, 3, 4

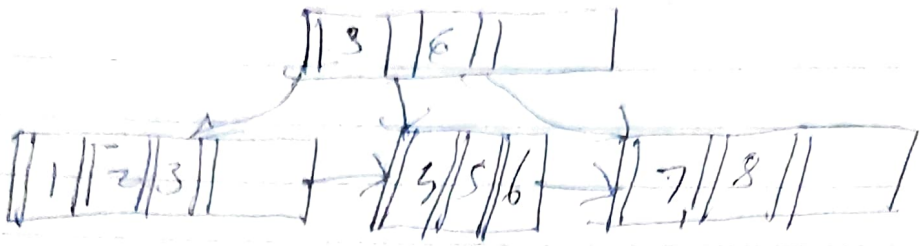


3)



⇒ B+ tree :

B Tree	B+ tree
→ Every key is stored either in leaf node or non leaf node but a key is never repeated	→ Every key which is stored in non leaf node is stored again in one of the leaf nodes
→ All leaf nodes are at same level but not sequencialy connected	→ All leaf node are at same level & are connected ~~by ref~~ sequencialy using additional pointer.
→ Records associated with keys are stored inside the b tree. This makes B tree bulky in size. So B tree cannot be stored in ram & cannot be processed.	→ Records associated with keys are stored outside the B+ tree. Address of these records are stored by the pointers in the leaf node. So B+ tree is small in size as compare to B tree. Due to this B+ tree can be taken into Ram & can be processed.
→ B tree can be traversed only randomly.	→ B+ tree can be traversed both randomly & sequencially



AVL tree

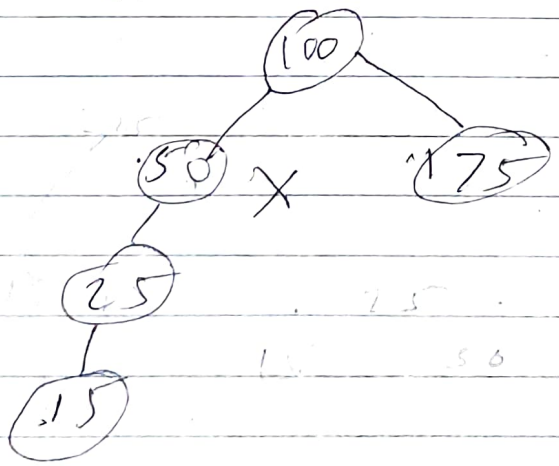
Adelson Velski & Landis tree.

-> AVL is called balanced binary search tree in which every node should satisfy the following criteria.

$$|h_l - h_r| \leq 1.$$

where h_l is height of left sub tree & h_r is height of right sub tree.

-> Consider following binary search tree.

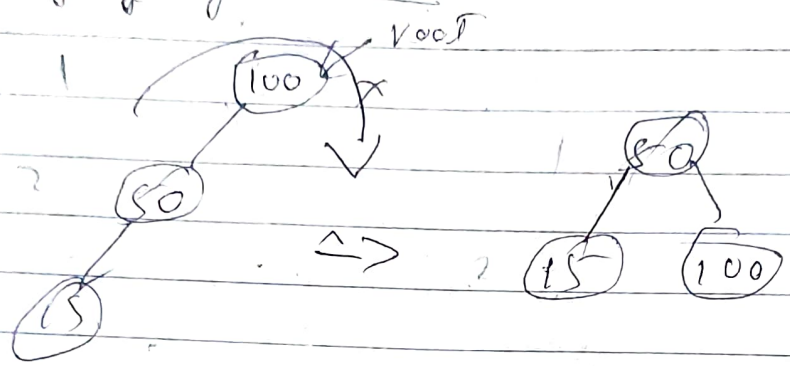


here tree is not balance & '50' is not balance.

There are 4 cases in which binary search tree will be unbalance, AVL have given rotation algorithm which help us in balancing the tree.

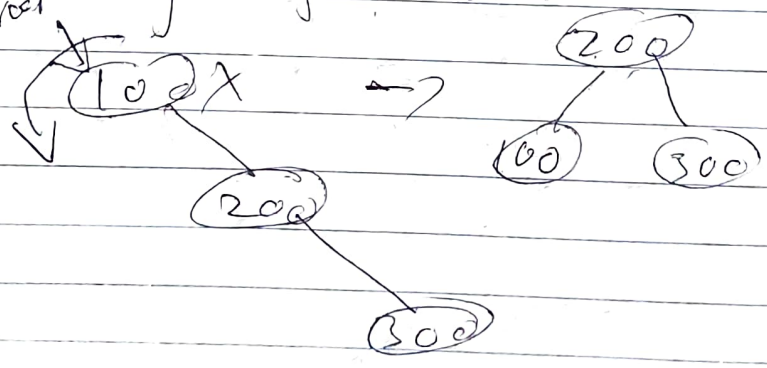
Case 1:

Left of left case :-



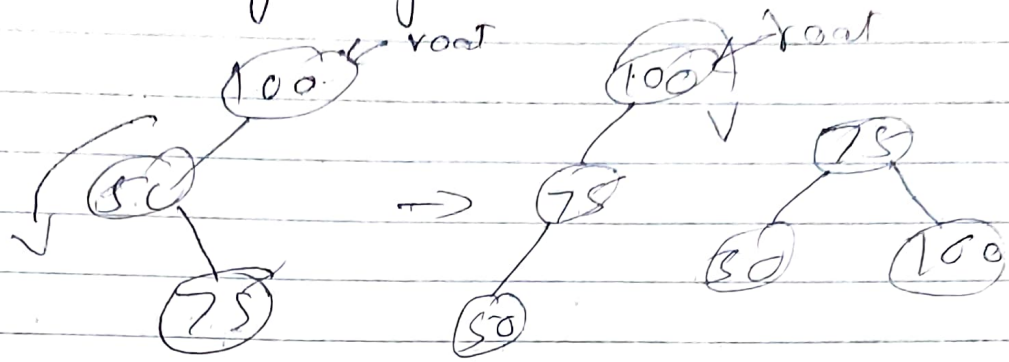
100 is unbalanced due to left-left so rotate 100 to right of 50
This is right rotation.

Case II right right case.



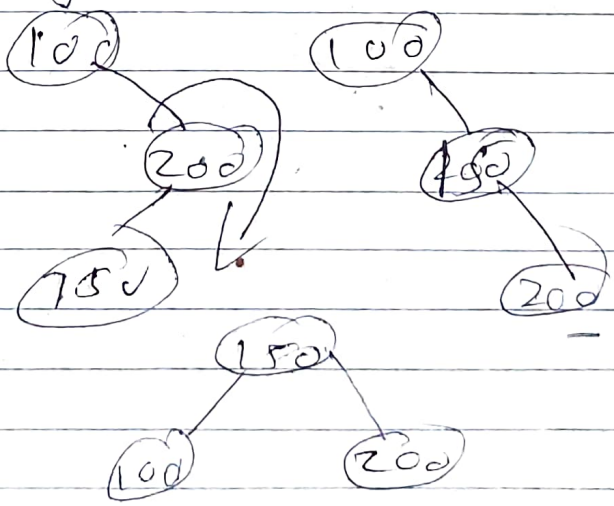
100 is unbalanced due right right so rotate 100 to left of 200. This is called left rotation.

Case 3:- left right case



100 is not balanced due to left right so rotate left right child of 100 i.e 50 to left of 75. This give left right case

case 4:- right-left case.



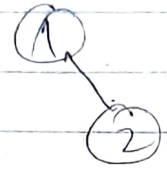
100 is not balance due to right left so rotate right left child of 100 i.e 200 so to right of 150. This give right left case.

Show step to create AVL tree for following data

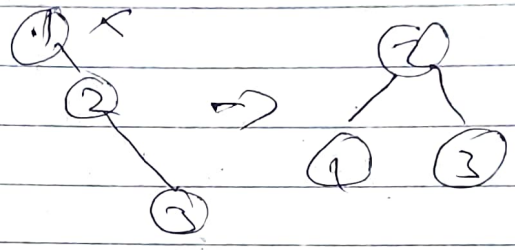
data → 1 to 7

1) add 1

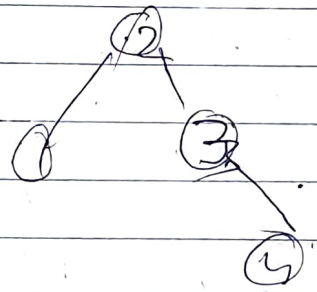
2) add 2



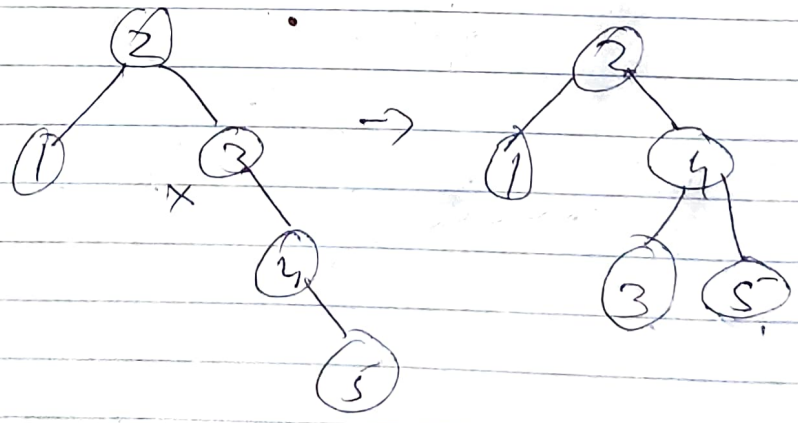
3) add 3



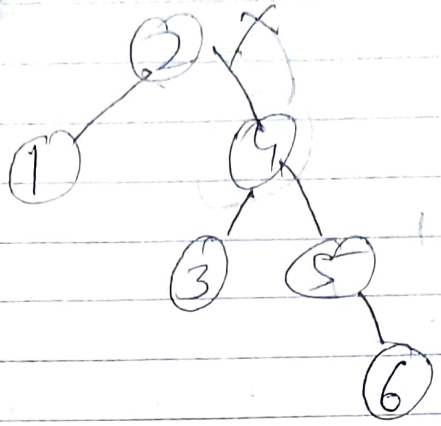
4) add 4

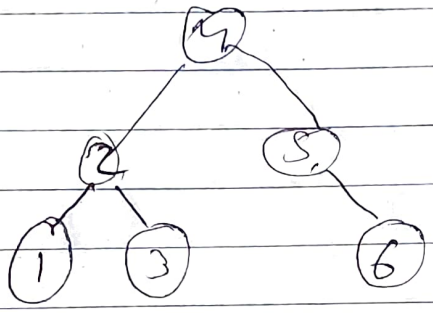


5) add 5

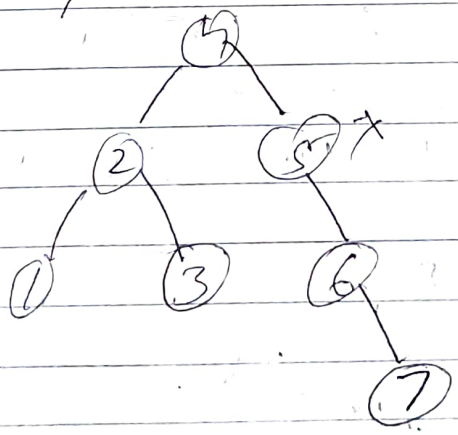


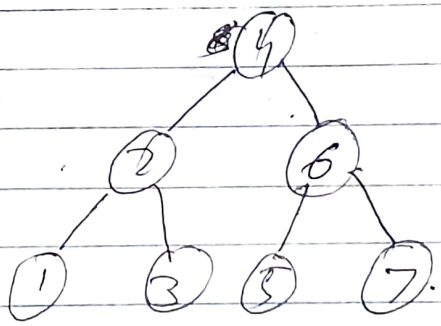
6) add 6





7) add 7.



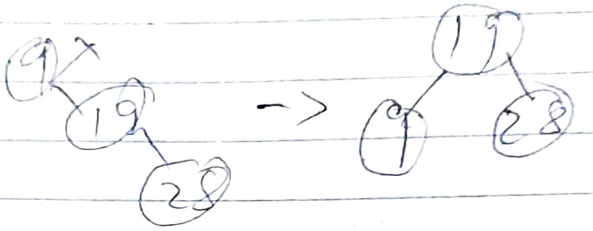


☆ Construct AVL tree by using following data. 9, 19, 8, 4, 3, 2, 6, 99, 8,

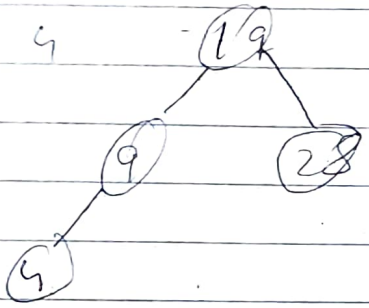
1) add 9

2) add 19

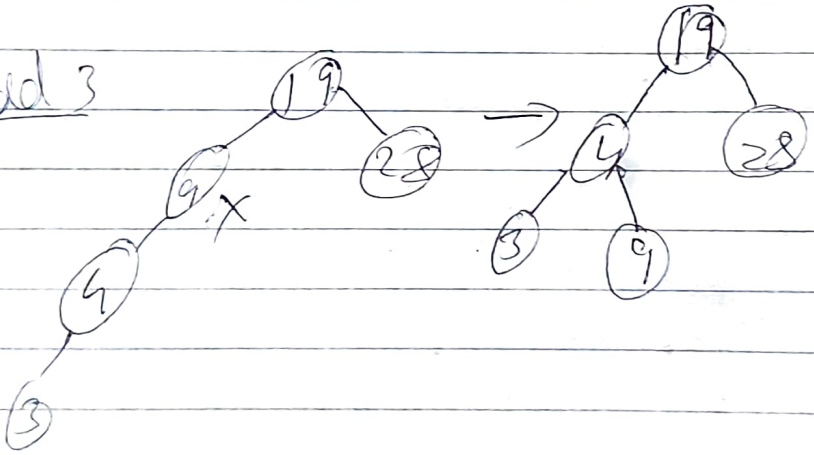
3) add 28



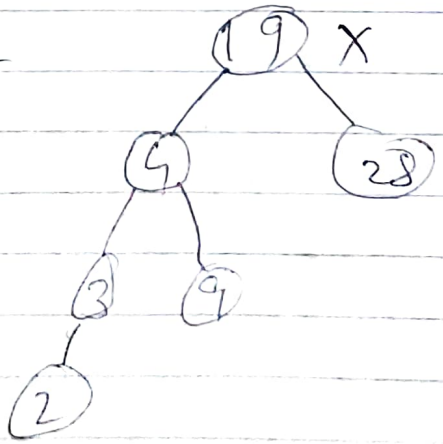
4) add 4

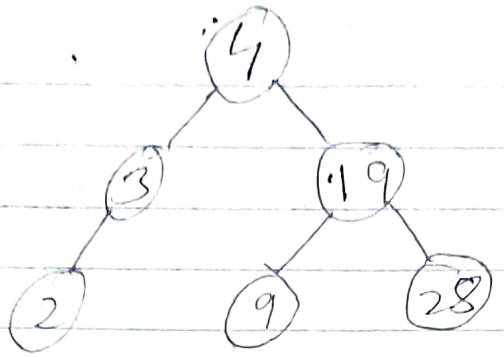


5) add 3

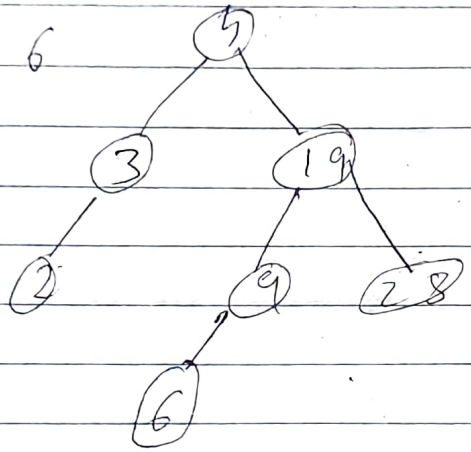


6) add 2

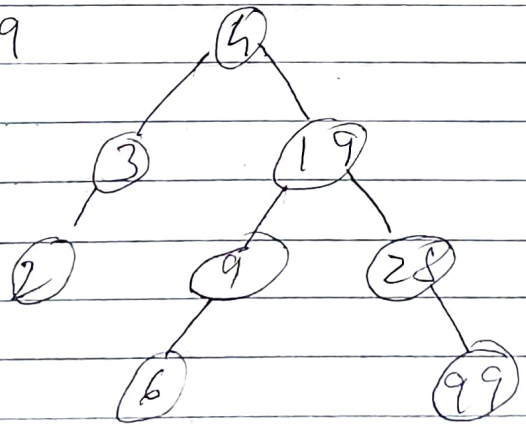




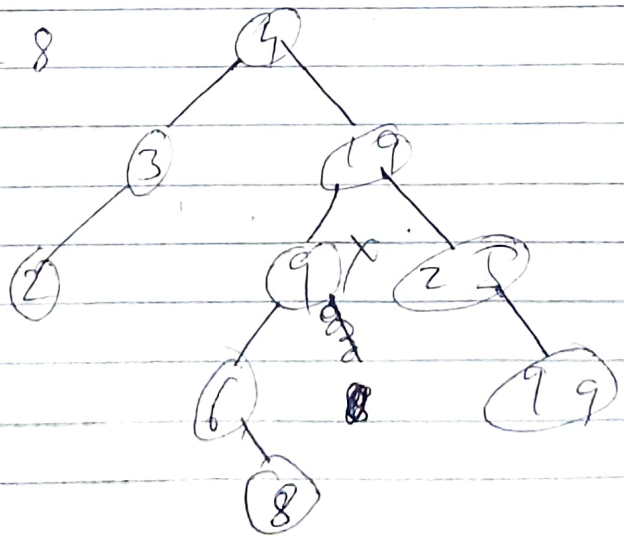
7> add 6

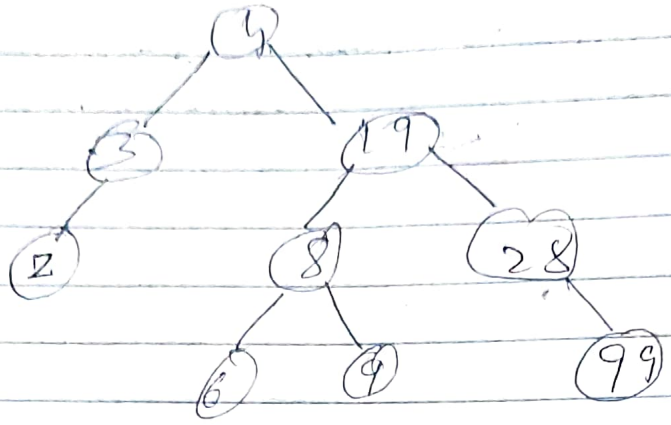


8> add 99

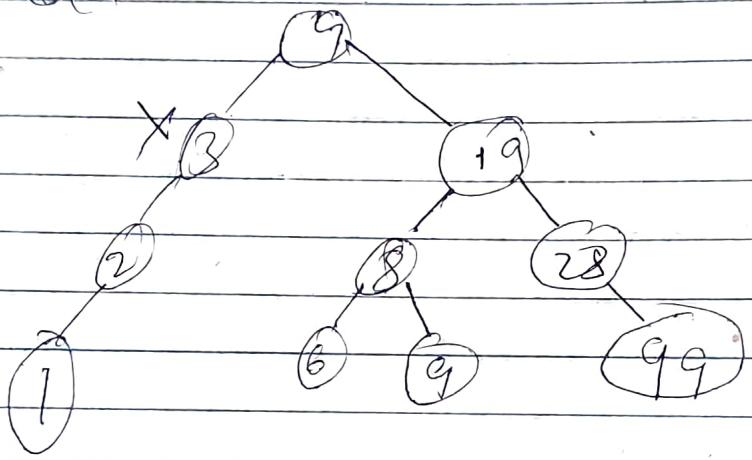


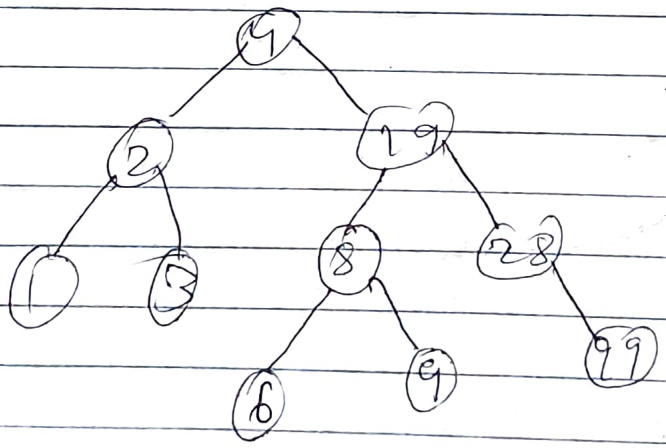
9> add 8





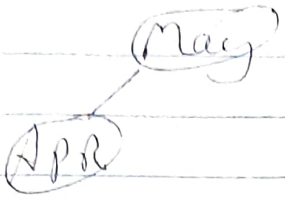
10> add 1

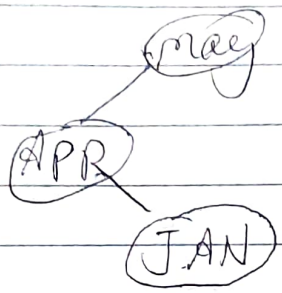


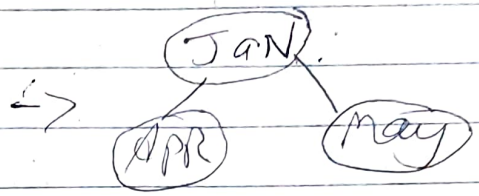


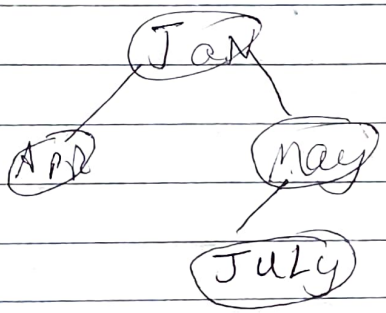
⁂ Construct AVL tree using full data.
Data :- May APR JAN July Dec June FEB.

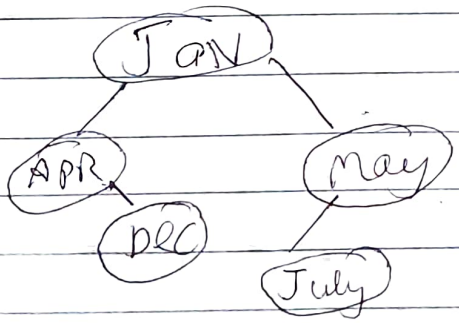
1) May (root)

2)


3)




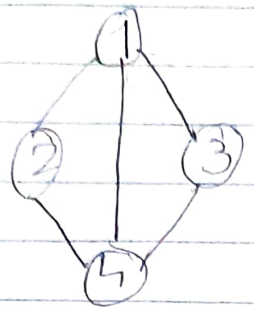
4)


5)


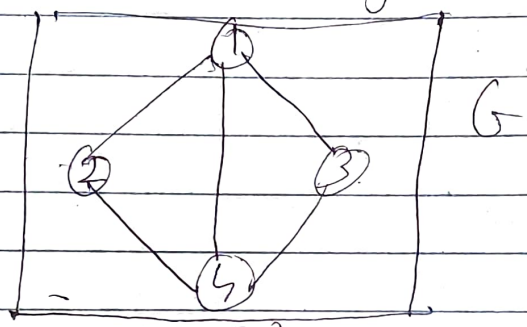
6) June

Dsa1 alg 0.

Graphs.



Graph is a data structure which is collection of vertices & edges every edge connects 2 vertices eg:-



$G = (V, E)$

$V = \{1, 2, 3, 4\}$.

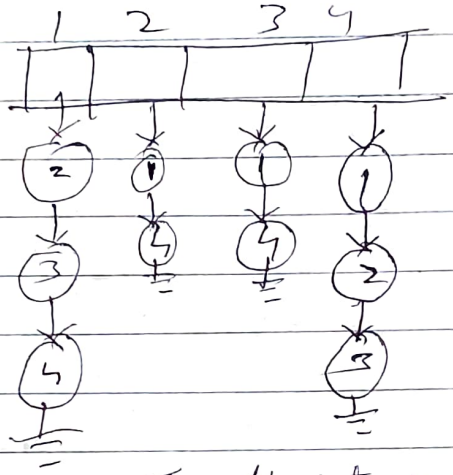
$E = \{(1,2), (1,3), (1,4), (2,4), (3,4)\}$

Representation of graph in computer memory

1) Adjacency matrix:- This method uses 2-D array having (V) Row & (V) Columns (V = no of vertices). (i, j) entry of matrix is set as 1, if there is edge from vertex i to vertex j otherwise entry (i, j) is set as 0.

	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

2> Adjacency list: This method declares array of pointers.
For a 'V' vertex graph the method will use 'V' pointers. Every pointer will maintain a link list representing adjacent vertices.



The first pointer pts to a link list which has vertices 2, 3, 4. This means that adjacent vertices of 1 are 2, 3, 4.

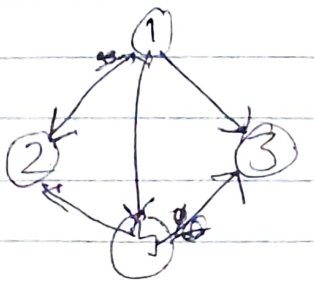
};

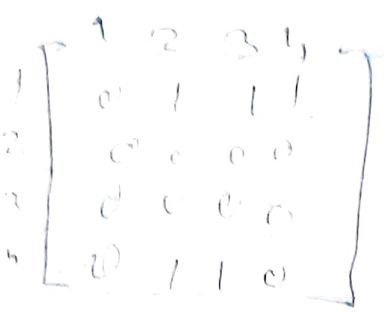
Types of graph.

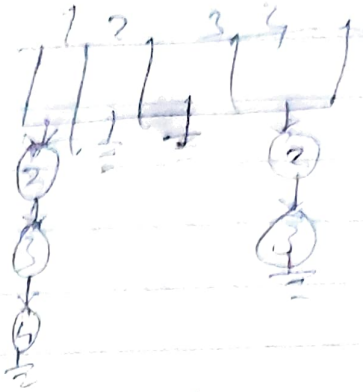
1> undirected graph: - All edges do not have direction

2> Directed graph (digraph) :- All edges must have direction

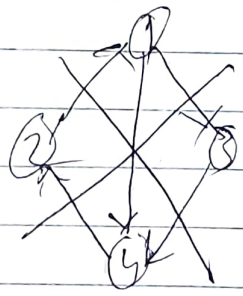
Eg

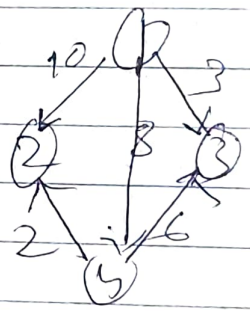


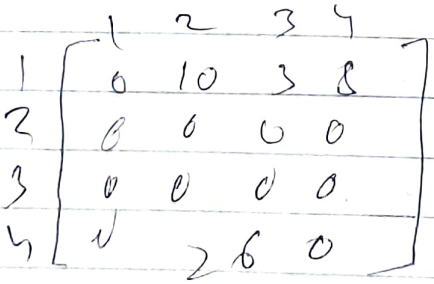


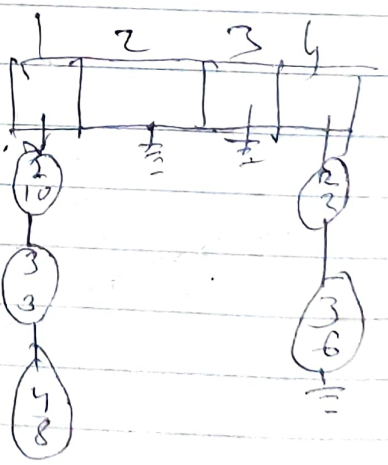


3) Weighted directed graph :- All edges have direction & every edge is associated with some weight.

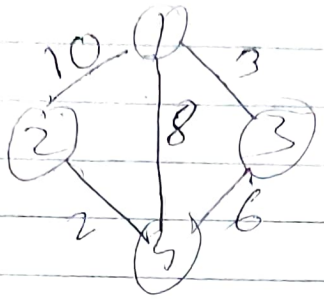




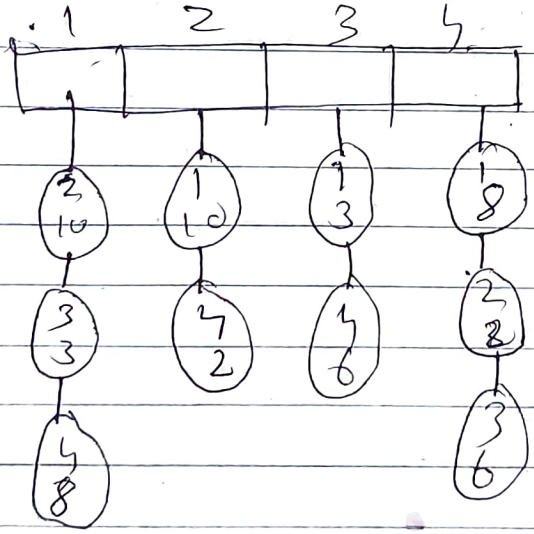




4) weighted undirected graph

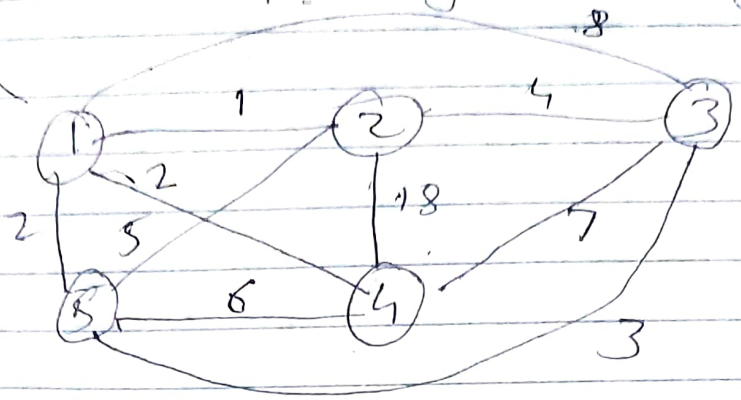


	1	2	3	4
1	0	10	3	8
2	10	0	0	2
3	3	0	0	6
4	8	2	6	0



All edge don't have directions & they have weights associated with them.

☆ Minimum cost Spanning tree of a graph



Given a weighted undirected graph of 'V' vertices & 'E' edges. This graph may contain some cycles. we want to create a tree from this graph such that tree will contain all 'V' vertices. But it will not have any cycles.

The total span of the tree (sum of all weight) should be minimum such a tree is called minimum cost spanning tree.

☆ Kruskal's algorithm :-

(comments) :- G is the graph which is weighted undirected the graph has 'V' vertices & E edges.

'T' is an empty spanning tree

'X' is a set of all edges in increasing order of weight

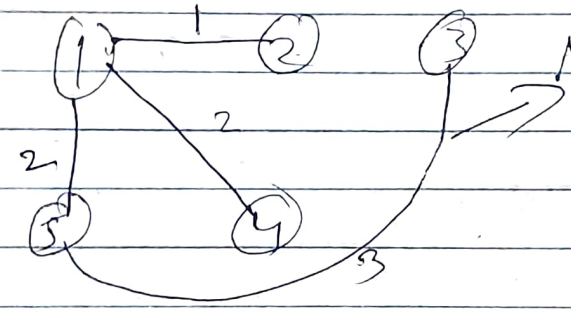
1. Repeat Step 2 While number of edges in T ≠
2. Let 'u' be next edge from set 'X'

Add edge 'e' to Tree T if Edge w (creates cycle in T) discard edge w from tree T.

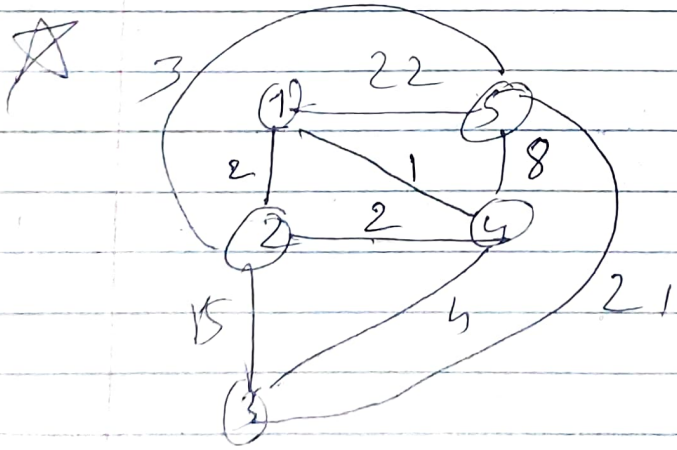
3. finish.

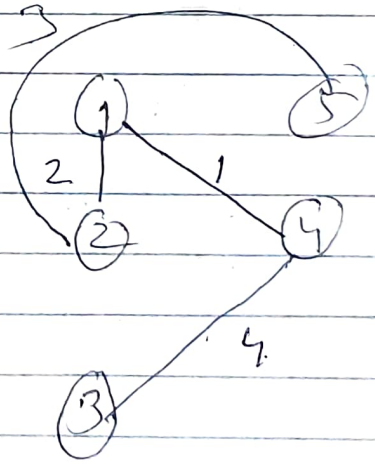
Kruskal's method

X = { (1, 2, 1), (1, 5, 2), (1, 4, 2), (5, 3, 3), (2, 3, 4), (5, 2, 5), (5, 4, 6), (4, 3, 7), (1, 3, 8), (2, 5, 8) }.



Min cost = 8.



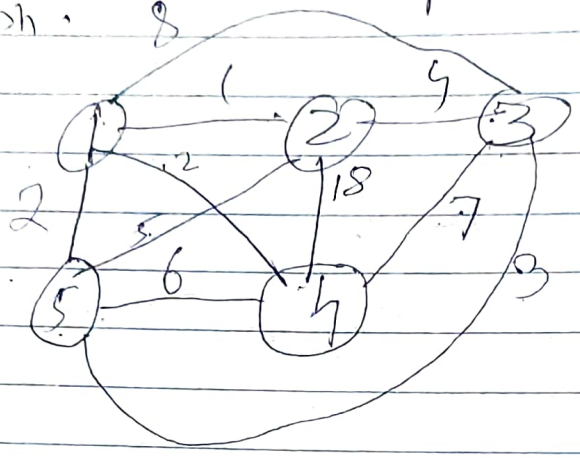


Min Cost = 10

Minimum cost spanning tree of a graph

prim's method

find out min cost spanning tree for following graph.



1) Vertex 1

	1	2	3	4	5
d	∞	1	8	2	∞
p	0	1	1	1	1
visited	1	1	0	0	0

2) Vertex 2

	1	2	3	4	5
d	0	1	8	2	2
p	0	1	1	1	1
visited	1	1	0	1	1

3) Vertex 4

	1	2	3	4	5
d	0	1	4	2	2
p	0	1	4	1	1

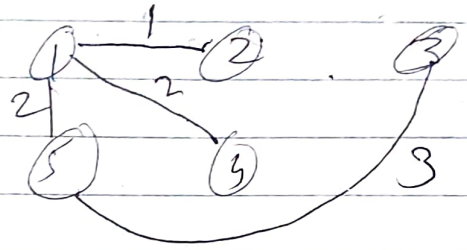
4) Vertex 3

1	2	3	4	5
0	1	1	2	2

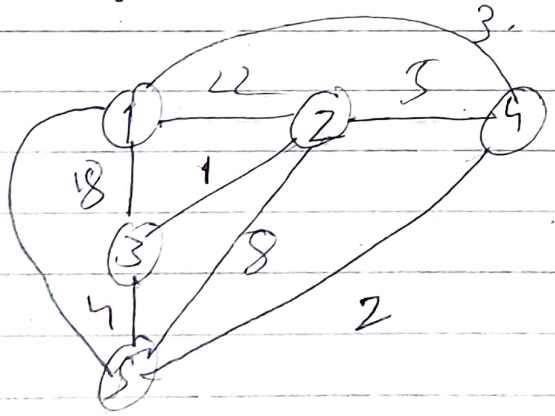
1	2	3	4	5
0	1	1	1	1

5) No pt to take Vertex 3.

Ans Min cost = $\Sigma d = 8$



find out min cost spanning tree for full graph using prim's method.



q

2 visited

1	2	3	4	5
∅	∅	0	0	0

1) vertex 1

1	2	3	4	5
∅0	∅22	∅18	∅3	∅9

p

1	2	3	4	5
0	∅1	∅1	∅1	∅

1

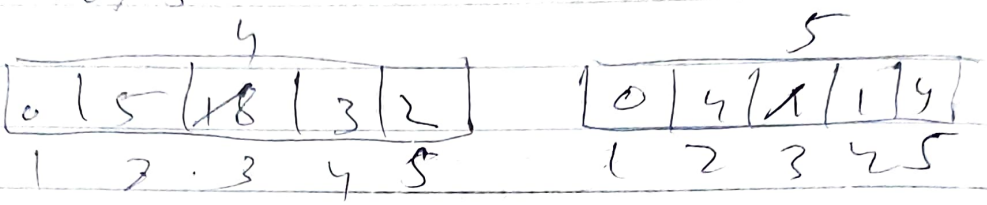
2) vertex 2

1	2	3	4	5
0	22	∅8	∅3	∅

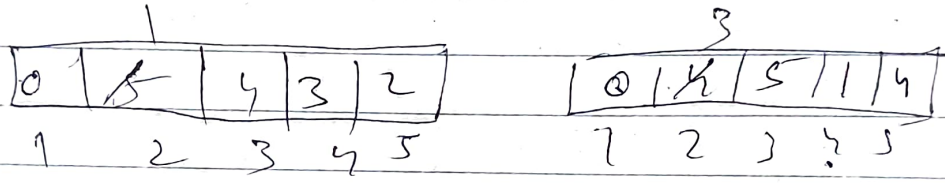
1	2	3	4	5
0	1	∅	1	∅

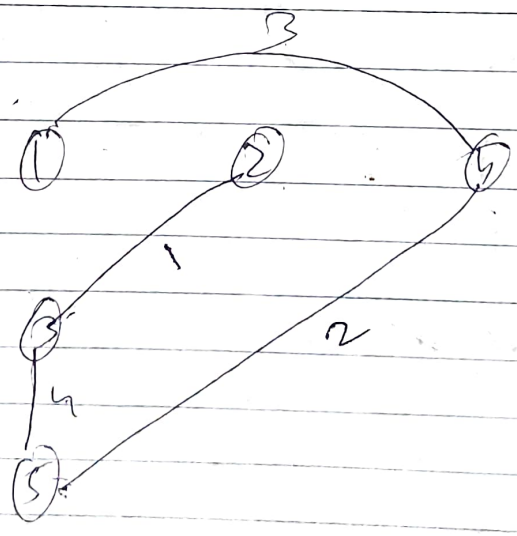
7) Vertex 4

8) Vertex 5

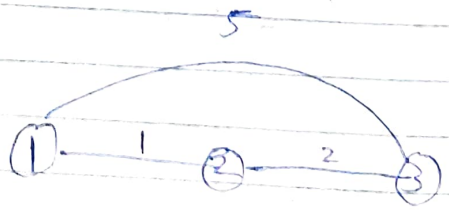


9) Vertex 3



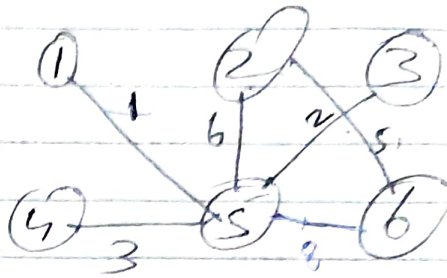


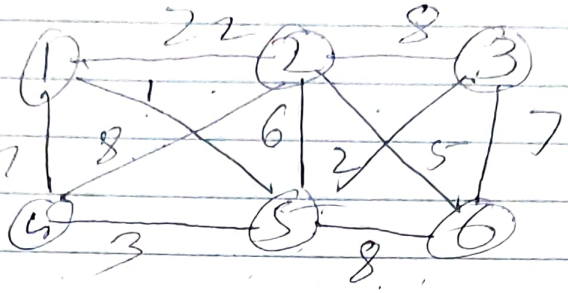
Min cost = 10



A Shortest distance problem

Dijkstra's method





~~To~~ find shortest dis from 1 to all

① vertex 1

d

~~∞~~	22	4	1		
~~∞~~	∞	∞	∞	∞	∞
1	2	3	4	5	6

P

	1	1			
0	1	1	1	~~∞~~	0
1	2	3	4	5	6

	1	2	3	4	5	6
1	0	22	0	4	1	0
2	22	0	8	8	6	5
3	0	8	0	0	2	7
4	4	8	0	0	3	0
5	1	6	2	3	0	8
6	0	5	7	0	8	0

1 visited

1	1				
~~∞~~	0	0	0	0	0
1	2	3	4	5	6

dc = d[1]

dc = 0

② Vertex 5

d

7	3			9	
0	~~22~~	~~∞~~	4	1	~~∞~~
1	2	3	4	5	6

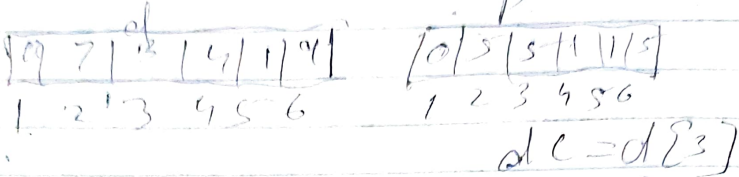
P

5	5			5	
0	x	0	1	1	~~∞~~
1	2	3	4	5	6

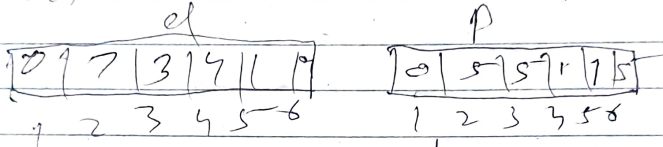
dc = d[5]

= 1

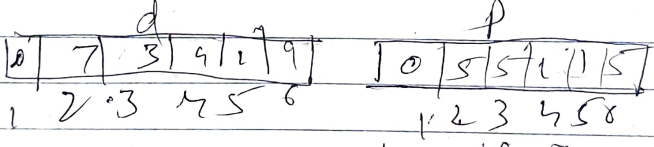
③ vertex 3



④ Vertex 4



5) vertex 2



6 ×
6 → 5
5 → 1
1

Shortest dist from 1

from	to	dist	array
1	2	7	d[2]
1	3	3	d[3]
1	4	4	d[4]
1	5	1	d[5]
1	6	9	d[6]

Each vertex	to	Path
1	2	1 -> 5 -> 2
1	3	1 -> 5 -> 3
1	4	1 -> 4
1	5	1 -> 5
1	6	1 -> 5 -> 6

Prim's algorithm to find min cost spanning tree :-

1) Given a weighted undirected graph of 'V' vertices & 'E' edges. This graph is represented in the form of adjacency matrix G.
2) The algo will use 3 arrays named D, P & visited. Each having V elements.
3) D array is initialize to infinity, P & visited array are initialize to 0.
4) a) let current = 1
   b) d[current] = 0.
   c) visited[current] = 1.
   d) let c = 1
   e).
5) repeat step 6 to 9 while c ≠ V
6) // process vertex current.
   for i = 1 to V
   {
   if (G[current][i] ≠ 0 & visited[i] ≠ 1)
   {
   if (G[current][i] < d[i])
   {
   d[i] = G[current][i]

P[i] = current.
}}.
}.

7. let current = vertex and shortest distance among unproces vertices.
8. visited [current] = 1
9. c = c + 1.
10. Ru, call = Ed.
11. finished.

☆ Dijkstra's algo to find shortest dist from source to dest.

1) 1 to 3 as same as prim's.

4) a) let current = source.
d[current] = 0.
visited [current] = 1

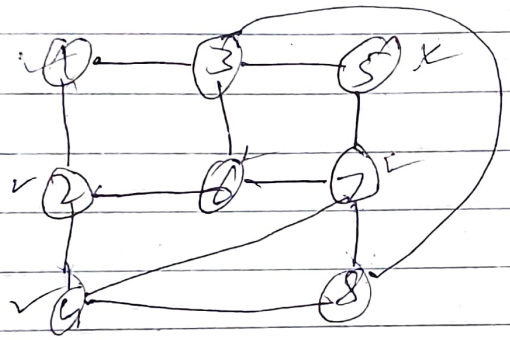
5) repeat step 6 to 8 while current ≠ ~~source~~ dest.

6) // process vertex current.
dc = d[current].
for i → 1 to V
{ if (G[current][i] ≠ 0 & visited[i] ≠ 1)
{ if (dc[current][i] + dc < d[i])
{ d[i] = G[current][i] + dc
P[i] = current
}}.
}.

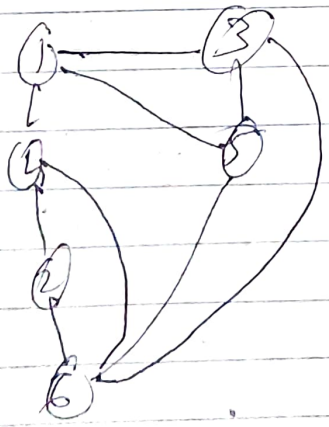
7) If current = index of shortest dist among unprocessed vertex.
8) visited [current] = 1.

9) for adj = d [dest]
10) End for.

☆ Depth first traversal of a graph. (DFS).



1 2 4 7 5 3 6 8



1 3 5 6 2 4.

```
algo dfs (k : vertex)
{
  visited[k] = 1  // visited value k.
  print k.
// process vertex k (check adj : vertex)
for i -> 1 to v
{
if (a[k][i] != 0 && visited[i] != 1)
  call dfs(k).
  }
  },
```

-> Comments

- given a graph of v vertices & E Edges this graph is represented by its adjacency mat a. the algo uses an array visited which has v elements all element of visited array are initially 0.

1) Develope an Algo which will check wheather a graph is connected or not

Ans Graph is said to be connected if every vertex is reachable from every other vertex, in other words graph is connected if it has only 1 component.

Solution -> DFS algorithm is same as above but calling func of dfs is

algorithm caller.

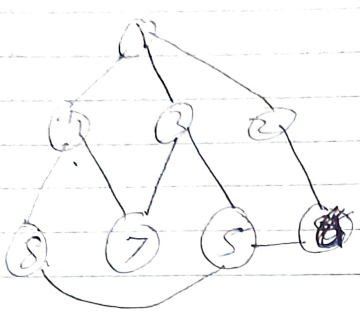
```
{
c=0
for i -> 1 to V
{ if (visited[i] == 0)
{ call dfs(i)
  c = c++
} }
if (c == 1)
display graph is connected
else
disp graph is not connected.
}.
```

2) ~~Ans~~ Develope an algo which will count total no of components in the graph.

Ans Algorithm caller.

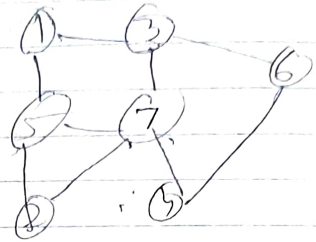
```
{
c=0.
:
: (same as above).
:
c = c++
}}
display No of component = c.
```

☆ Breadth first search traversal of a graph



1 2 3 4 6 5 7 8

~~1 4 3 2 7 8 5 6~~



1 3 5 6 7 2 4

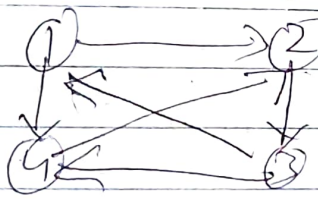
```
algorithm bfs (k : vertex).
{
  insert (q, k)
  while ( not empty (q))
  {
    k = delete (q)
    visited [k] = 1
    print k
    // check adj vertices of k.
    for i -> 1 to V
    {
      if [a[k][i] ≠ 0 & visited[i] ≠ 1
          i not in q).
      { insert (q, i)
      }
    } // end for
  } // end while
} // end algo
```

Comment

Given a graph of v vertices & edges this graph is stored in the form of adjacency mat G. The algorithm uses an array visited which has v elements this array is initialize to 0. The algo uses a data structure q which will store all those vertices which are not yet visited.

☆ path matrix or transitive closure of a graph.

Given a graph of v vertices & E edges this graph is represented by adjacency matrix G.



	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	1	0	0	1
4	0	1	0	0

G gives us all paths of length 1; Because it gives information of adjacency.

G^2 gives all path of length 2.

G^2	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	0	2	0	1
4	0	0	1	0

where $G^2[1][3] = 1$ mean there is 1 path of length 2 from 1 to 3.

18

Similarly G^2 will give all paths of length = 2.

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

finally path matrix P can be found as
$P = G + G^2 + G^3 + \ldots\ldots G^V$

'P' is called path matrix because it tells whether path between vertices is available or not.

if $P[i][j] = 0$ then their is no path from i to j but if $P[i][j]$ is $\neq 0$ then their is some path from i to j

Warshall's algorithm :-

```
{ p = G          // p is path mat & G is adj matrix
  for k -> 1 to V
  for i -> 1 to V
  for j -> 1 to V
  p[i][j] = p[i][j] or p[i][k] & p[k][j]
  } // finished
```

☆ program of warshall's algo

In this create graph which is ~~only~~ directed graph (undirected not req).

```
#include <stdio.h>
int g[20][20], v, e, p[20][20];
void creategraph()
{
  int i, j, a, b;

  ~~for(i=1;~~
  printf("enter no of vertices");
  scanf("%d", &v);
  printf("enter no of edges");
  scanf("%d", &e);

for(i=1; i<=e; i++)
{
  printf("enter edge i/p");
  scanf("%d%d", &a, &b);
  g[a][b] = 1;
}
} //end of create graph.

void warshall()
{
int i, j, k;
  for(i=1; i<=v; i++)
  for(j=1; j<=v; j++)
  p[i][j] = g[i][j];
```

```
for (k=1; k<=v; k++)
for (i=1; i<=v; i++)
for (j=1; j<=v; j++)
p[i][j]=p[i][j] || p[i][k] &&
                   p[k][j];

for(i=1; i<=v; i++)
{
  for(j=1; j<=v; j++)
    printf("%5d", p[i][j]);
  printf("\n");
}

void main()
{
  creategraph();
  warshall();
}
```

Backtracking algorithm:

```
algo place (k: queen no)
{
// return true if k^th queen
is placed properly.
else
return false.
};
```

```
algo nqueen (n: no of queens).
{
// x is array of n element 1:n.
k=1
x[k]=0
while(k>0)
{
x[k]=x[k]+1
while (not place(k) and x[k]<=n)
x[k]=x[k]+1

if (x[k]>n)
k=k-1 // backtrack
else
{ if (k==n)
disp array x 1:n
else
{
k=k+1
x[k]=0
}
}
}//end while }//end algo
```

Program

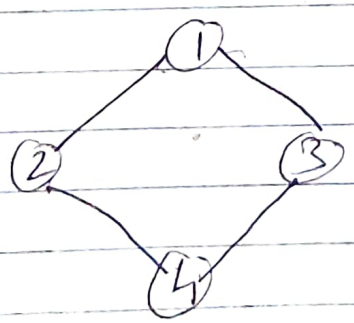
```
int x[10];  -> global declaration.
int place (int k)
 {
  int i;
  for (i=1; i<=k-1; i++)
   {
   if (x[i] == x[k] || abs(x[k]-x[i])
        == (k-i)).
    return 0;
   }
   return 1;
  } // end place

void nqueen(int n)
{
  int [redacted], k=1, i;          // k is queen no.
   x[k] = 0;
   while(k>0)
   {
   x[k] = x[k] + 1;
  while(!place(k) && x[k]<=n)
  x[k] = x[k] + 1;
  if (x[k] > n).
    k = k-1; // backtrack.
   else.
   { if (k==n)
     {
       // print solution
      for (i=1; i<=n; i++)
      printf("%d", x[i]);
      printf("\n");
   }
   else
```

```
{
  k = k+1;
  x[k]=0;
 }; }
} // end while.
} // end of n queen function

Void main()
{
 nqueen(4);
}.
```

Graph color problem:-

Given a graph of v vertices & e edges & given m diff color. We want to color all vertices of graph in such way that adjacent vertices should not have same color. graph is said to be n colorable if more min no of colors req to color the graph.

Eg:- Let graph is 3 colorable & solution are as follows.



V_1	V_2	V_3	V_4
1	2	3	1
1	3	2	1
2	1	3	2
2	3	1	2
3	1	2	3
3	2	1	3

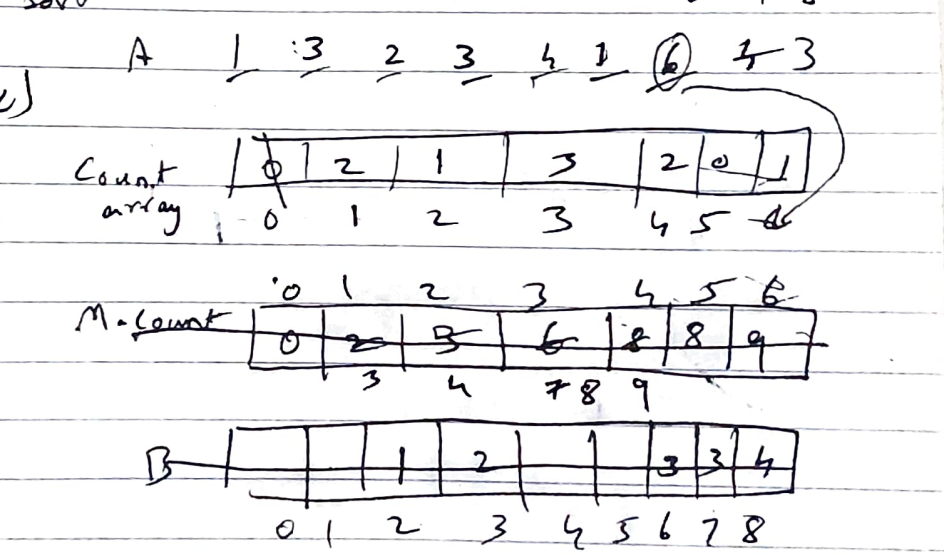
Program :

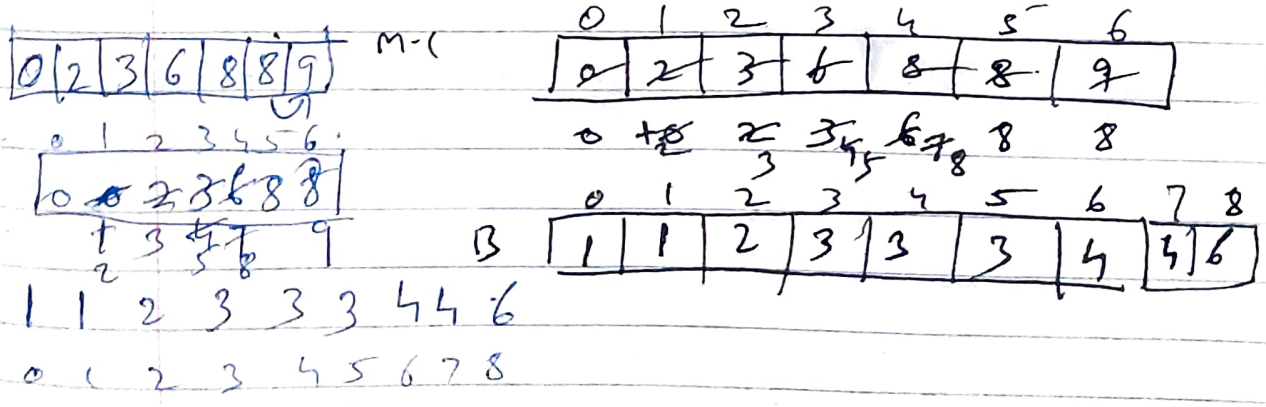
```
#include <stdio.h>
int g[20][20], x[20], v, e;
// void creategraph();
int color(int k)
{
  int i;
  for(i=1; i<=k-1; i++)
  {
  if(g[k][i]==1 && x[k]==x[i])
    return 0;
  }
  return 1;
}

void graphcolor(int n)
{
  int k=1;
  x[k]=0;
  while(k>0)
  { x[k]=x[k]+1;
  while(!color(k) && x[k]<=n)
  x[k]=x[k]+1;
  if(x[k]>n)
    k=k-1;   // backtrack
  else
  if(k==v)
  { for(i=1; i<=v; i++)
    printf("%d", x[i]);
    printf("\n");
  }
```

```
else
{
  K = K+1;
  x[K] = 0;
}
} // end while.

void main()
{
  creategraph();
  graphcolor(3);
}
```

Counting Sort.

A Space O(n+k)
Time (n+k)
Stable





Merge sort:

Simple merge :- Given an array of n elements & given 3 position j, s, t such that all elements from the s-1 are sorted & all elements from s to t are also sorted i.e 2 sorted partition are given back to back. We want to merge these 2 sorted partition so that all elements from j to t are sorted

~~Simple merge~~ :-

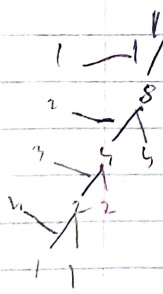
```
void simplemerge(int a[], int j, int s, int t)
{
    int i, j, temp[100], k, w;                    (n)
    i=j; j=s; k=0;
    while (i<=s-1 && j<=t)
    {
        if (a[i] < a[j])
        {
            k++;
            temp[k] = a[i];
            i++;
        }
        else
        {
            k++;
            temp[k] = a[j];
            j++;
        }
    } // while end
```

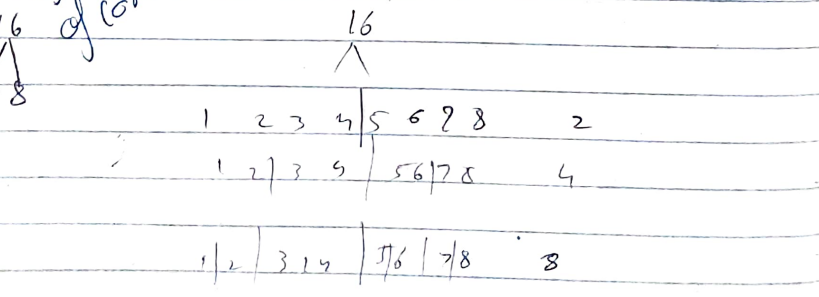
```
if (i > s-1)
{
for (w=j; w<=t; w++)
{
k++;
temp[k] = a[w];
}
}
else
{ for (w=i; w<=s-1; w++)
{
k++;
temp[k] = a[w];
}
}
// copying temp array from a to k into array a
for (w=0; w<=k; w++)
a[j+w] = temp[w];

void merge sort (int a[], int l, int r)
{
int m;
if (l<r)
{
m = (l+r)/2;
merge sort (a, l, m);
merge sort (a, m+1, r);
simple merge (a, l, m+1, r);
}
}
```

```
Call
  mergesort(a, 0, n-1);

void main()
{
  int a[100], n, i;
  printf("enter no of element");
  scanf("%d", &n);
  /* Scan in element */
  for(i=0; i<=n-1; i++)
  {
    printf("enter element %d", i+1);
    scanf("%d", &a[i]);
  }
  mergesort(a, 0, n-1);
  /* print sorted array */
  for(i=0; i<=n-1; i++)
  printf("%d\n", a[i]);
  } /* main end */
```

(8)

Pg-30 of corner

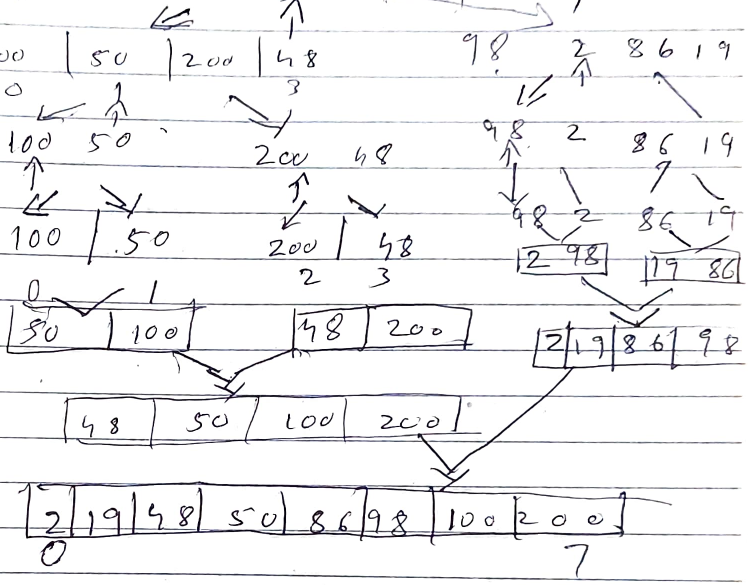




pg 1 to 20 corrn

→ Show steps in which following array will be sorted using merge sort.

100 | 50 | 200 | 48 | 98 | 2 | 86 | 19



Quick Sort (partition sort):-

Show steps in which following array are ~~sorted~~ Partition using partition process.

300, 150, 400, 250, 800, 150, 14, 600

i j

300 150 400 250 800 150 14 600 400

i j

300 150 7 250 800 150 14 600 400

i j

150 300 150 7 250 14 150 800 600 400

j i

2) 50 45 30 18 7 5
i j

5 50
50 45 30 18 7 8
i j

5 45 30 18 7 50

3) 20 25 38 42 56 77
i j

20 25 38 42 56 77
j i

20 25 38 42 56 77

4) 150 160 170 144 3 300 2
i j

150 2 160 170 144 3 300 160
i j

150 2 3 170 144 170 300 160
i j

150 2 3 144 150 170 300 160
j i

144 2 3 150 170 300 160

Develope a func which will partition array a from position l to position r.

```
Int partition (Int a[ ], int l, int r).
{
   Int i, j, t, x;
i = l; j = r; x = a[l];
while (i < j)
{
  while (a[i] <= x && (i <= r).
    i++;
  while (a[j] > x)
    j--;
if (i < j)
{
  t = a[i];
a[i] = a[j];
a[j] = t;
}.
} // end of while.

t = a[j];
a[j] = a[l];
a[l] = t;
return j;
}

void quick (int a[ ], int l, int r).
{ int p;
  if (l < r)
{ p = partition (a, l, r);
  quick (a, l, p-1);
```

```
quick(a, p+1, r);
}
```

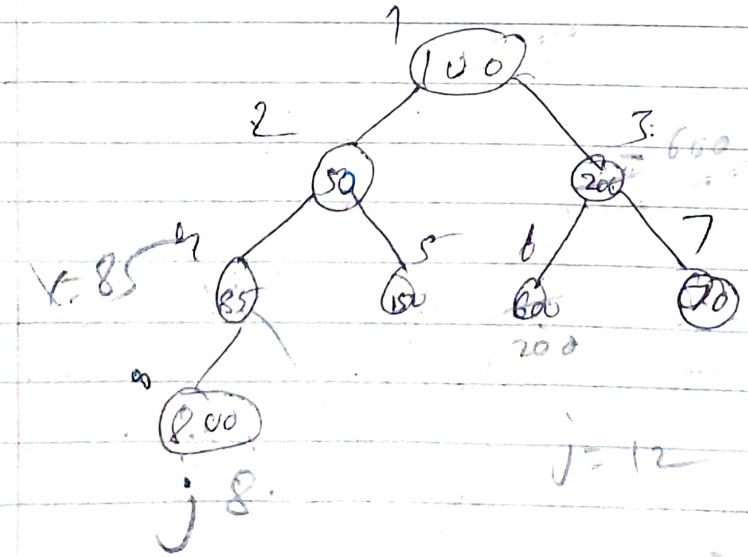
```
Void main ()
{
  quick (a, 0, n-1)
}
  (rest is same)
```

Heap Sort :-

What is heap :- Any array can be represented in form of sequential binary tree in which values are added in a sequence from left to right. This sequential binary tree is called heap Eg :-

100 50 200 85 150 600 70 800

1 2 3 4 5 6 7 8



n=9

i=4

j=12

j 8.

following properties are always true in a heap: for child c, parent p = c/2
for parent p, c_L = p×2, c_R = p×2+1

* max heap: It is sequential binary tree in which every parent node must have a value larger than both child node.

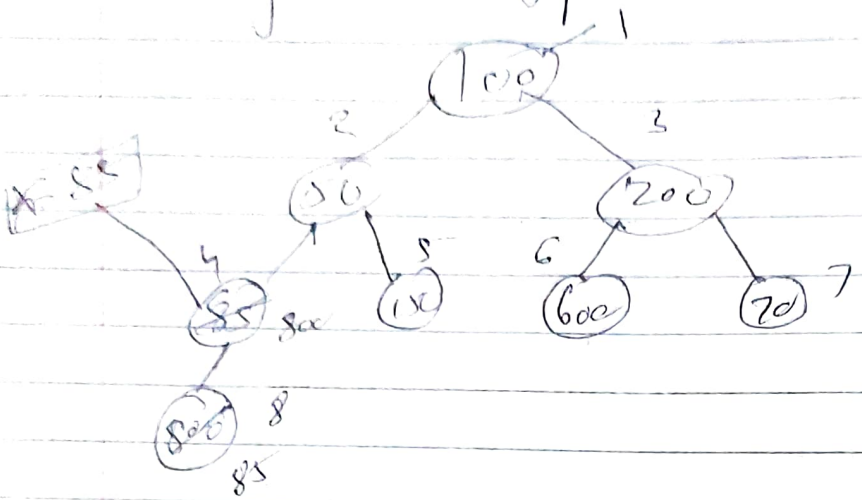
* min heap: It is seq binary tree in which every parent node must have a value less than both child node.

Explain process of heap sort.
Heap sort works in 2 stages
stage 1 - convert array a into max heap
Eg:

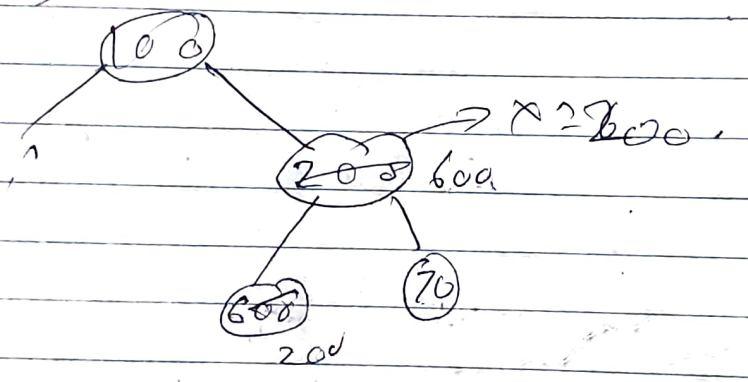
100	50	200	85	150	600	70	800
1	2	3	4	5	6	7	8

a[1] > a[2], a[3]
a[2] > a[4], a[5]
a[3] > a[6] a[7]
a[4] > a[8].
All above condi should be satisfy to make a max heap:
stage 1 will adjust value of each parent so that every parent node has a value larger than its child. we start adjusting last parent & come down till first parent.

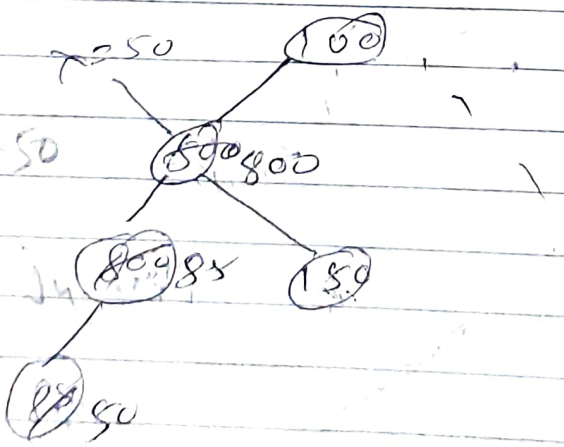
1) adjust value of parent 1



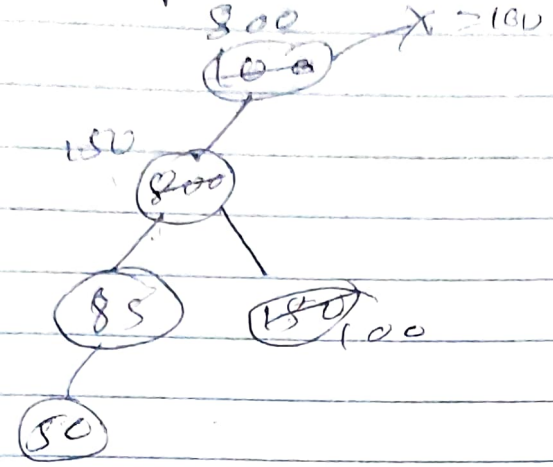
2) parent = 3



3) parent = 2



4) adjust parent = 1 P -> P*2 P*2+1

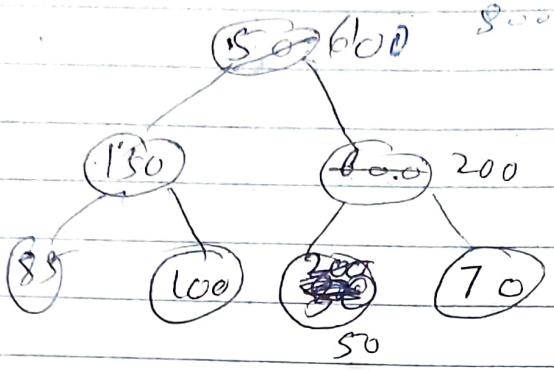



```
void adjust (int a[], int i, int n).
{
  int j, x;
  j = i * 2; x = a[i];
  while (j <= n)
  {
    if (j < n && a[j] < a[j+1])
      j = j + 1;
    if (x > a[j]).
      break;
    a[j/2] = a[j];
    j = j * 2;
  } //end while.
  a[j/2] = x;
} // end adjust
```

Stage 2: Sort the array by using max heap which was found in Stage 1.

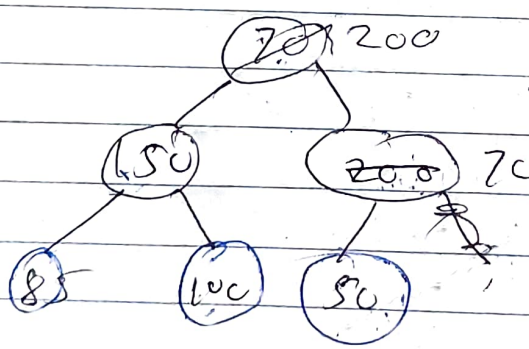
i) swap a[1], a[8]

50 150 600 85 100 200 70 | 800



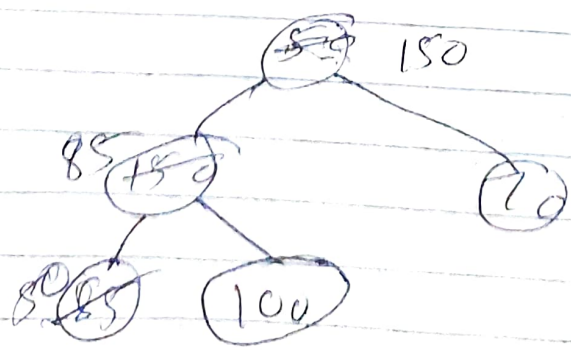
ii) swap a[1], a[7]

70 150 200 85 100 50 | 600 800



iii) swap a[1], a[6]

50 150 70 85 100 200 | 600 800



```
Void main ( )
{
    int a[10], n, i;
    printf("enter no of element");
    scanf("%d", &n);

    for (i=1; i<=n; i++)
    { printf("enter element %d", i);
      scanf("%d", &a[i]);
    }
    // heapify.
    // convert array into max heap.
    for(i=n/2; i>=1; i--)
    adjust(a, i, n);

    // new heap sort.
    for(i=n; i>=2; i--)
    {
    t=a[1]; a[1]=a[i]; a[i]=t;
    //reheapify.
    adjust(a, 1, i-1);
    }
    // print array
    for (i=1; i<=n; i++)
    Printf("%d", a[i]);
    }//end main.
```

$O(n) + (n \log_2 n)$

$O(n \log_2 n)$

← Greedy Algorithm :-

This is technique or type of algo in which we greedyly select a part of solution so that we get optimum result. in other words algo uses some strategy. (usually min, max) to obtain some part from solution set Eg 1). kruskal algo which find min cost spanning tree of a graph is greedy for the edge with min weight.

Eg 2 :- Dijkstra's algo which find shortest distance bet 2 vertices of a graph is greedy because at every step it takes vertex at shortest

Explain eg using kruskal

— Divide & Conquer Algo :- This type of Algo is base on strategy of divide & rule. Algo will divide the given data set into smaller data set. Solution is first found on smaller data set. these small solutions are now combine (merge). To get final solution.

Eg 1 quick sort repeatedly partition the given array into left & right parts.

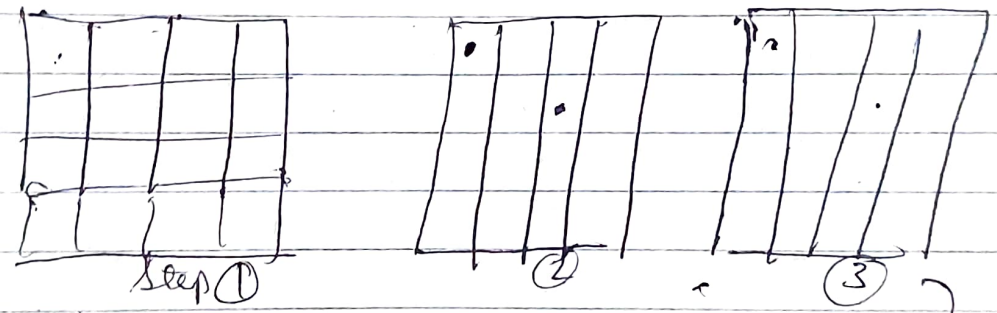
Eg 2 :- merge sort repeatedly divide array by finding mid point.

01123S.8

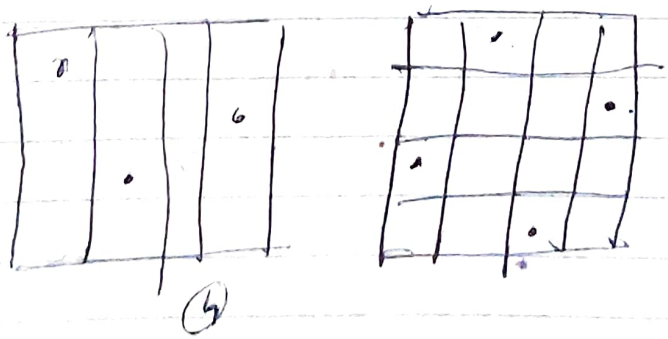
- Backtracking algo: This type of algo has ability to rectify or repair solution which was found in previous steps. i.e such algo has artificial intelligents because it understands that partial solution found before is wrong & it has ability to rectify the partial solution.

Eg: N queen problem is problem of placing n queens on n×n chess board. in such a way that no queens should attack each other for more step for 4 queen problem.

1) Place 1st queen 2nd 3rd 4th queen.



3rd queen can not placed because partial solution is wrong backtrace to 2nd queen & adjust the place.



Dynamic algorithm

This type of algo stores result found in current pass in a buffer so that this result can be utilize by next pass

Eg:- generate 20 fibonacci no's

In this problem 2 variables a & b act as a buffer because they store previous 2 fibonacci no. so that next fibonacci no is found as c = a+b.

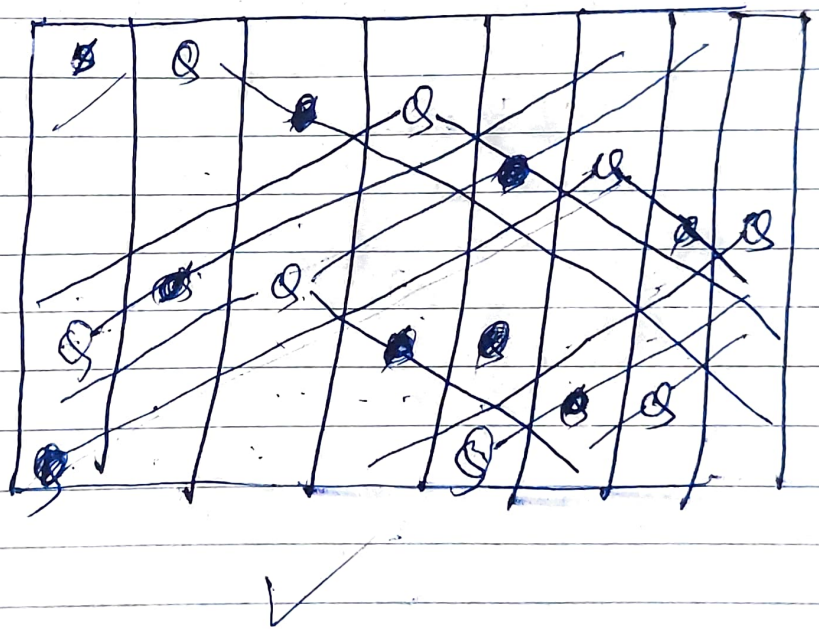
```
void main()
{
int a=1, b=1, c, i;
printf("%d %d", a, b);
for(i=1; i<=18; i++)
{
c = a+b;
printf("%d", c);
a=b;
b=c;
}
}
```

Eg2 Pascal triangle

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
1 5 10 10 5 1
```

Sat – 7 to 5:30
Sun – 7 to 1

It can be done by saving i th no in buffer
so that i+1 th no can be generated
by recursive addition of successive values



Interpolation Search :-

This search req array to be sorted in increasing order & all values should be equally spaced. address of any element can be found using a formula.

$$addr\ x = low + \frac{high - low}{a[high] - a[low]} * (x - a[low])$$

Eg if x = 50 then addr of 50 can be found out as,

$$= 0 + \frac{9-0}{100-10} * [50-10]$$

$$= \frac{1}{10} * 40$$

$$= 4.$$

5	10	15	20	25	30	35	40	45	50
	10	20	30	40	50	60	70	80	90	100
	0	1	2	3	4	5	6	7	8	9

Robust Interpolation Search :-

This method use when array is sorted but values are not equally spaced.

~~Mid~~.

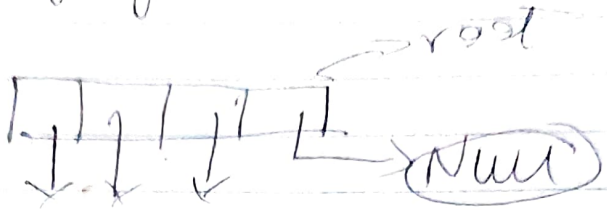
```
void RobustInt(int a[], int n)
{
int low, high, x, mid;
printf("Enter value to search");
scanf("%d", &x);
low = 0; high = n-1;
mid = low + (high - low) * (x - a[low]) / (a[high] - a[low]);
while (low <= high && a[mid] != x)
{
if (a[mid] < x)
low = mid + 1;
else
high = mid - 1;
mid = low + ------
}
if (low > high)
printf("Not found \n");
else
printf("val found at %d \n", mid);
}
```

Tries (dictionary data structure).

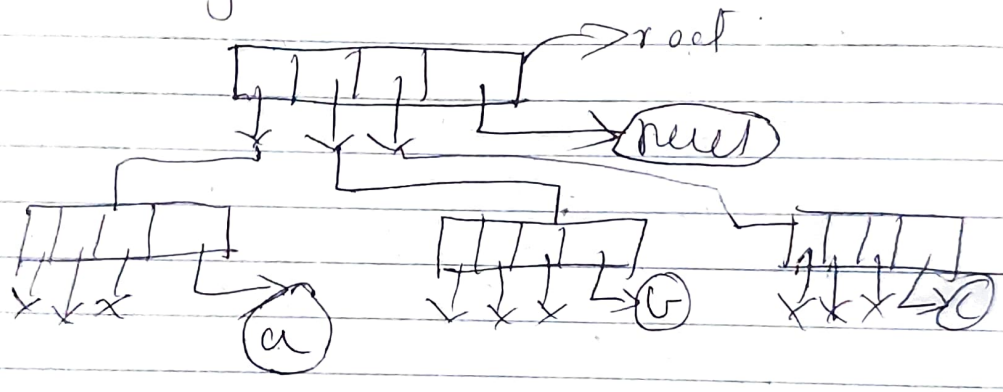
(Digital search tree) provided use digit.

Tries is dictionary Data Structure in which every node contains n pointers each pointer is responsible for words starting with a particular letter. One additional right most pointer is store in each node which points to actual string. Consider a Trie

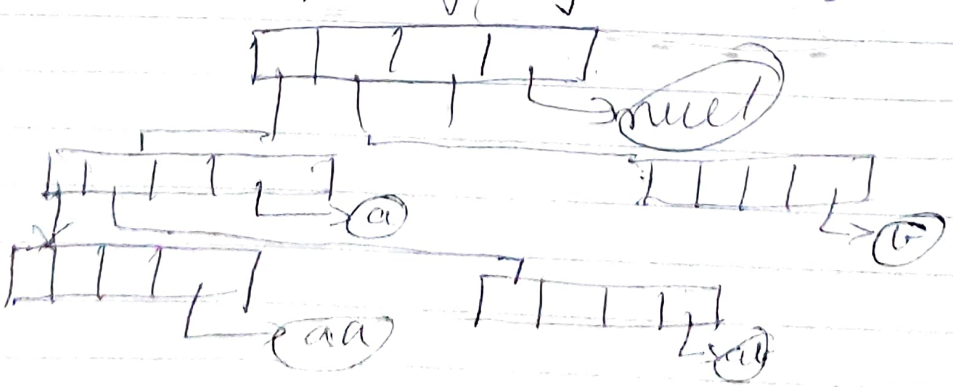
of order 3 which can construct any word of english using 3 alphabet a,b,c.



The 1st 3 ptrs of this node will lead to strings starting with a, b, & c respectively. The last pointer pts to null string.

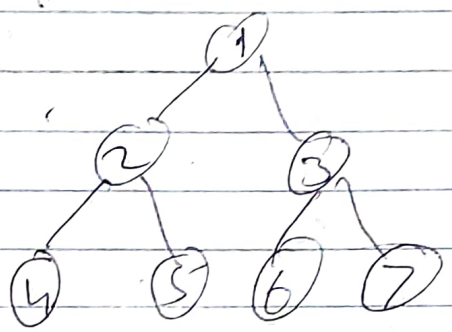


1st child of root constitutes to a string a similarly other childs will give string b & c. If we continue in this fasion we would get all possible strings begining with a, b, c



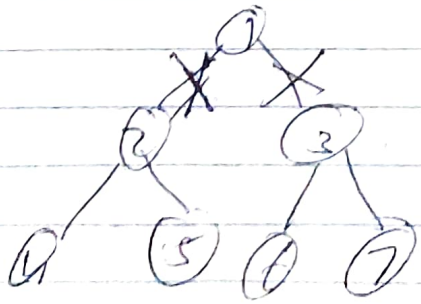
In order traversal of such a tree will print all the words in alphabetical order which can be use as dictionary of all possible words.

☆ Topological sort :- Topological sort is traversal technique which is applicable only for acyclic graph i.e graph which does not have cycle & every vertex in this graph is reachable only in 1 way Eg :-



Steps in which topological sort will traverse this graph are

→ Insert Vertex 1 in queue. q: 1
→ k = front element of queue.
→ ∴ k = 1 print vertex k i.e 1
→ insert all adjacent vertex of k
→ queue = q: 2 3
→ Also cut edges (1, 2) & (1, 3)



→ k = front element of q ∴ k = 1.

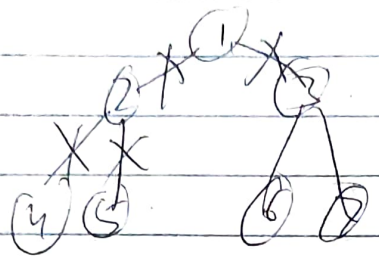
Print k ... i.e 1

Now insert all adjacent vertex of k.

q

3 4 5

∴ all out edges (2, 4) & (2, 5)



→ we proceed till the queue is empty.

Topology

```
algorithm topology (k : vertex)
{
    insert (q, k)
  while (not empty (q))
  {
    k = delete (q)
    print vertex k.
    // check adj vertices of k.
    for i -> 1 to v.
    {
      if (g[k][i] ≠ 0)
      {
        insert (q, i)
        g[k][i] = g[i][k] = 0 ;
      }
    } // for
  } // while.
}
```

☆ Analysis of algorithm

Time complexity of sorting algo:
2 major factor which affect the time of sorting algo are

-> data size (n)
-> No of comparision.

Note:- No of swaps is not considered as a factor which affects time. This is because No of swap are dependent on nature of data.
Eg:- If data is already sorted in increasing order then any algorithm will perform 0 swaps.

Analysis of bubble sort:-

Refer algo of bubble sort

```
for i -> n-2 down 0
    for j -> 0 to i
        if (a[j] > a[j+1])
            swap a[j], a[j+1]
```

assume size of array i.e n=5
then pass of bubble sort can be shown using counter i & j as follows

```
i   j
3   0
    1
    2
    3
```
} 4 comparition.

1	1	
2	2	} 3 comparision
	1	
1	0	} 2 comparision
	1	
0	0	} 1 comparision.

Total no of comparision will decide total time required to run bubble sort.

Total comparision $= (n-1) + (n-2) + \cdots$

$= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

$$\boxed{T(n) = O(n^2)}$$

Time req to sort n elements is of the order of n^2. This means time req to sort n elements is always less than n^2

★ Big-oh notation :- $O(f(n))$ is set of all functions $g(n)$ such that value of $g(n)$ is always $< f(n)$ for some n greater than n_0.

$$O(f(n)) = \{ g(n) \mid g(n) < f(n) \} \text{ for } n > n_0$$

Eg $O(n^2) = \{ n, n\log n, \frac{n}{2}, \frac{n^2}{2} - \frac{n}{2} \}$ for ~~$n > n_0$~~ $n > 0$

Analysis of Selection Sort

```
for i -> 0 to n-2.
{
  p = i.
  min = a[i]
  for j -> i+1 to n-1
  { if (a[j] < min)
    {
      min = a[j]
      p = j;
    }
  }
  swap a[p], a[i]
}.
```

assume $n = 5$ then passes of selection sort can be shown by using a counter i & j as follows:

i j

0 1, 2, 3, 4 } 4 comparisons

1 2, 3, 4 } 3 Compa

2 3, 4 } 2 Compa

3 4 } 1 Compa

Total Compa = $(n-1) + (n-2) + \cdots$

$= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

$T(n) = O(n^2)$

Radix Sort :- (Bucket Sort)

No of digit

```
void radix (int a[], int m, int n)
{
 int count[10], exp=1, i, bucket[100][10];
 for (i=1; i<=m; i++).
 { /* initialize all count */
 for (j=0; j<=9; j++).
 count[j] = -1;
 /* map all array element into bucket */
 for (j=0; j<=n-1; j++)
 {
   digit = (a[j]/exp) % 10;
   col = digit;
count[digit] = count[digit] + 1; row = count
   bucket[row][col] = a[j];
 }.
 /* copy all buckets back into array */
   x=0
   for(j=0; j<=9; j++)
   {
    if (count[j] != -1)
    {
     for (k=0; k<=count[j]; k++)
     {
      a[x] = bucket[k][j];
      x++;
     }
    } // end if
   } // end for
   exp = exp * 10;
 } end of outer for
 } end of radix
```

```
void main()
{
int a[100], n, i;
printf("enter no of element");
scanf("%d", &n);
for (i=0; i<=n-1; i++)
{
printf("enter element %d", i+1);
scanf("%d", &a[i]);
}.

radix (a, 3, n); /*assume each 3 digit
                   no */.

/*print sorted array */
for (i=0; i<=n-1; i++)
printf("%d", a[i]);
}.
```

[digit];

Analysis of insertion sort :-

```
for i -> 1 to n-1
{
x = a[i]
j = i
while (a[j-1] > x & j > 0)
{ a[j] = a[j-1]
  j = j-1
}.
a[j] = x.
} // end for.
```

Worst case of insertion sort:

- Consider array a of 5 element as follows

5 4 3 2 1.

In pass 1 2 comparisions are made (4, 5) & (Boundary condi')

Array after this 1st pass is

4 5 3 2 1.

In pass 2 3 comparision are made (3, 5) & (3, 4) & boundary.

Array after pass 2 is.

3 4 5 2 1.

Similarly remaining passes execute ~~& total~~ comparision.

Total comparision = 2+3+4+5+...+n

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

$= \frac{n^2}{2} + \frac{n}{2} - 1$

$T(n) = O(n^2)$.

- Best case of insertion :-

Consider a sorted array of 5 element

1 2 3 4 5.

In pass 1 only 1 comparision

(1 done (2, 1)
2 pair 2 again 1 comparision
11 done (3, 2).
In each pass their will be only 1 comparision. hence total comparision
$= n-1$
$\therefore [T(n) = \Omega(n)]$

Definition of Ω notation :-

$\Omega(f(n))$ is defines a set of all function $g(n)$ such that $g(n) \geq f(n)$ for $n > n_0$ (cutoff)

$$\Omega(f(n)) = \{g(n) \mid g(n) \geq f(n) \text{ for } n \geq n_0\}$$

★ Knapsack :- given a knapsack having a capacity M & given n diff object. also given is profit & weight for every object. we want to fill knapsack in such a way that sack full capacity will reach & the profit will be max. This is greedy problem because object will be taken in decreasing order of profit by weight. i.e object having max profit / weight will be filled 1st in knapsack.

Eg:- $n = 7$ $M = 15$

P

10	5	15	7	6	18	3
1	2	3	4	5	6	7

W

2	3	5	7	1	4	1
1	2	3	4	5	6	7

(i) calculate profit/weight for all objects.

	No.
- 5	1
- 1.6	2
- 3	3
- 1	4
- 6	5
- 2	6
- 3	7

(ii) Sort both profit & weight array in descending order as profit/weight.

P

6	10	3	15	8	18	7
1	2	3	4	5	6	7

W

1	2	1	5	4	3	7
1	2	3	4	5	6	7

(iii) fill the knapsack:

initial capacity = 15

profit = 0

object packed	capacity remain	profit
1	14	6
2	12	16
3	11	19
4	6	34
5	2	42 profit
6	0	42+3.33
		45.33

0/1 knapsack :- It is method in which object is either taken or left.

fractional :- It is method in which object may be broken into smaller fraction.

Analysis of Merge sort :- Time req to sort n element by merge is given by recurrence relation.

$$T(n) = 2(T(n/2)) + n.$$

where $T(n/2)$ is time req to sort $n/2$ element & n are no of comparision needed to merge 2 partition of size $n/2$

$\therefore T(n) = 2T(n/2) + n.$

divide both LHS & RHS by n,

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1. \quad -(1)$$

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1 \quad -(2)$$

$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1 \quad -(3).$$

$$\vdots \quad \vdots \quad \vdots$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1 \quad -(*).$$

add all LHS & R.H.S & compare

$$\frac{T(n)}{n} = \frac{T(1)}{1} + \log_2 n$$

$$T(n) = n + n\log_2 n$$

$(5/3 \times 2) = 333.$

$$T(n) = O(n \log_2 n).$$

Analysis of quick sort :-

Best case of quick sort :- It perform in best case only if partition is always found in middle i.e 2 equipartition are of equal length. In such case, time which is given by recurrence relation.

$T(n) = 2T(n/2) + n$

where $T(n/2)$ is time req to sort $n/2$ elements & n are no of comparision needed to partition array of n elements $\therefore T(n) = O(n \log n)$

Worst case :- 2 worst cases of quick sort are

- array is sorted in increasing order
- array is in reverse order -

Consider a sorted array as follows.

1 2 3 4 5 (5 comparision needed)

2 3 4 5 (4 comparision needed)

3 4 5 (3)

$\therefore T(n) = n(n-1)(n-2) + \cdots 1$

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

$T(n) = O(n^2)$

Analysis of Radix sort

$T(n) = O(mn)$

Where m are no of digits.

n are no of elements in every pass. n values of array are adjusted according to digit as their are m digit & n adjustment in each pass performance is $O(mn)$.

Sum of subset

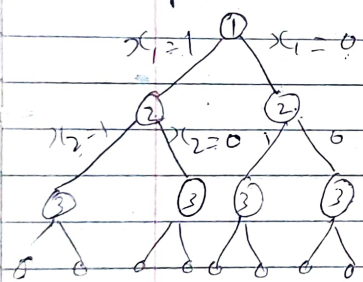
$w[1:6] = \{5, 10, 13, 15, 18\}$

$n = 6, \quad m = 30.$

	1	2	3	4	5	6
x	1	1	0	0	1	0

0/1 x_i

State space tree (DFS)



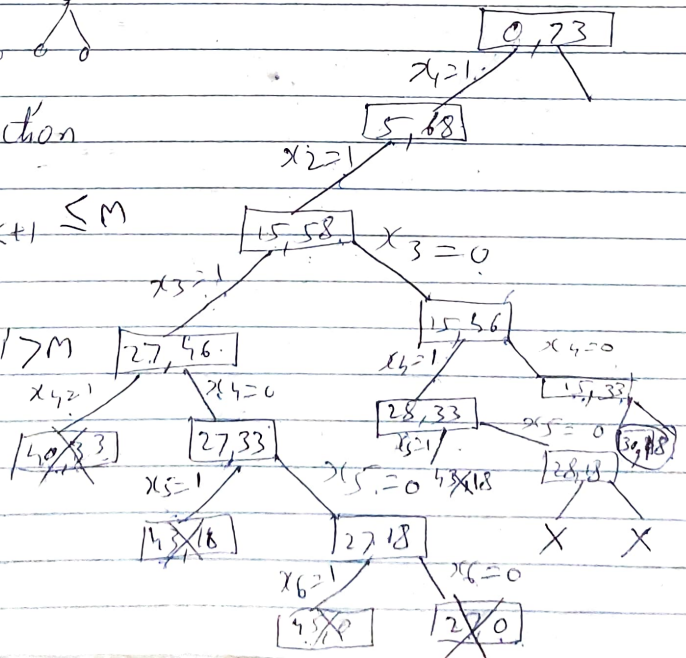
6 weight level (7) height = 6.

2^6 = path $\rightarrow 2^n$

Bonding function

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq M$$

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > M$$



→ KMP algo.

T = abcabdeabcb. (n)
P = abcdabcy. (m)
abcd

Possible proper prefix · [a, ~~ab~~, ab, abc]
suffix [d, cd, bcd]

Naive O(mn)
KMP O(m+n).

Eg:- T = abcdabc.
P = abab.

→ longest Proper Prefix which is also Proper suffix.

To prepare lps-

```
j = 0
lps[0] = 0
i = 1
while (i < p.length)
  { if (p[j] == p[i])
      { lps[i] = j+1
        i++
        j++
      }
    else
      { if (j != 0)
          { j = lps[j-1] }
        else
          { lps[i] = 0 }
        i++ ~~j++~~ }
}
```

	i	j		
	a	b	a	b
lps	0	0	1	2
index	0	1	2	3

	i	j						
Pattern	a	b	c	d	a	b	c	y
lps	0	0	0	0	1	2	3	0
index	0	1	2	3	4	5	6	7

a b c d e f g h
d e f

a b c d a b c a b c d f
a b c d f

a b c d a b c

P1 a b c d a b c a b f
0 0 0 0 1 2 0 1 2 0

P2 a b c d e a b f a b c
0 0 0 0 0 1 2 0 1 2 3

P3 a a b c a d a a b e
0 1 0 0 1 0 1 2 3 0

P4 a a a a b a a c d
0 1 2 3 0 1 2 0 0

Example working.

T a b a b c a b c a b a b d
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

P a b a b d
0 0 1 2 0

index 1 2 3 4 5

Linear Probing

$h(k) = k \bmod m$ (11)

52	=	52 mod 11	= 8	
23	=	23 mod 11	= 1	
78	=	78 mod 11	= 1 ⇒ collision	
3	=	3 mod 11 = 3	Insert 1+1 = 2	
0	=	0 mod 11 = 0		
69	=	69 mod 11 = 3	... 3+1	
29	=	29 mod 11 = 7		
11	=	11 mod 11 = 0	0+1 = 1	
13	=	13 mod 11 = 2	1+1 = 2	
-2	=		2+1 = 3	
-8	=		3+1 = 4	

0	0
1	23
2	78
3	3
4	69
5	11
6	13
7	29
8	52
9	
10	

Hash table

$11 \mid -8$ (-1) → -11 → 3

$11 \mid -2$ (-1) → -11 → 9

(1) $2k \bmod -2 \bmod 11 = k$

$a + 2 = 11$

$a = 11 - 2 = 9$

Quadratic Probing

$$h(k,i) = \left(h(k) + c_1(i) + c_2 i^2\right) \bmod m$$

$$= h(k) + c_1(i) + c_2 i^2 + c_3 i^3$$

=

0	52
1	23
2	
3	3
4	78
5	
6	
7	69
8	82
9	
10	

$= 52 + 1 \cdot 0 + 2 \cdot (0)^2$

$= 52 \bmod 11$ ——— (8)

$23 \bmod 11 = 1$ ⟶

$78 \bmod 11 = 1$ collision

$52 \bmod 11 = 8$ ✓

$23 \bmod 11 = 1$ ✓

$78 \bmod 11 = 1$ collision ∴

$78 + 1 \cdot (1) + 2 \cdot (1)^2$

$78 + 4 = 82 \bmod 11 =$ (4)

$3 \bmod 1 = 3$

$0 \bmod 11 = 0$

$69 \bmod 11 = 3$ collision

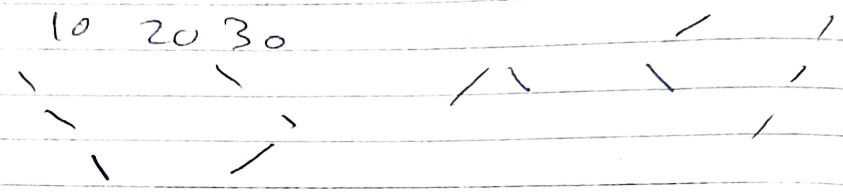
$69 + 1 \cdot (1) + 3 \cdot (1)^2 = 73 \bmod 11$

$= 7$

$69 + 1 \cdot (2) + 3 \cdot (2)^2$

10 20 30 40 50 60 70

40



$\frac{2nC_n}{n+1}$ $\frac{2n3C_3}{3+1}$ =

Job Sequencing with deadlines

$n=4 (P_1 P_2 P_3 P_4) = (100, 10, 15, 27)$

$(2, 1, 2, 1)$

J J_1 J_2 J_3 J_4 J_5 $n=5$

P 20 15 10 5 1 Unit time

dead 2 2 1 3 3

1 2 3 4 5 6

5 3 3 2 4 2

200 180 190 30 120 100

J_2 J_1 J_4

0 — 1 — 2 — 3 } = 40

{ J_2 J_1 J_4 }

{ J_1 D_2 J_4 }

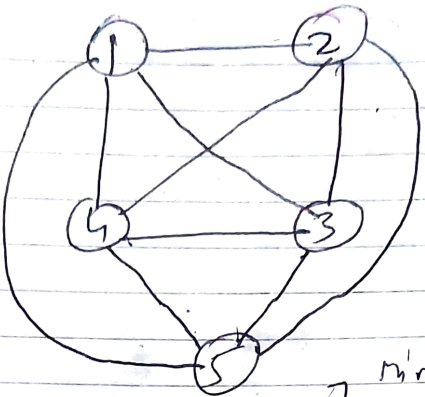
J	C	slot assign	Job	Prof.
	1	[1, 2]	J_1	20
	2	[0, 1]	$\{J_1 J_2\}$	20+15
J_4		[2, 3]	$J_1 J_2 J_4$	20+15+5

J_4 J_3 J_1 J_2

0 1 2 3 4

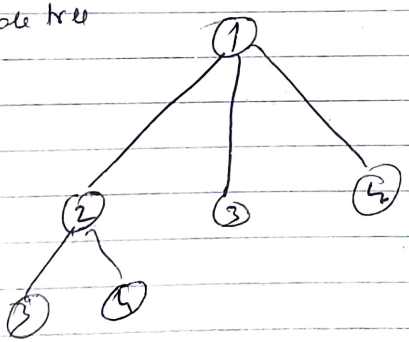
	J_1	2	3	4	5	6	J_7
	35	30	25	20	15	12	5
	3	4	4	2	3	1	2

TSP Branch & Bound



						Min row
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	19	6	18	∞	3	3
5	16	4	7	16	∞	4
						21

State Space tree



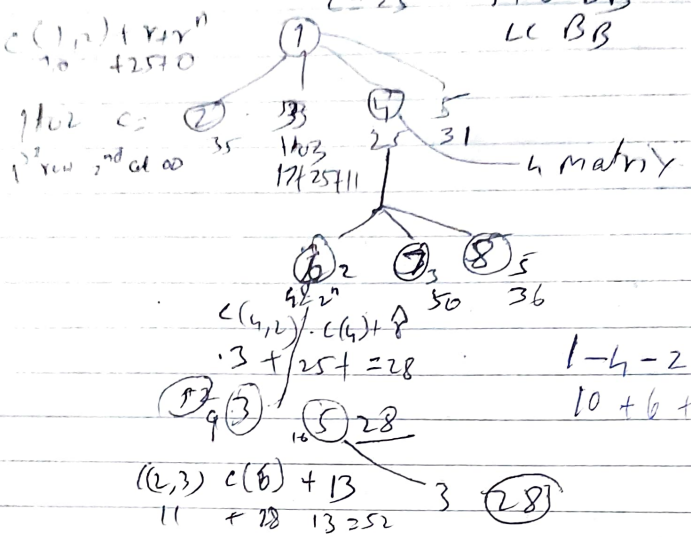
1	∞	10	26 (17)	0	1
2	13 (12)	∞	14 (11)	2	0
3	10	3	∞	0	2
4	16 (15)	3	15 (12)	∞	0
5	12 (11)	0	03	12	∞
	1	0	3	0	0

21+3+2+3

Min cost of tour 25 upper ∞ c 8

c = 25

FIFO BB
LC BB



GS.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4
5	6	:	8
9	10	7	11
13	14	15	12

IS

$c(x) = f(x) + g(x)$

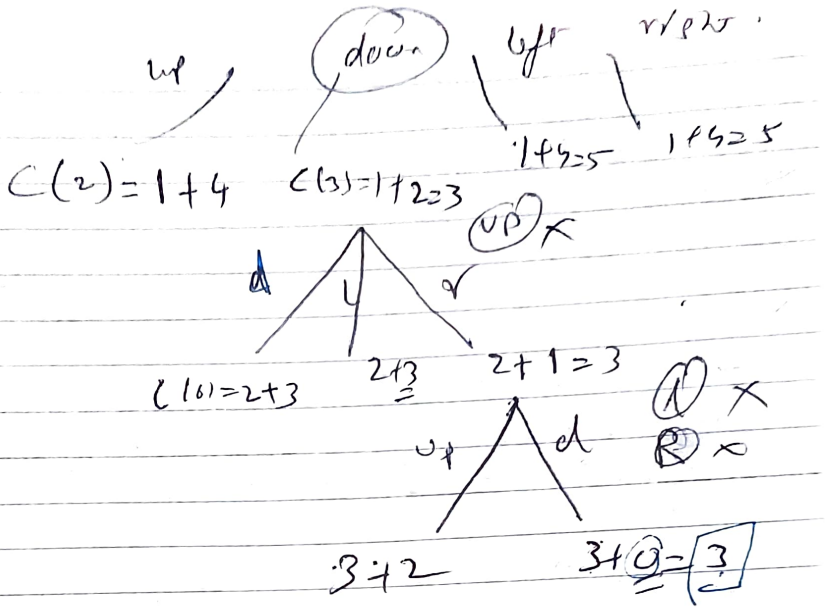
$f(x)$ → No of moves from zs.

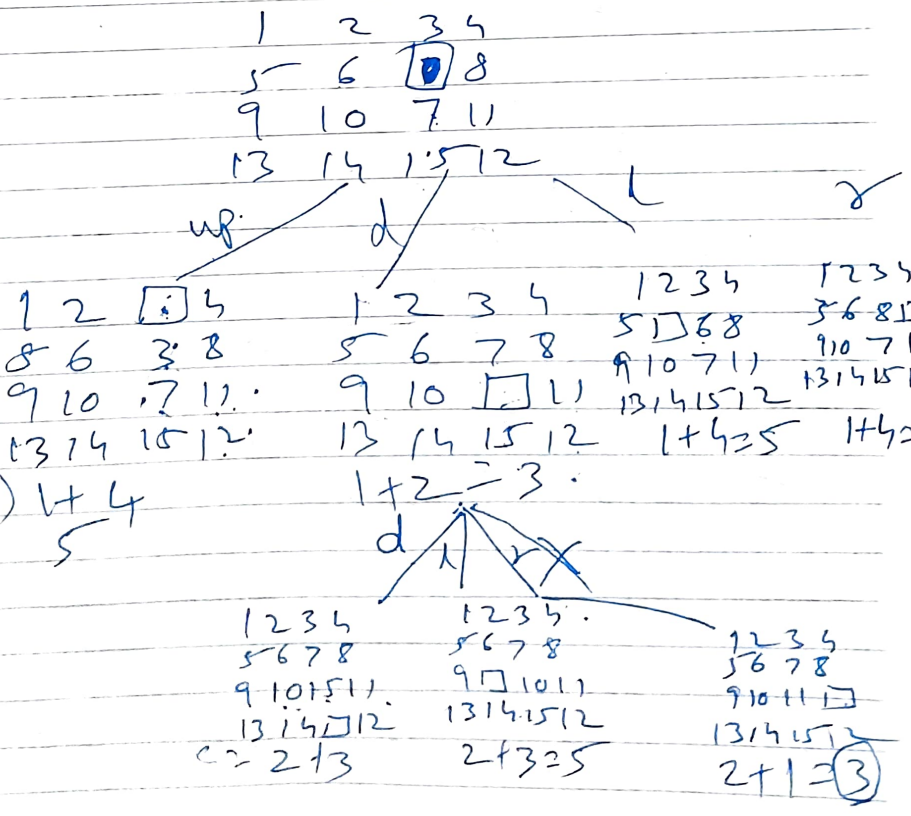
$g(x)$ → No of non blank that are not in their goal pos

1	2	3
4	5	6
7	8	

1	2	3
4	8	5
7		6

C(1) = 0 + 3

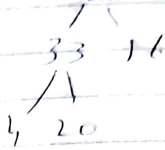

up
down
left
right
1+4=5
1+5=5
C(2)=1+4
C(3)=1+2=3
up
d
r
C(0)=2+3
2+3
2+1=3
u
d
3+2
3+0=3


1 2 3 4
5 6 8
9 10 7 11
13 14 15 12
up
d
l
r
C(x) 1+4
5
1+2=3
d
C=2+3
2+3=5
2+1=3

→ Heapify analysis time complexity

→ Build Max heap

→ Heapification



→ Heapification analysis

Number of swapp = height of node.

Max distance node can cover = height of tree

height = $O(\log_2 n)$

Build Max heap

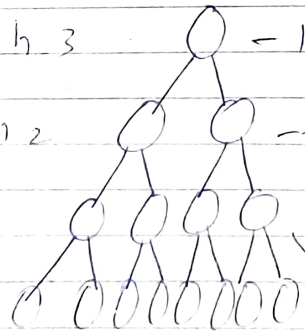
for i = n/2 to i >= 1 — O(n)

{ heapify (A, i, n) — n(log n)

}.

Total $(n \log_2 n)$ = not true.

$O(n)$



Total time at height h = Total node at h x $\log_2 n$

Sum of each node at height (Total node at height) $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

Hence total time taken at height 'h' = $\left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h)$

Total leaf node → $\lceil \frac{n}{2} \rceil + 1$ for $n = 10$ (5+1) to 10 all leaf

$$\sum_{h=0}^{\log_2 n} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h)$$

↓

$O(h)$

$$= \frac{cn}{2} \sum_{h=0}^{\log_2 n} \left\lceil \frac{h}{2^h} \right\rceil$$

$$< \frac{cn}{2} \sum_{h=0}^{\infty} \left\lceil \frac{h}{2^h} \right\rceil$$

harmonic progression.

↓ Z Convergence Approach

$$< \frac{nc}{2} \times$$

$$< O(n) \Rightarrow$$ Complexity to build a Max heap is $O(n)$

∴ heapsort.

{ Build max heap — $O(n)$

{ for (i = N to i >= 1 — $O(n)$

{ swap — const.

heapify — $\log(n)$

} } } $O(n \log n)$

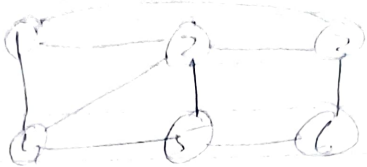
$32 \cdot (\log_2 32)$

Total time = $n \log n$ $32 \cdot 5 = 160$

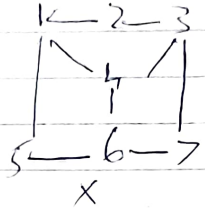
$= O(n \log n)$

32 50 75 100

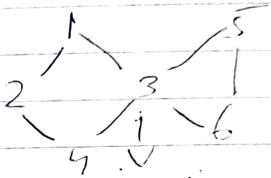
Hamiltonian cycle / circuit



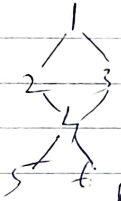
1, 2, 3, 6, 5, 4, 1
1, 2, 4, 5, 6, 3, 1
1, 4, 2, 5, 6, 3, 1
X 2, 3, 4, 5, 4, 1, 2



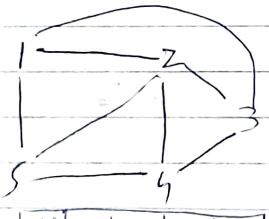
X



Junction
Articulation
Point



pendant vertices



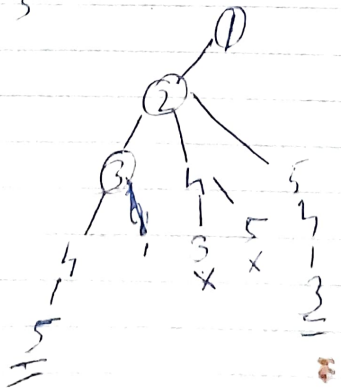
G

	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

x

1	2	3	4	5

1 2 3 4 5



```
Algo hamiltonian (k)
{ do
  { nextvertex (k);
    if (x[k]==0)
        return;
    if (k==n)
        print (x[1:n]);
    else
        hamiltonian (k+1);
  } while (true);
}
```

```
Algo nextvertex (k)
{ do
  {
    x[k]= (x[k]+1) mod (n+1);
    if (x[k]==0) return;
    if (G[x[k-1], x[k]] ≠ 0)
    {
      for j=1 to k-1 do if (x[j]==x[k]) break;
      if (j==k)
        if ((k<n or (k==n)) && G[x[n], x[1]] ≠ 0
            return;
    } while (true);
}
```

1	2	3	4	5
1	2	3	5	0
1	2	4	0	0
1	2	5	4	3

condition
- no must not in array
- edge must there with previous
- last edge must have connection with 1.

$n!$
$(n-1)!$
$n!$
$O = (n^n)$