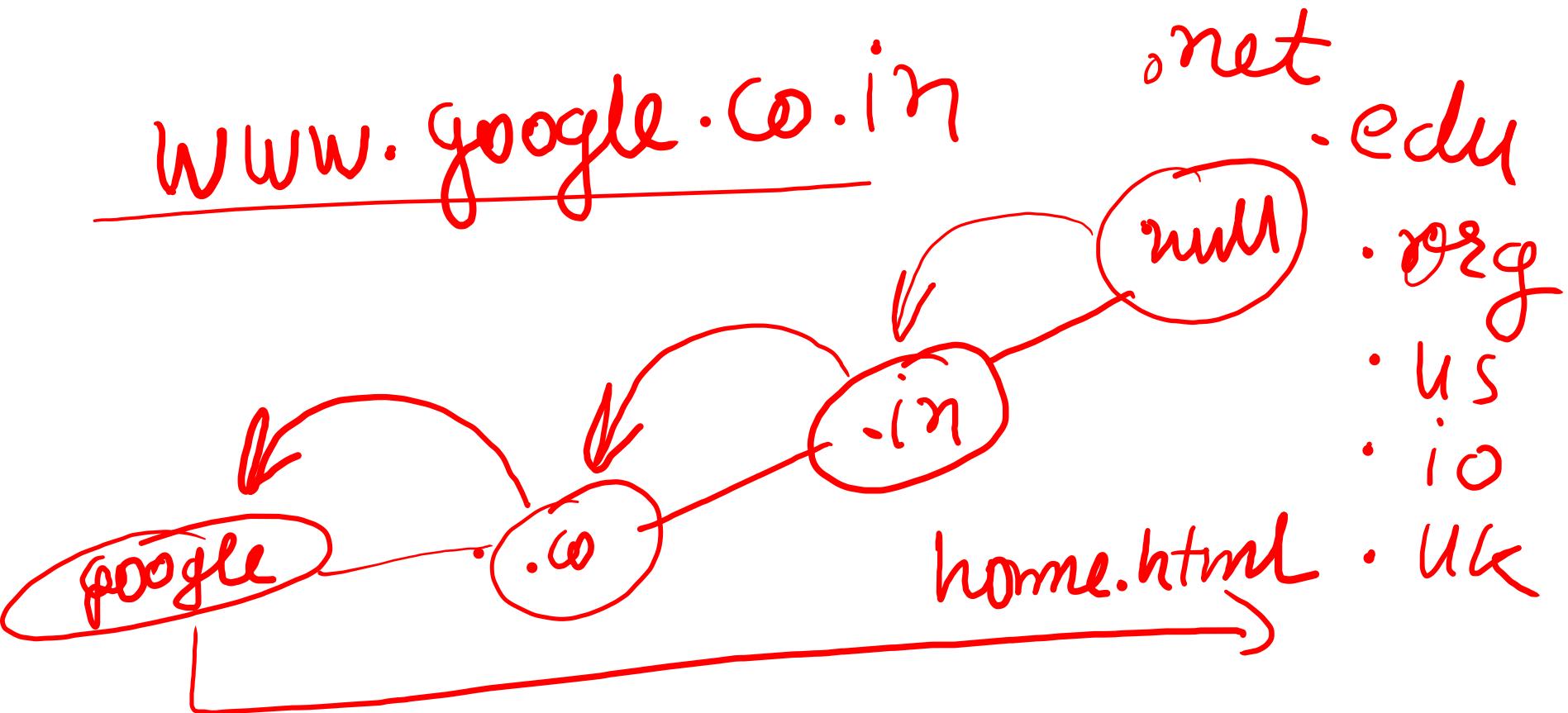


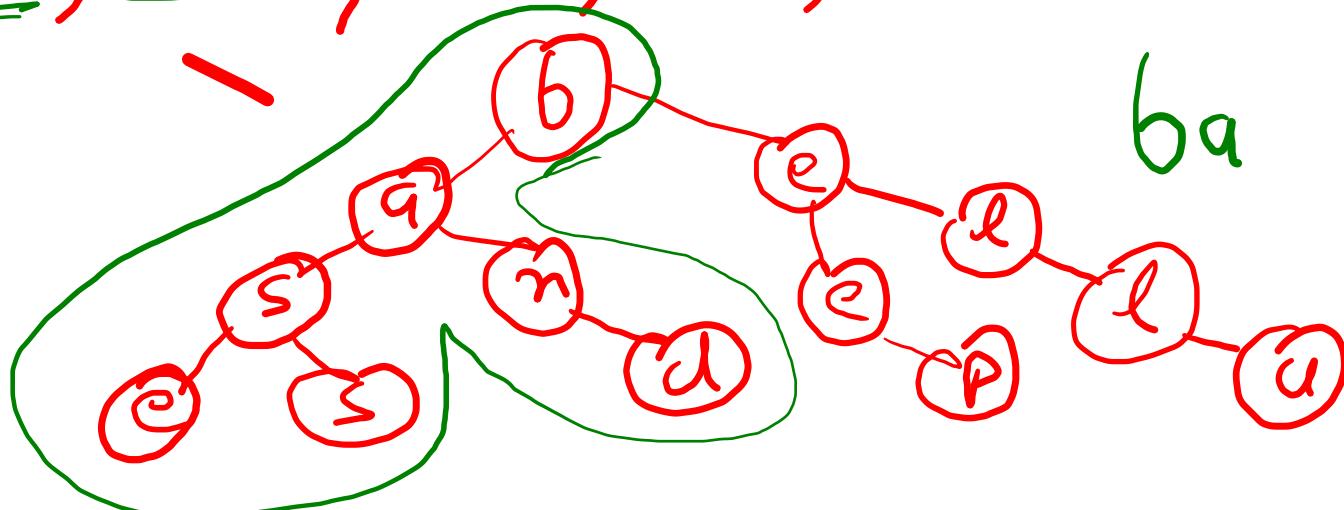
www.google.co.in



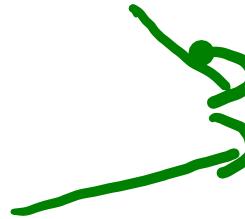
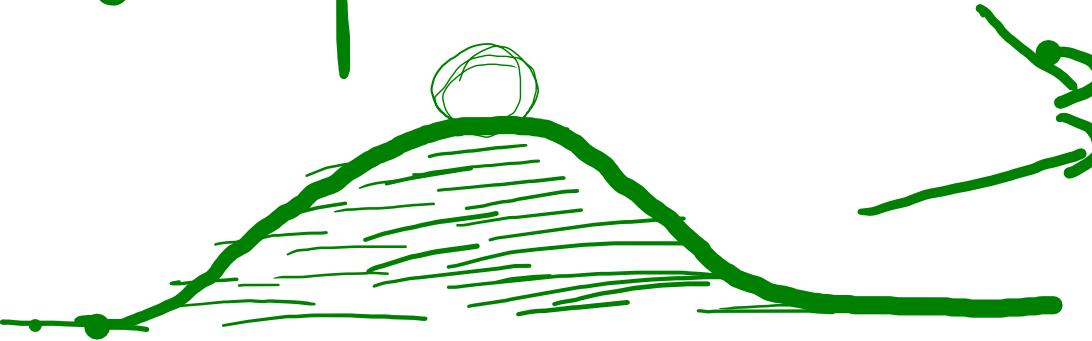
2

Tries → Tree → Auto correct  
Auto complete

base, band, bella, bee, beep, bass



Heap Sort  $\rightarrow$  Tree Based Sort



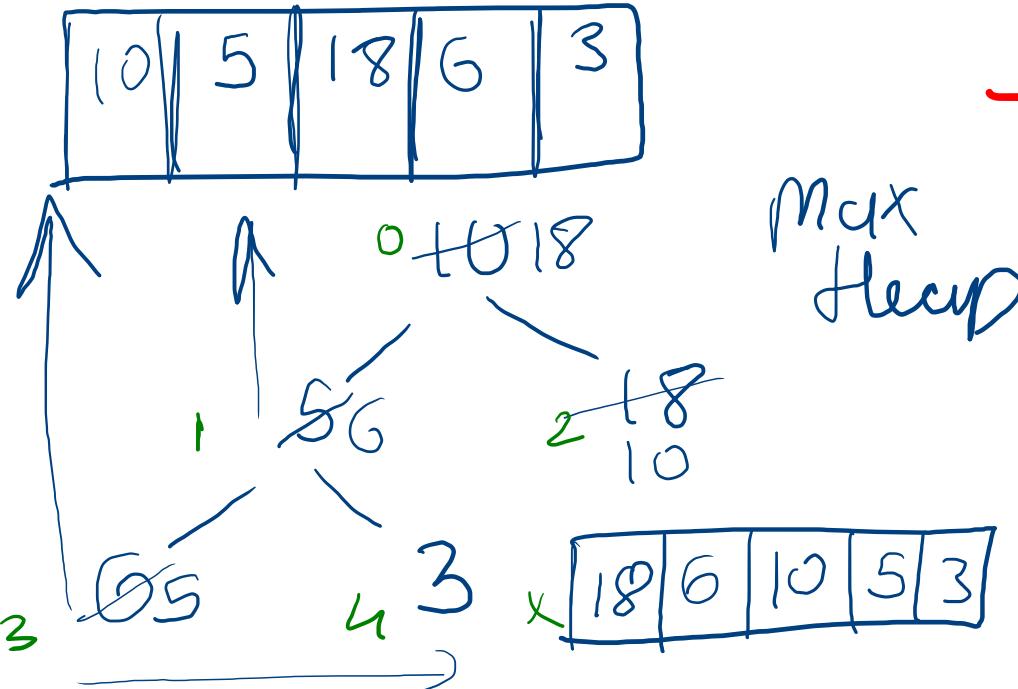
Heap  $\rightarrow$  Min

Parent  $<$  child

Max

Parent  $>$  child

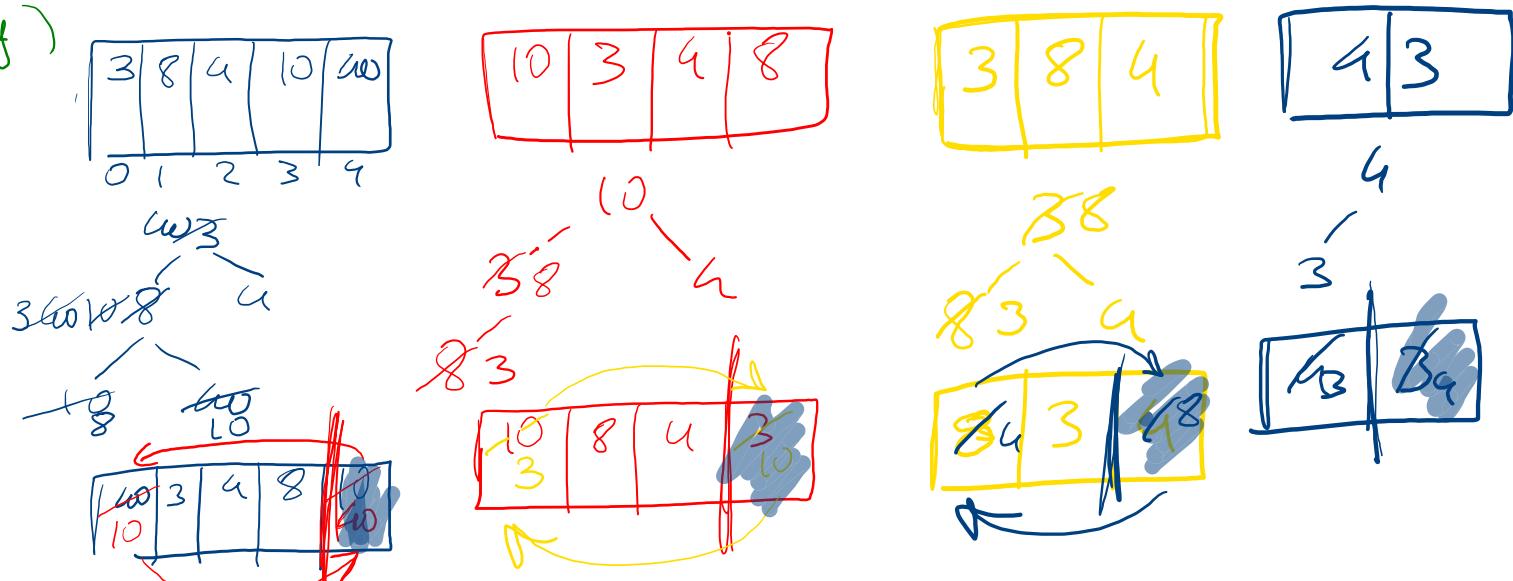
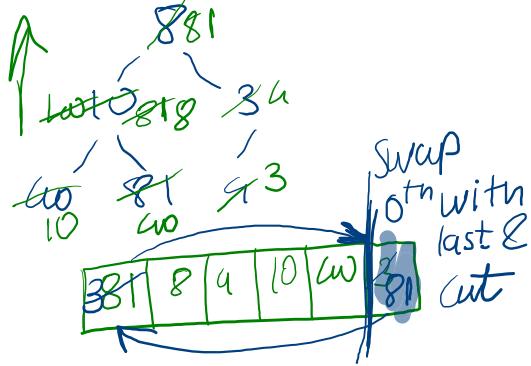
Heapify → Array → form a tree  
→ use Heap(m/m)



→ rearrange  
bottom to top  
left to right

# MaxHeap(Ascending)

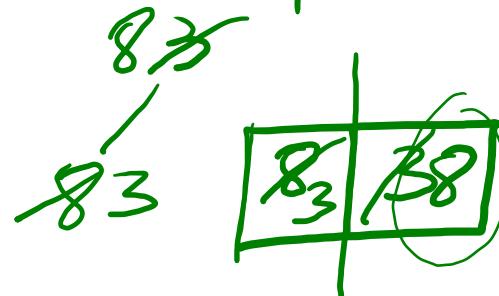
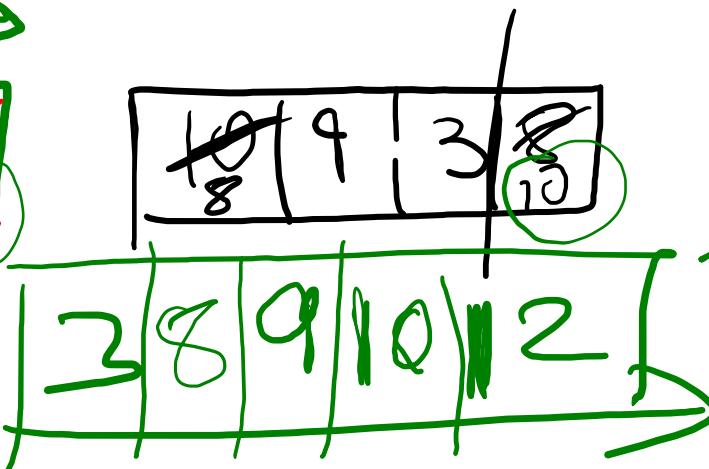
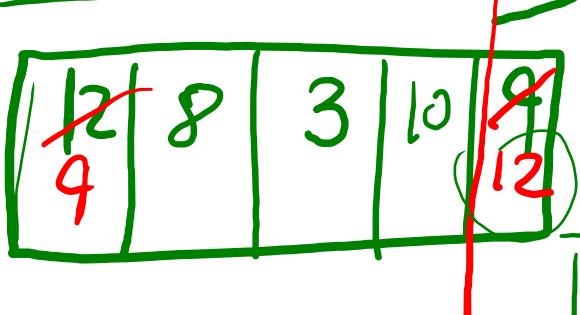
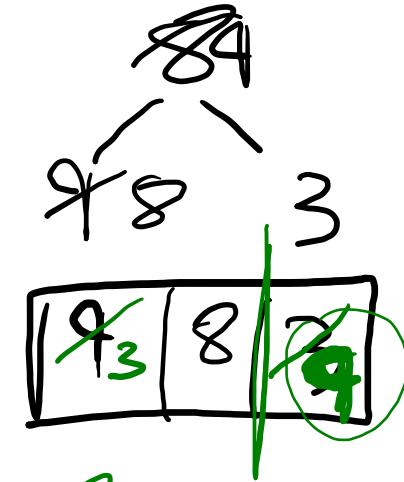
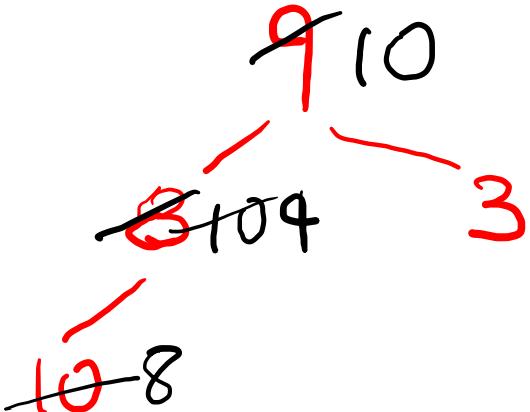
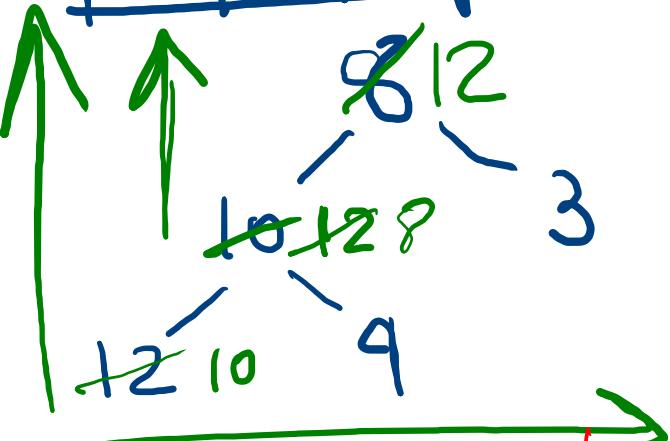
<i>t</i>	8	10	3	40	81	4
	0	1	2	3	4	5



8	10	3	12	9
---	----	---	----	---

9	8	3	10
---	---	---	----

8	9	3
---	---	---



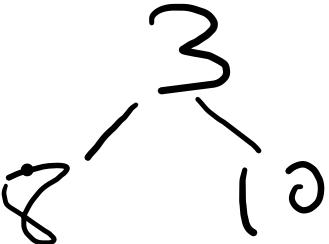
8	3	10	2
---	---	----	---

3	8	10
---	---	----

10	8
----	---

Min 82  
32810

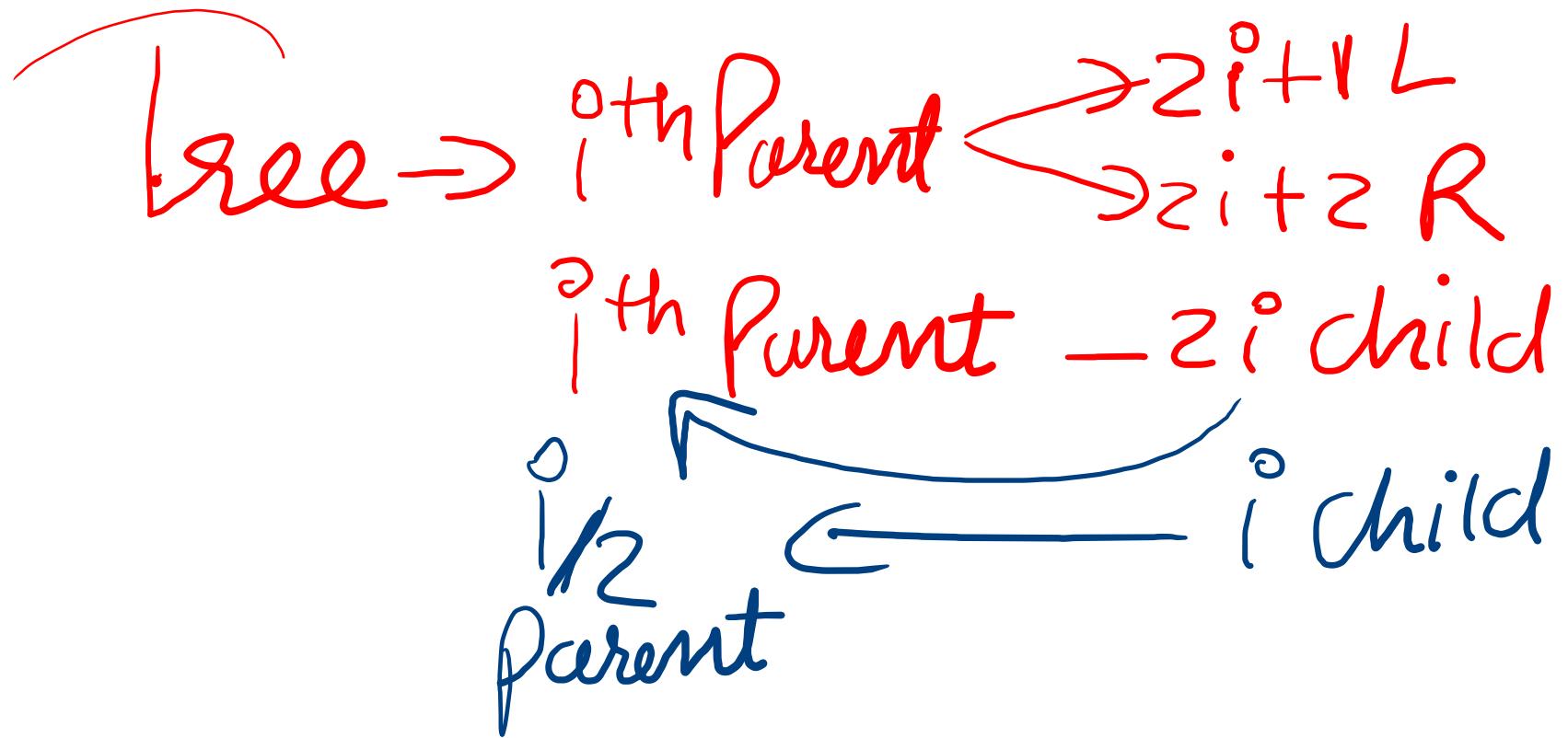
2	3	10	3	2
---	---	----	---	---

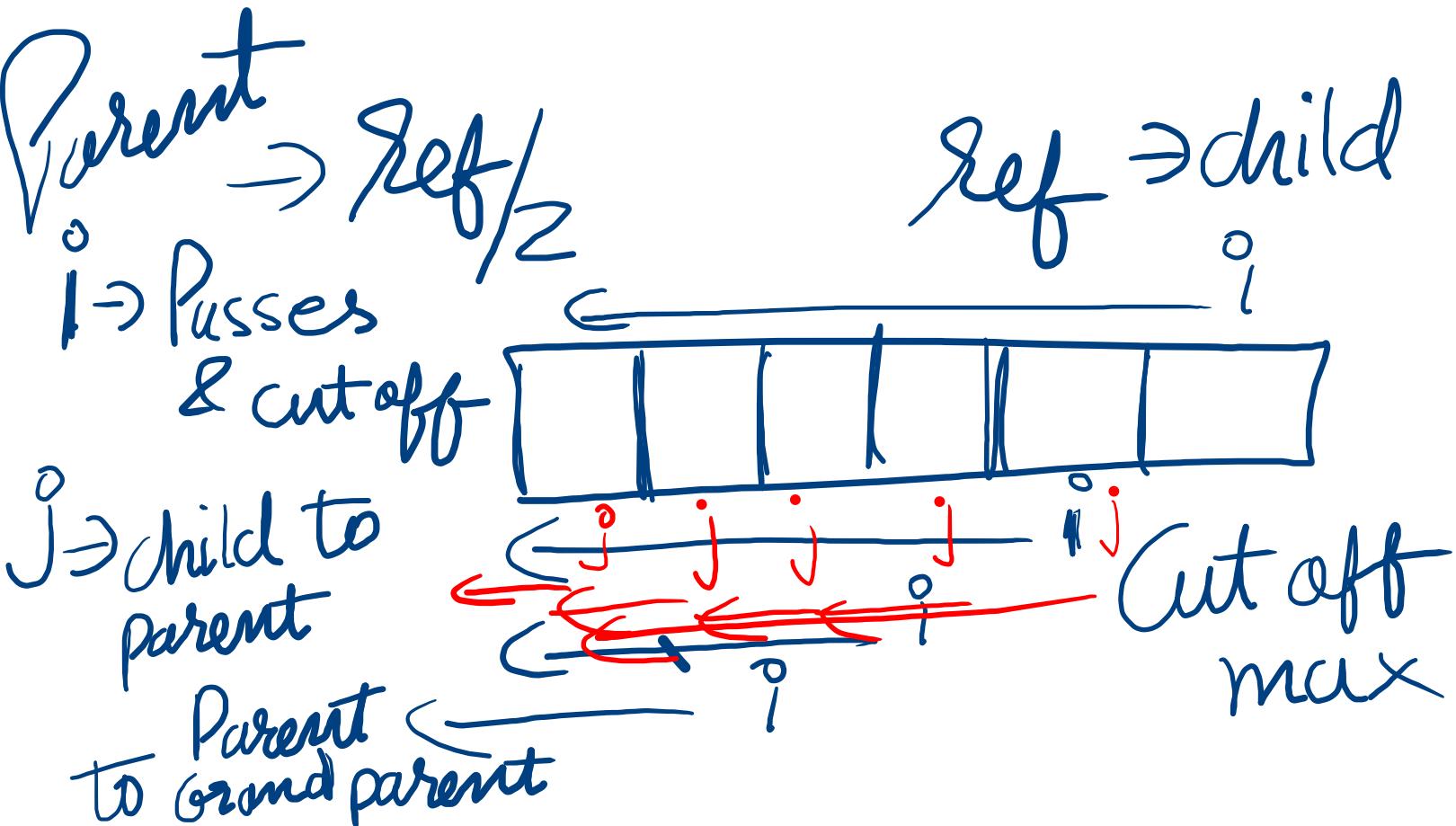


3	8	10	3.
---	---	----	----

8	10	8.
---	----	----

10	8	3	2
----	---	---	---





```

void heap()
{
int i,k,lp,temp,done=0;
for(i=a.length-1;i>=0;i--) // i → last → 1st
{
for(k=0;k<=i;k++) // K Child → Parent → Grand Parent
{
done=0;
lp=k; ← lp → parent(lp/2) child(lp)
while(lp>0 && done!=1)
{
if(a[lp]>a[lp/2])
{
temp=a[lp];
a[lp]=a[lp/2];
a[lp/2]=temp;
lp=lp/2;
}
else
done=1;
}
temp=a[0];
a[0]=a[i];
a[i]=temp;
}
}

```

$i=0$	$k=1$	$k=2$
10	13	8
12	12	6

$\nwarrow$  Start  
 $\nearrow$  Child → Parent → Grand Parent  
 $\downarrow$  lp/2 lp  
 $\nearrow$  lp → parent(lp/2) child(lp)

Compare child to Parent

if needed Swap Max Heap

move to next Parent  
for Grand Parent comp.

86 → 386 83  
lp/2      lp/2      lp

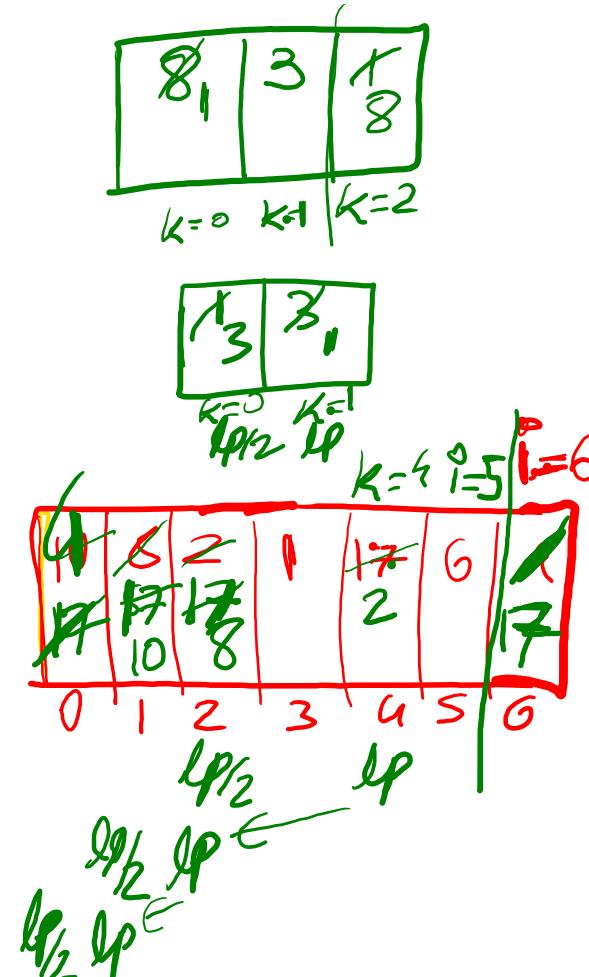
Swap  
0<sup>th</sup> with 0<sup>th</sup>  
& cut from i--

10 ← 7 ← 3  
lp/2      lp  
< lp/2 - lp done=1

```

void heap()
{
int i,k,lp,temp,done=0;
for(i=a.length-1;i>=0;i--)
{
for(k=0;k<=i;k++)
{
done=0;
lp=k;
while(lp>0 && done!=1)
{
if(a[lp]>a[lp/2])
{
temp=a[lp];
a[lp]=a[lp/2];
a[lp/2]=temp;
lp=lp/2;
}
else
done=1;
}
}
temp=a[0];
a[0]=a[i];
a[i]=temp;
}
}

```



Searching → Result  $i^{\text{th}}$  pos  
J → Response - 1

Searching

Sorted

Unsorted

Binary Search

Sequential

```
int sequentialsearch(int key)
```

```
{  
    //for(int i =0;a[i]==key || i<a.length;i++);
```

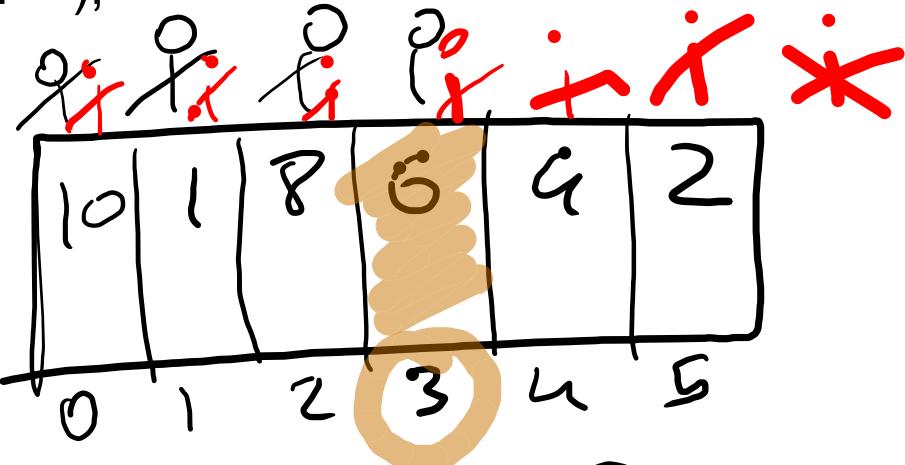
```
    for(int i=0;i<a.length;i++)
```

```
    {  
        if(key==a[i]) -  
            return(i);
```

```
}
```

```
return(-1);
```

```
}
```



SC<sup>(Worst)</sup>

SC<sup>(Best)</sup> Return 3  $\Rightarrow 3+1$   
index position

Search(20)

10	20	30	40	50	60	70
0	1	2	3	4	5	6

$$\text{mid} = \frac{0+6}{2} = 3$$

Search(100)

key < a[mid]      key > a[mid]

10	20	30
0	1	2

$$\frac{0+2}{2} = 1$$

50	60	70
a	s	6

$$\frac{4+6}{2} = 5$$

key = a[mid]



key > a[mid]

$\boxed{70} \neq 100$   
S    m    E    not found

```
void binarysearch(int start,int end,int key)
```

```
{
```

```
if(start<=end)
```

BS(0,4,50)

100

BS(3,4,50)

100

```
int mid=(start+end)/2;
```

BS(4,4,50)

100

```
if(a[mid]==key) X
```

```
System.out.println("found at:"+(mid+1));
```

```
if(key<a[mid])//left only
```

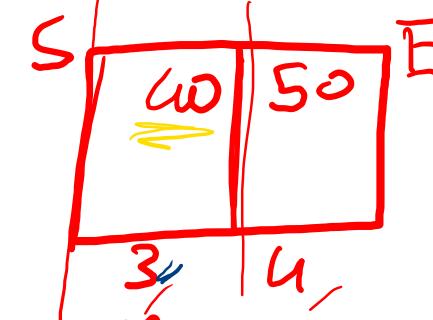
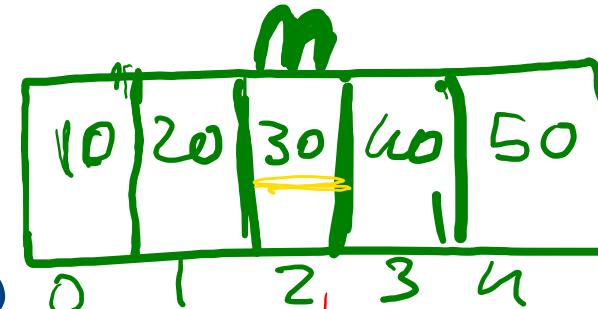
```
binarysearch(start,mid-1,key);
```

```
if(key>a[mid])//right only
```

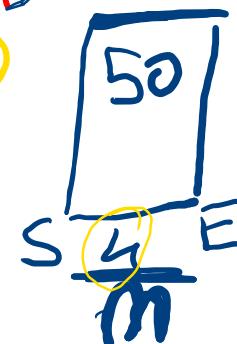
```
binarysearch(mid+1,end,key);
```

```
else
```

```
System.out.println("Not found");
```



BS(5,4,100)

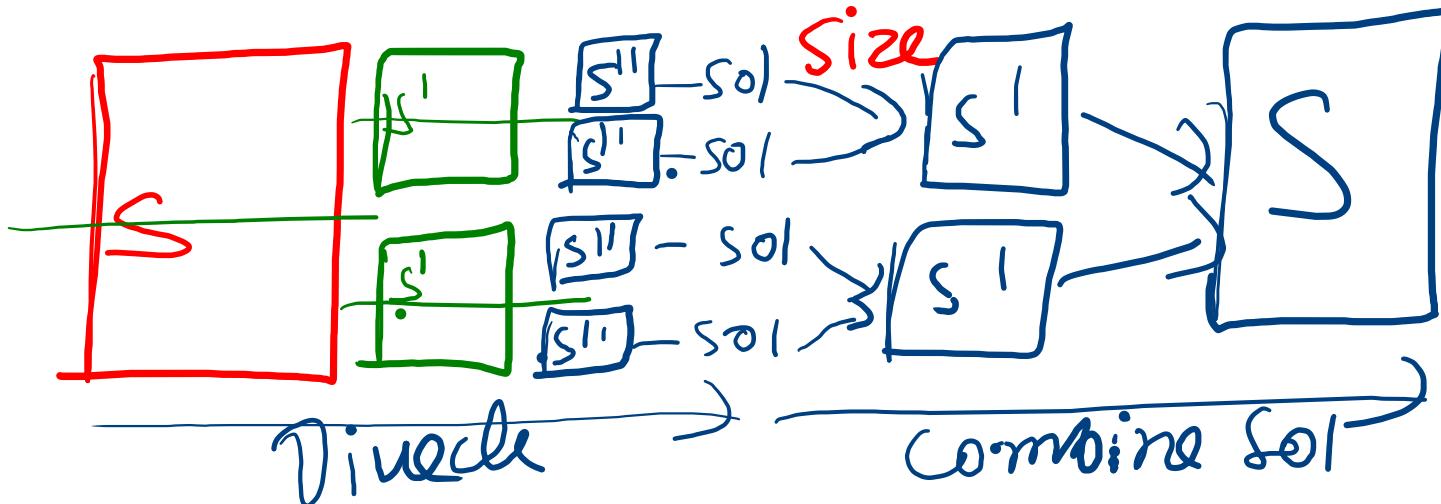


# Types of Algorithms

## 1 Divide and conquer

- Merge sort
- Binary search

→ Problem is too big  
then sub div till  
it reaches to handleable



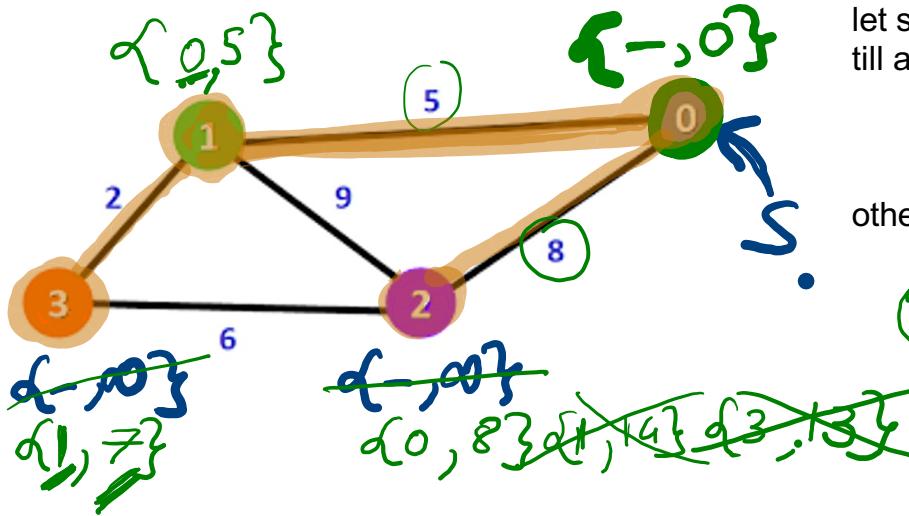
# Types of Algorithms

## 2 Greedy Algorithm

- Talks About Optimization
- Optimization
  - Minimization-Cost
  - Maximization-Profit

# Dijkstra's Shortest Path Algorithm

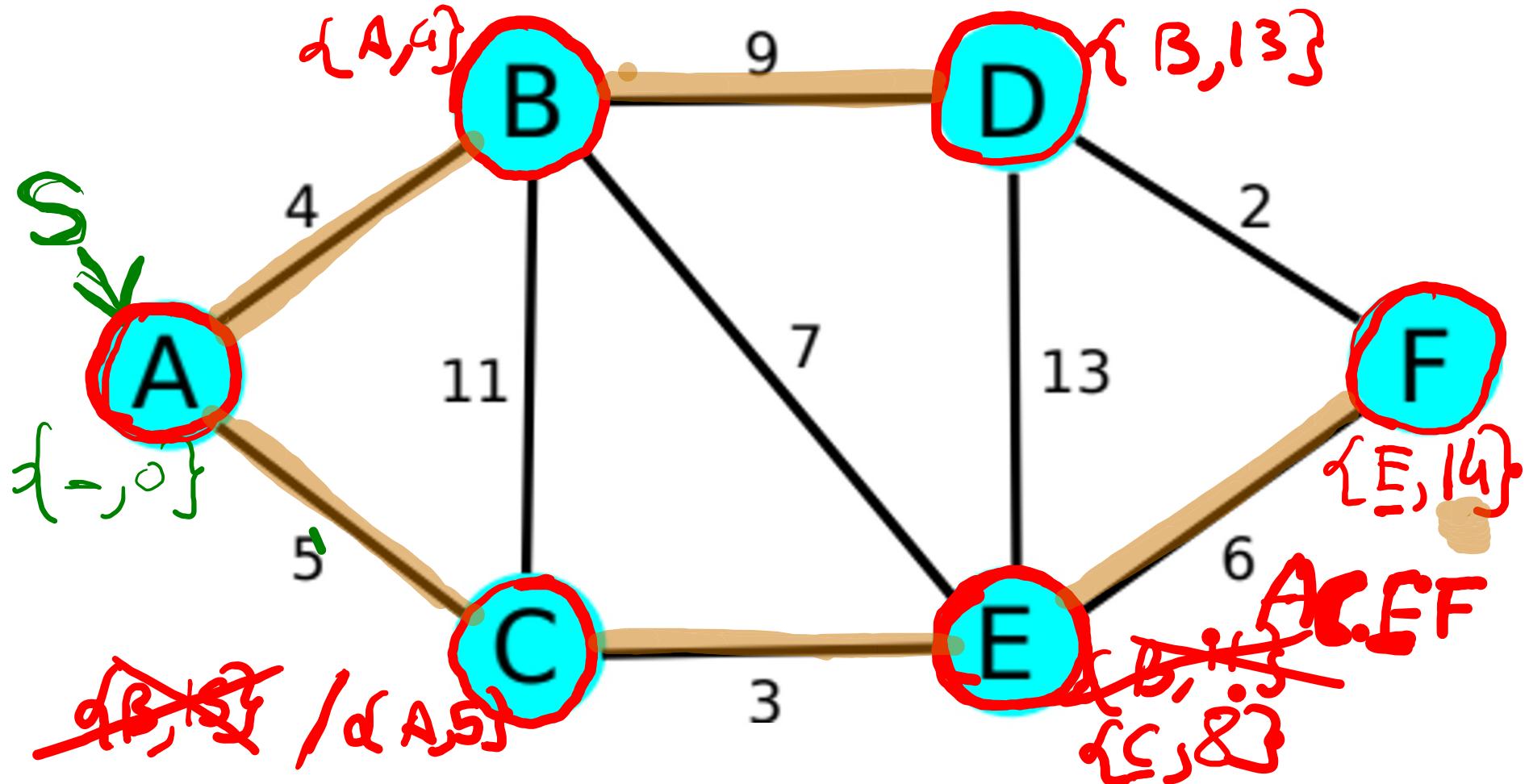
- Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
- While **sol** doesn't include all vertices
  - Pick a vertex **u** which is not there in **sol** and has a minimum distance value.
  - Include **u** to **sol**.
  - Then update distance value of all adjacent vertices of **u**.
    - To update the distance values, iterate through all adjacent vertices.
    - For every adjacent vertex **v**, if the sum of the distance value of **u** (from source) and weight of edge **u-v**, is less than the distance value of **v**, then update the distance value of **v**.

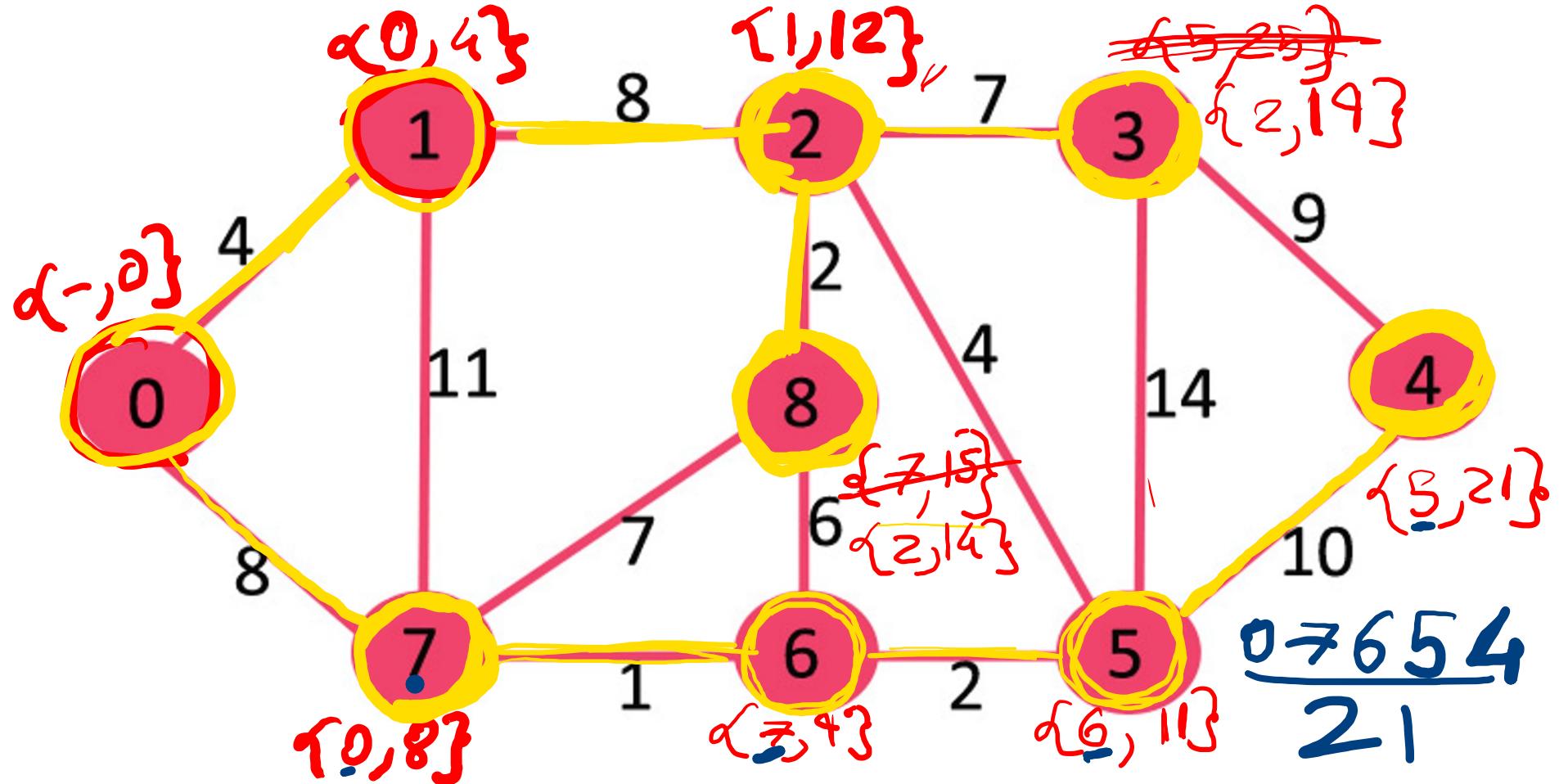


mark all  $\{-\infty\}$   
let source be  $\{0\}$   
till all are not visited  
1. search for smallest path  
2. mark it visited  
3. forward weight to all  
others including travelling cost

$$0 - 3 = 7$$

$$0 \rightarrow 1 \rightarrow 3$$



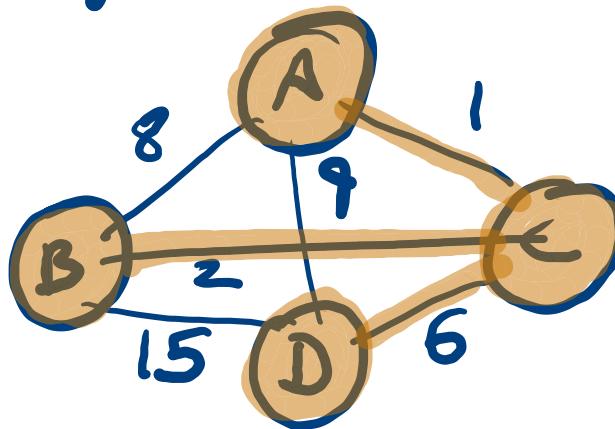


# Minimum Spanning Tree App of Graph

- Given an undirected and connected graph , a spanning tree of the graph is a tree that spans (that is, it includes every vertex of ) and is a subgraph of (every edge in the tree belongs to )with minimum cost.

$$\frac{1+6+2}{9 \text{ cost.}}$$

I/P  $\rightarrow$  Graph  
O/P  $\rightarrow$  Tree



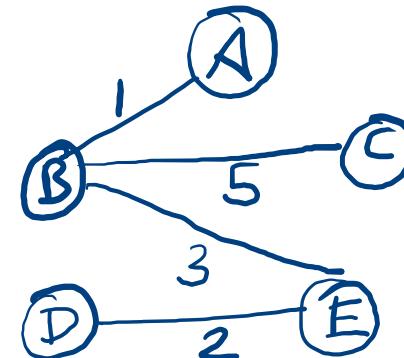
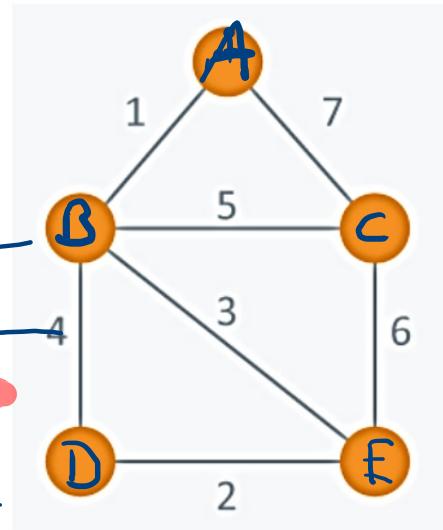
# Kruskal's Algorithm

- **Algorithm Steps:**
- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

Edge based

- (A,B,1)  
(D,E,2)  
(B,E,3)  
~~(B,D,4)~~  
(B,C,5)  
(C,E,6)  
(A,C,7)

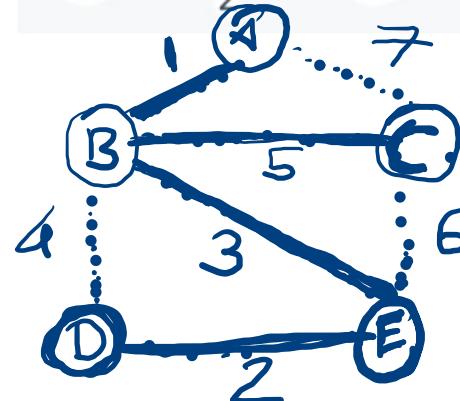
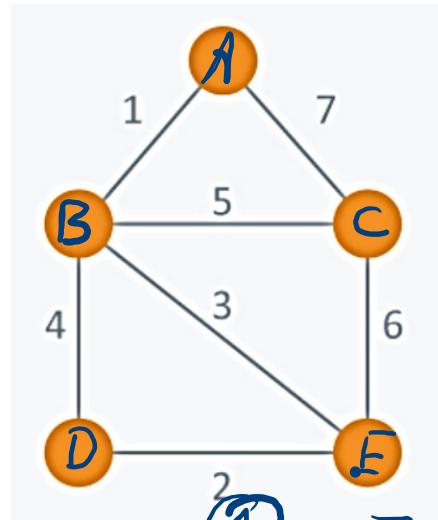
MST



# Prim's Algorithm

- **Algorithm Steps:**
- Start from given source add it to sol tree.
- While all vertices are not in sol tree do
  - Select the cheapest edge between vertex of sol tree and non-sol vertex iff not forming cycle.

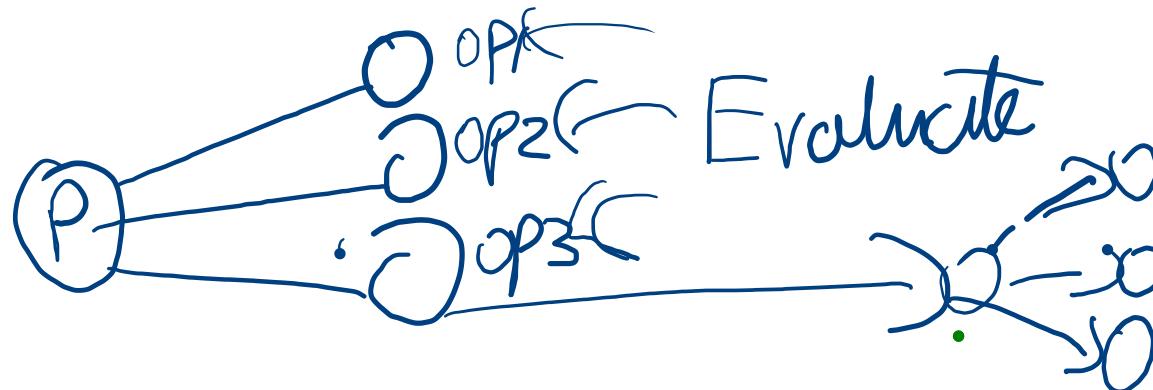
*node base  
Better Control*

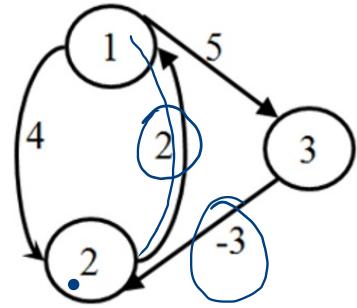


# Types of Algorithms

## 3 Dynamic Programming

- Break problem in states, evaluate each state with formula recursively, select best in each step so finally get optimized answer.
- Example: All source shortest path(Floyd warshal's)





$$W = D^0 = \begin{array}{|c|c|c|} \hline & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 5 \\ \hline 2 & 2 & 0 & \infty \\ \hline 3 & \infty & -3 & 0 \\ \hline \end{array}$$

$$P = \begin{array}{|c|c|c|} \hline & 1 & 2 & 3 \\ \hline 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 0 & 1 \\ \hline 3 & 0 & 0 & 0 \\ \hline \end{array}$$

$D^1$

0	4	5
2	0	7
$\infty$	-3	0

$D^2$

0	4	5
2	0	7
$\infty$	-3	0

Begin

for  $k := 0$  to  $n$ , do

    for  $i := 0$  to  $n$ , do

        for  $j := 0$  to  $n$ , do

            if  $\text{cost}[i,k] + \text{cost}[k,j] < \text{cost}[i,j]$ , then

$\text{cost}[i,j] := \text{cost}[i,k] + \text{cost}[k,j]$

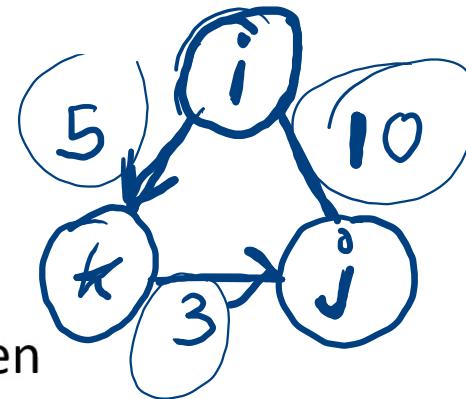
            done

        done

    done

display the current cost matrix

End

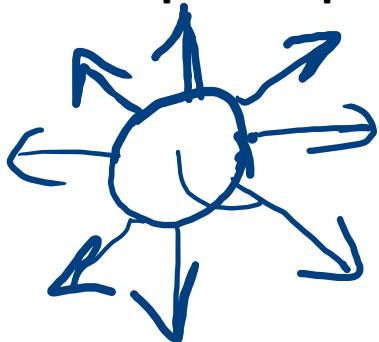


$$\frac{\text{Cost}(i,j) > \text{Cost}(i,k) + \text{Cost}(k,j)}{10 > 5 + 3}$$

# Types of Algorithms

## 4 Back Tracking

- Only solution needed . (May/May not be optimal)
- N-queens problem  $\rightarrow 4 \times 4 \rightarrow$  4 Queens On 4x4 Board such that no one attacks other



$\alpha_1$	$Q_1$			
$\alpha_2$	$Q_2$	$Q_2$	$Q_2$	$Q_2$
$\alpha_3$	$Q_3$	$Q_3$	$Q_3$	$Q_3$

$\leftarrow B.T$

$\leftarrow Q_3$

$\leftarrow B.T$

$\leftarrow Q_4$

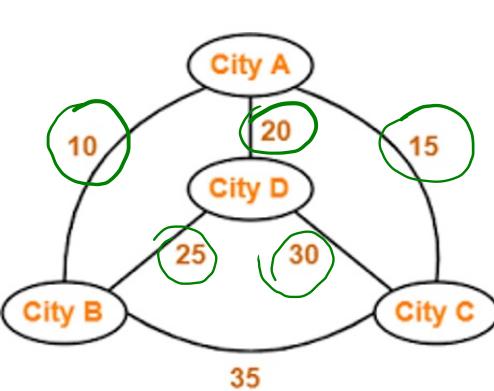
$\alpha_1$	$Q_1$			
$\alpha_2$		$Q_2$	$Q_2$	$Q_2$
$\alpha_3$	$Q_3$	$Q_3$	$Q_3$	$Q_3$
$\alpha_4$	$Q_4$	$Q_4$	$Q_4$	$Q_4$

	$Q_1$		
$\alpha_2$	$Q_2$	$Q_2$	$Q_2$
$\alpha_3$	$Q_3$		
$\alpha_4$	$Q_4$	$Q_4$	$Q_4$

# Types of Algorithms

## 5 Branch n Bound

- All solutions
- State Space tree(decision tree)
- Example: Traveling salesman



Travelling Salesman Problem

Start & end  
on same city  
with min. cost  
of travel

