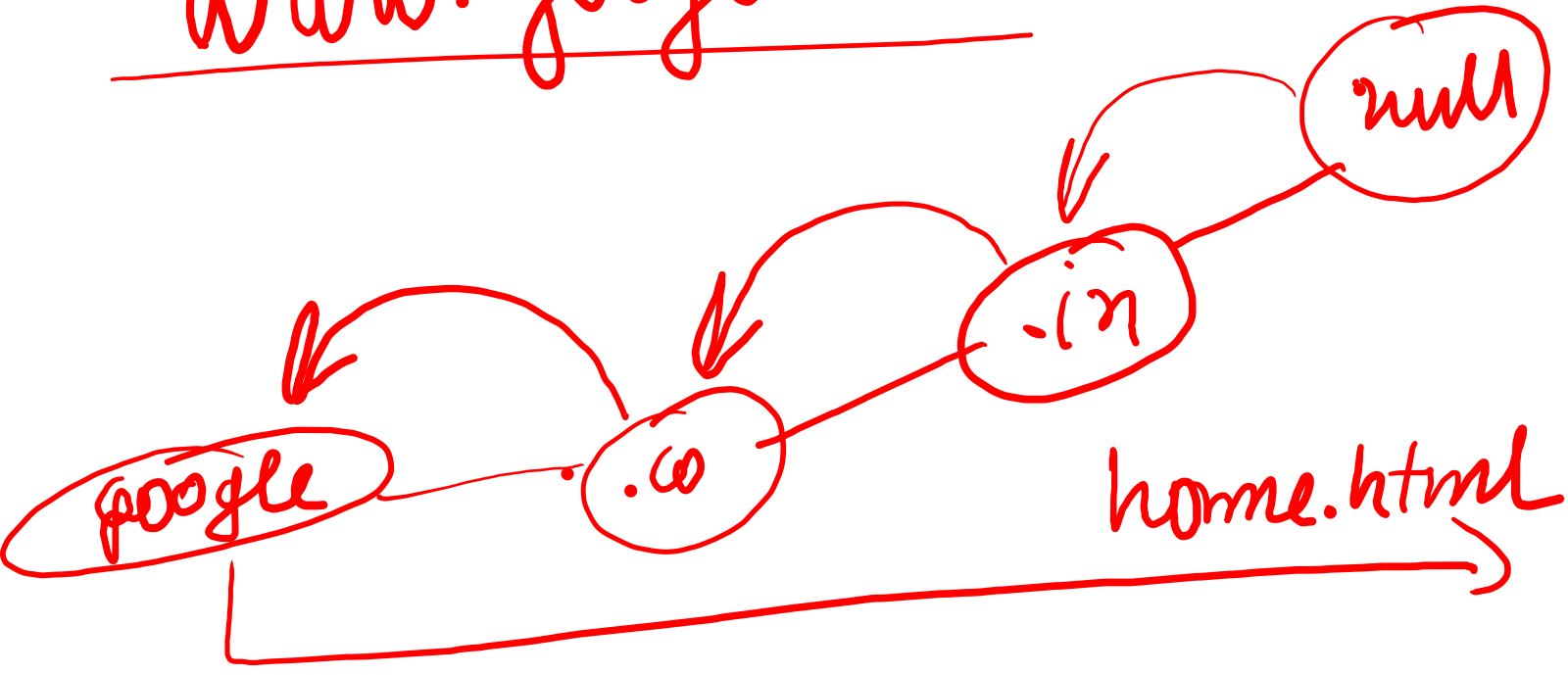
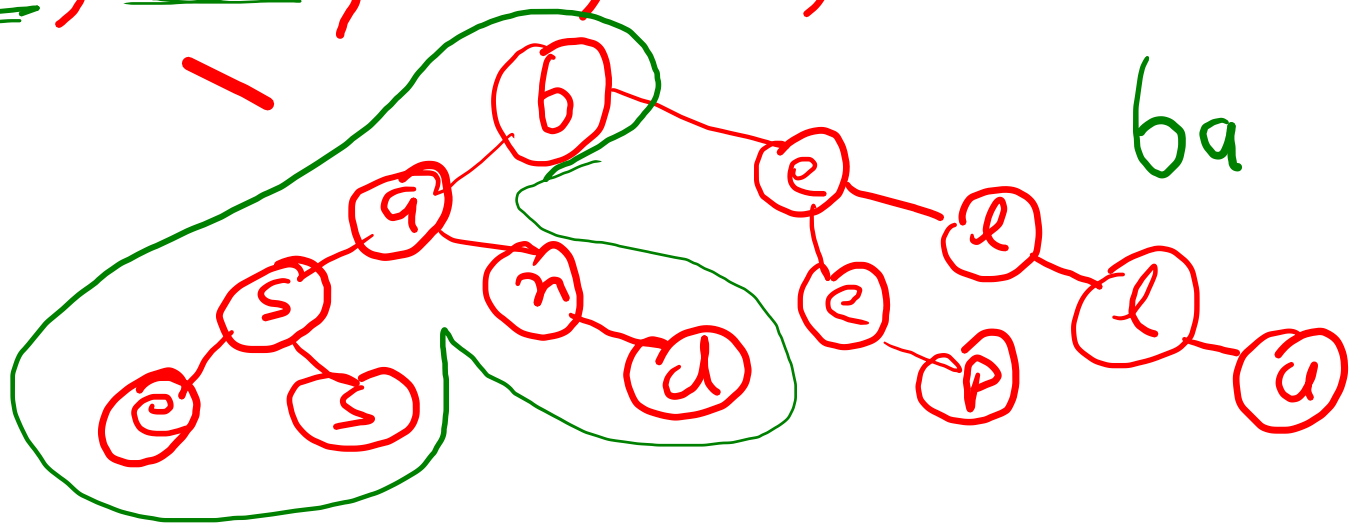
www.google.co.in



.net
.edu
.org
.us
.io
.uk

Tries → Tree → Autocorrect
Auto complete

base, band, bella, bee, beep, bass



ba

Heap Sort ⇒ Tree Based Sort

Heap → Min

Parent < child

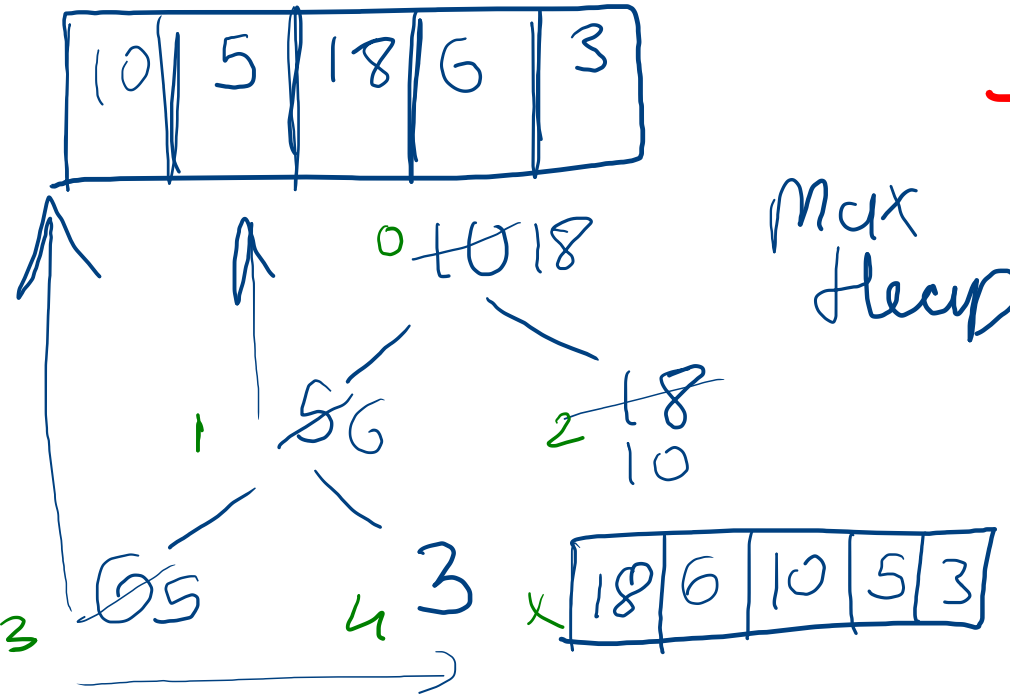
↓

Max

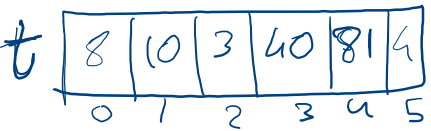
Parent > child

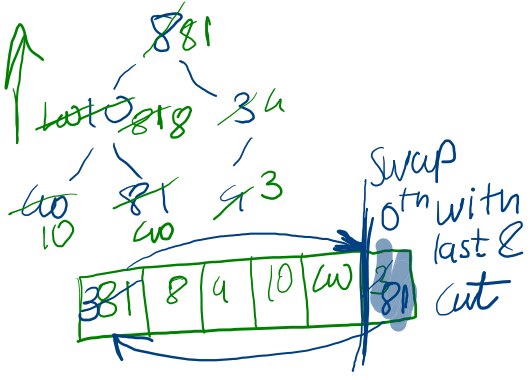
Heapify → Array ⇒ form a tree
⇒ use Heap(M/m)

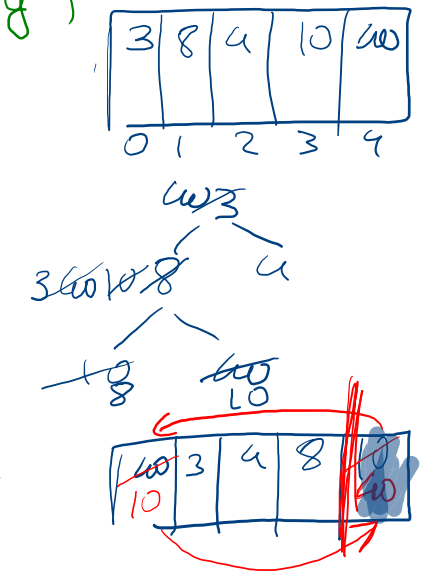
⇒ rearrange bottom to top
left to right

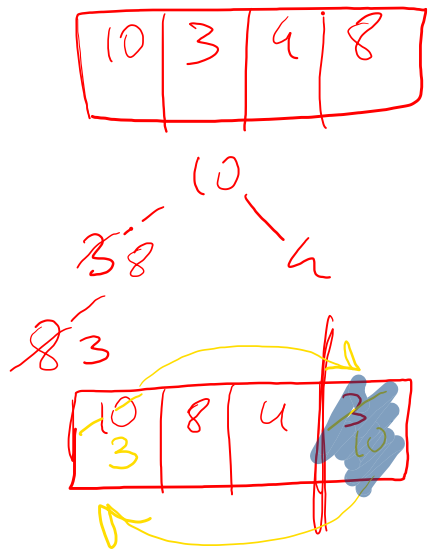


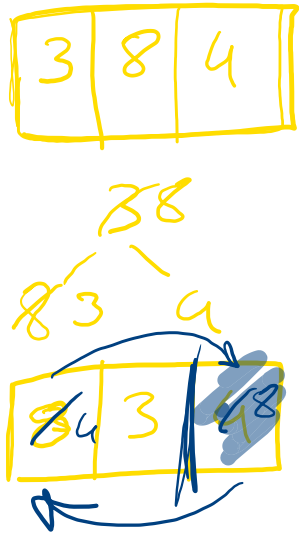
MaxHeap (Ascending)

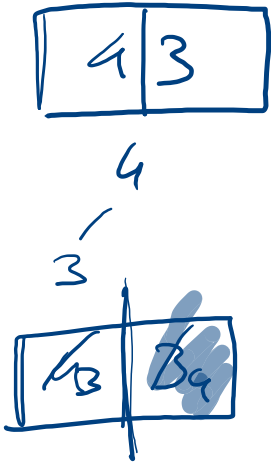


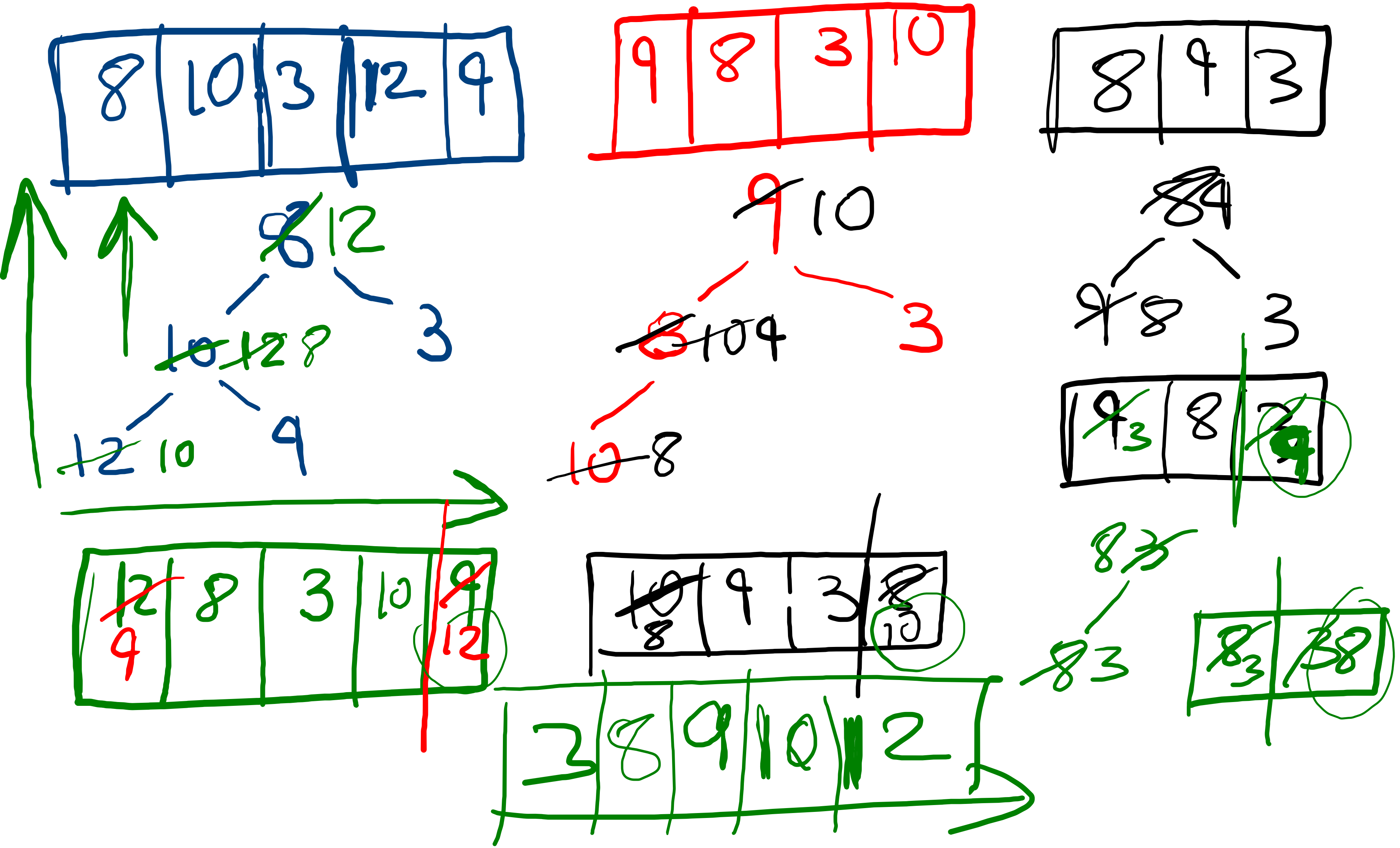


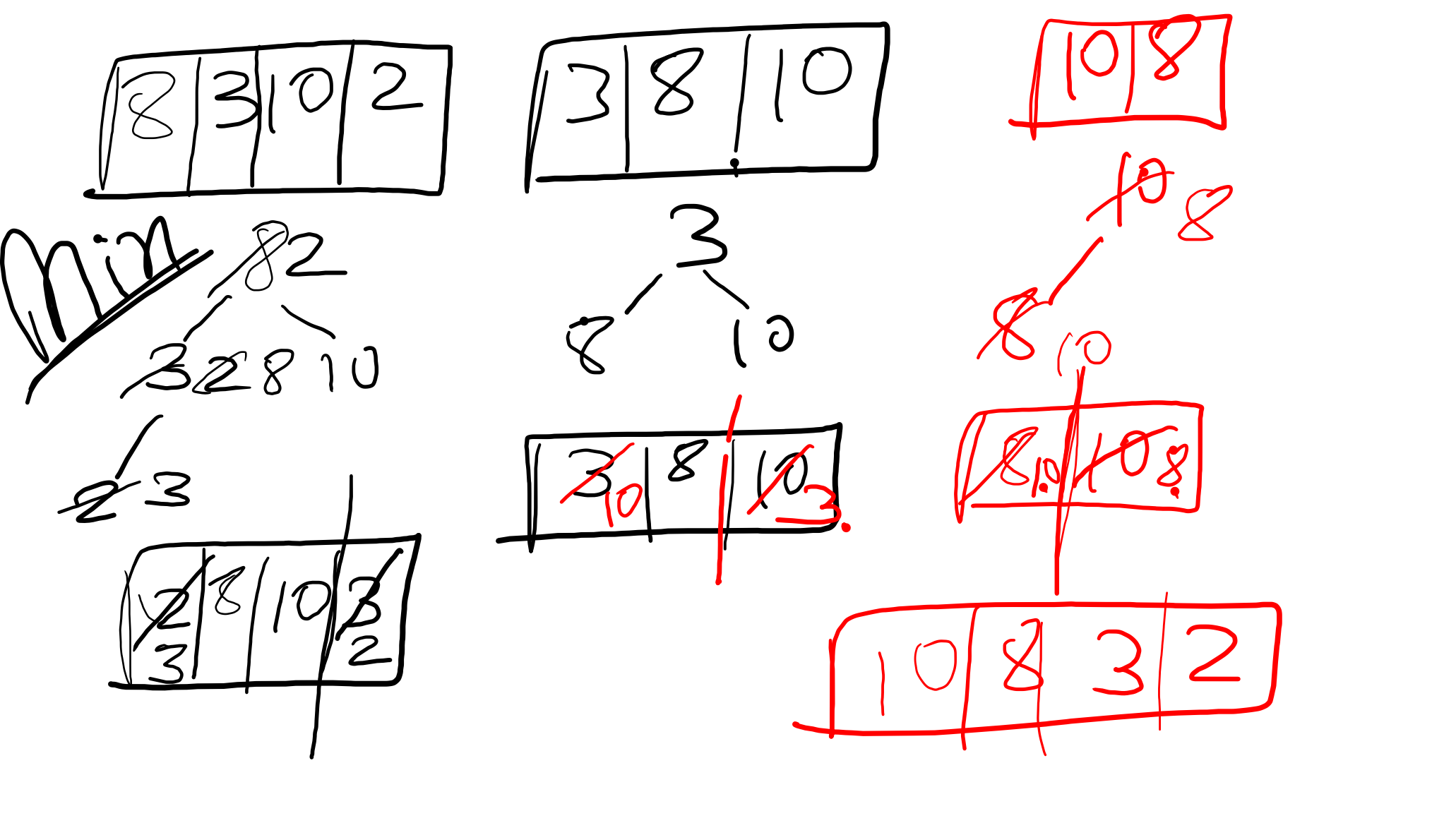


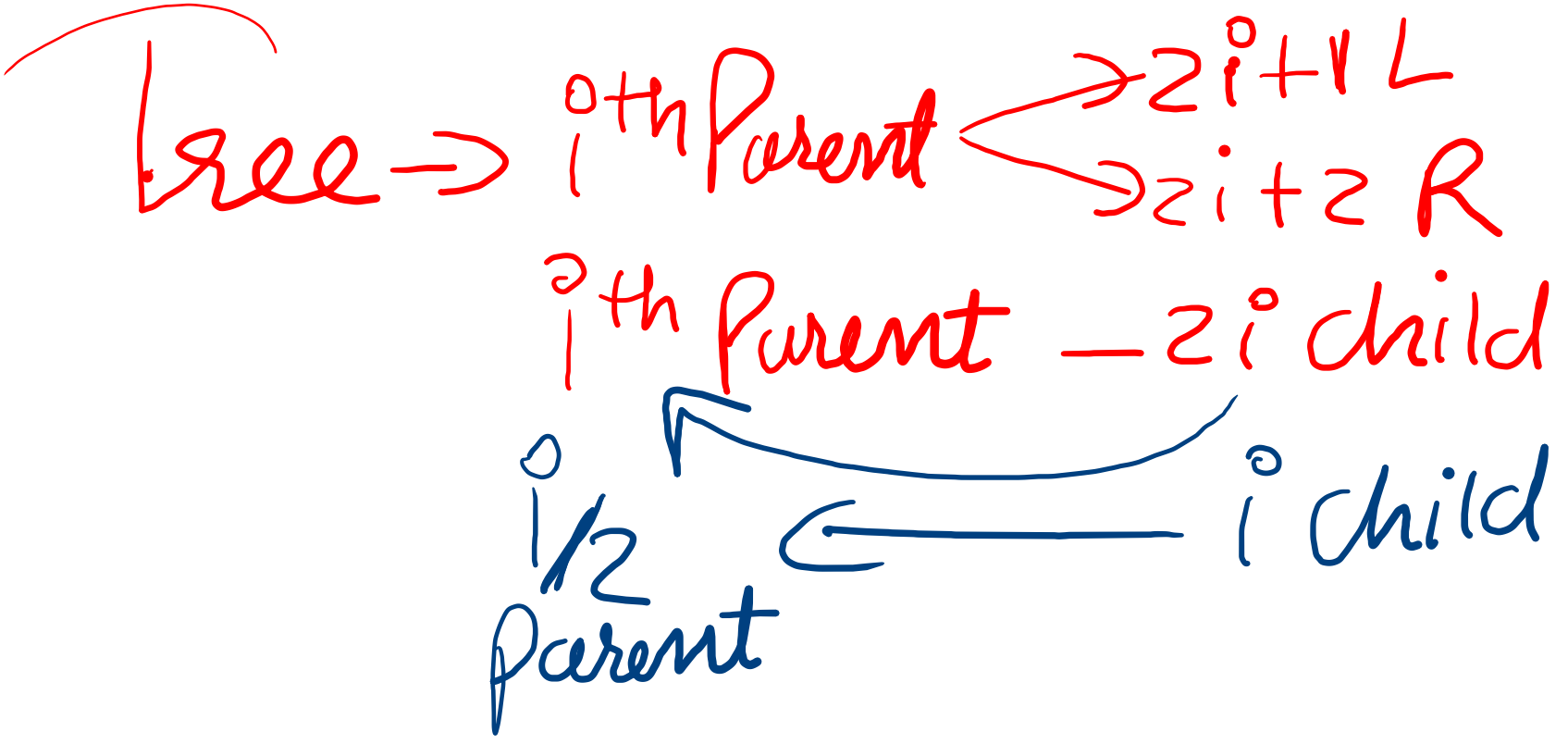


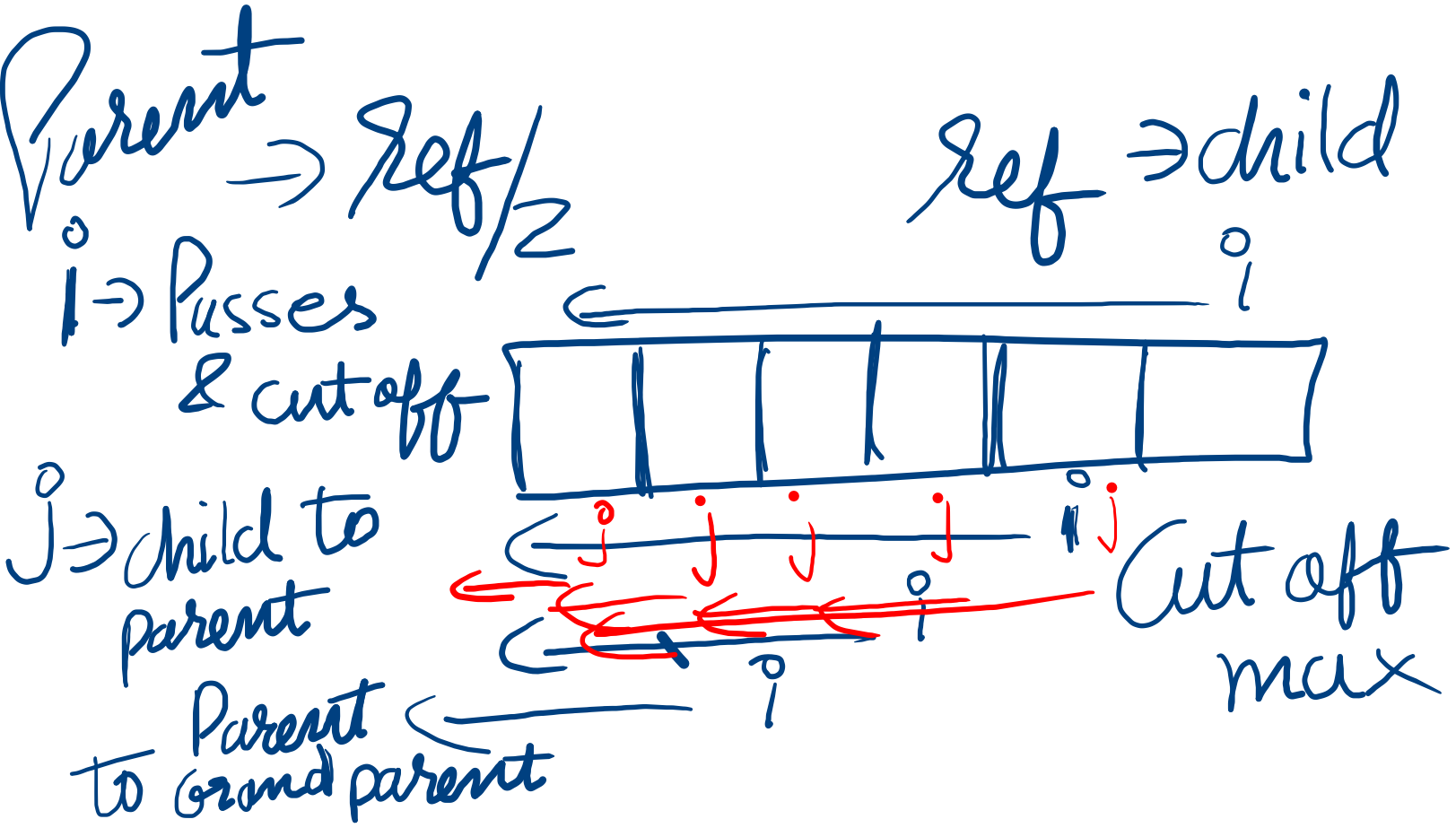





8 10 3 12 9
9 8 3 10
8 9 3
12 8 3 10 9
3 8 9 10 12


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


Tree → ith Parent
2i+1 L
2i+2 R
ith Parent — 2i child
i/2 Parent
i child


Parent
Ref/2
Ref ⇒ child
i ⇒ Passes & cut off
j ⇒ child to parent
Cut off max
Parent to Grand parent

```
 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;
}
}
```

i → last → 1st

Child → Parent → Grand Parent

start

lp/2 lp

lp → parent (lp/2) child (lp)

Compare child to Parent

MaxHeap

if needed swap

move to next Parent for Grand Parent comp.

8 6 — 8 6 8 3

lp/2 lp/2 lp

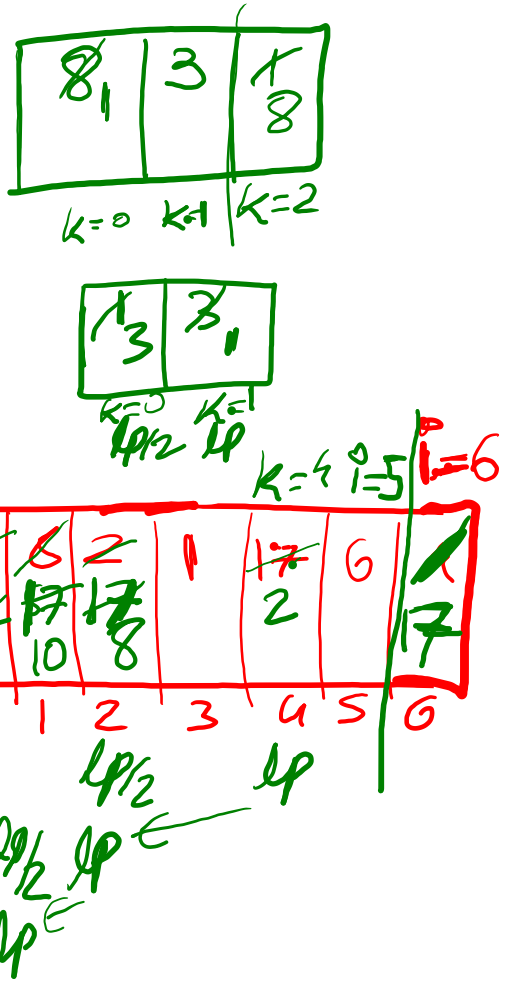
10 ← 7 ← 3

lp/2 lp

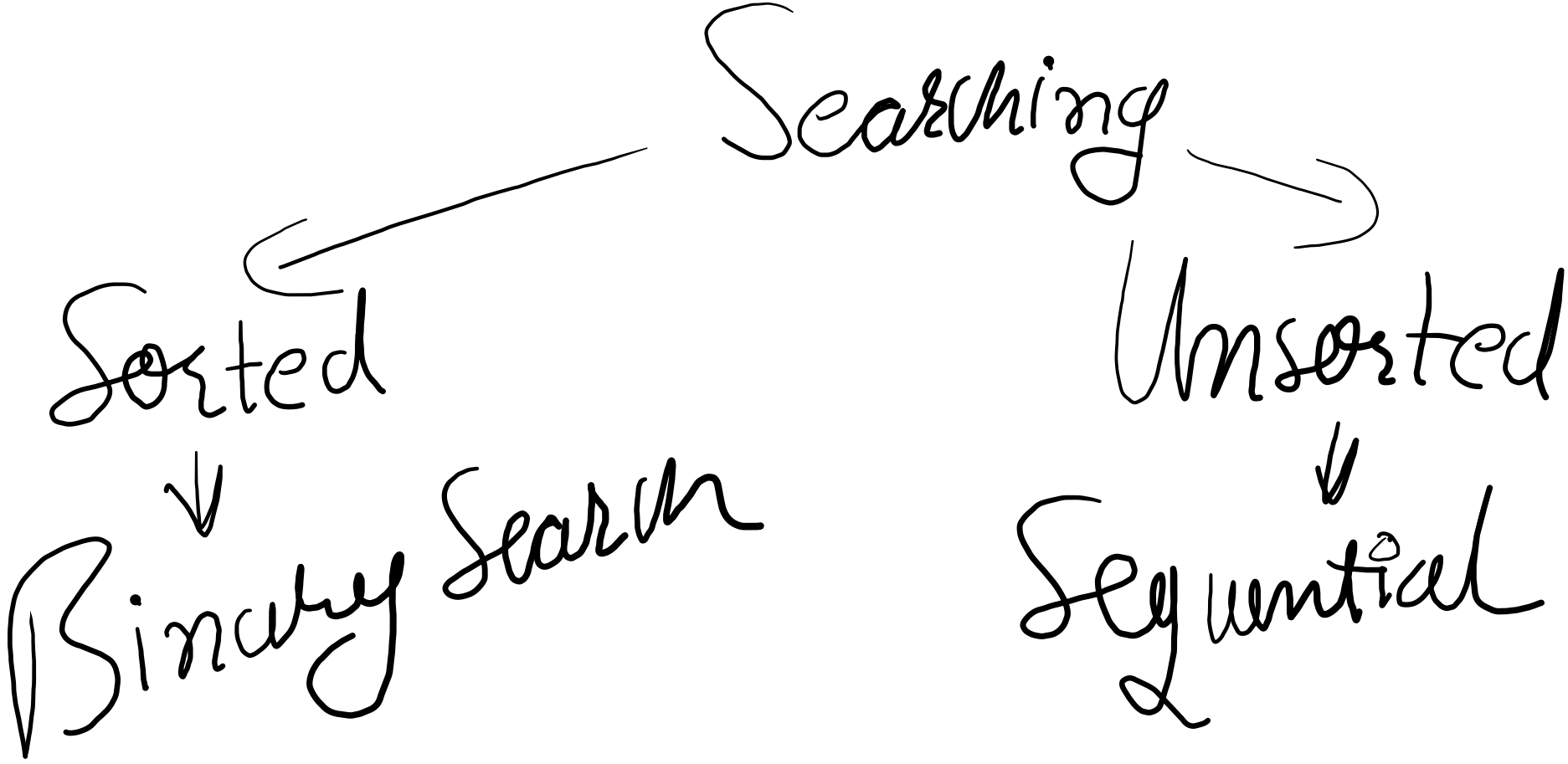
lp/2 lp done=1

swap 0th with ith & cut from i--

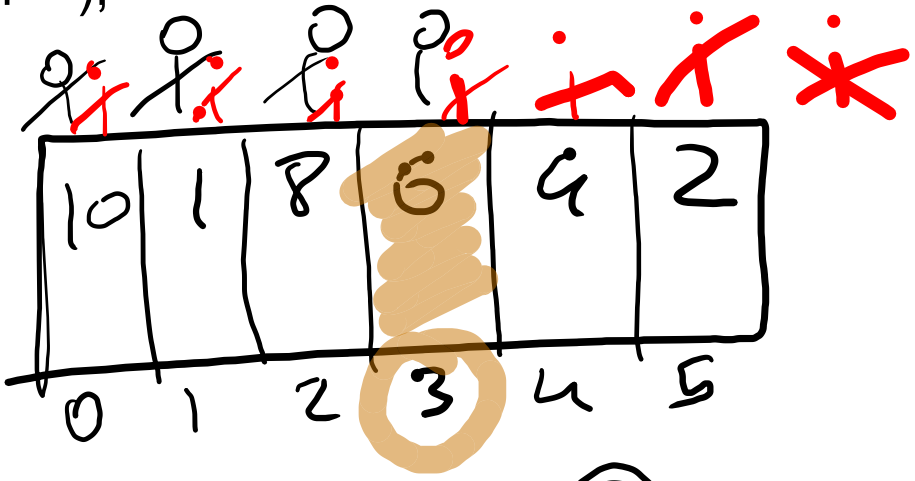
```
 void heap()
{
int i,k,lp,temp,done=0;
for(i=a.length-1;i>=0;i--)
{
for(k=0;k<=i;k++)
{
done=0;
lp=k;
while(lp>0 && done!=1)
{
if(a[lp]>a[lp/2])
{
temp=a[lp];
a[lp]=a[lp/2];
a[lp/2]=temp;
lp=lp/2;
}
else
done=1;
}
}
temp=a[0];
a[0]=a[i];
a[i]=temp;
}
}
```



Searching → Result in pos
→ Response -1


Searching
Sorted
Binary Search
Unsorted
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);
}
```



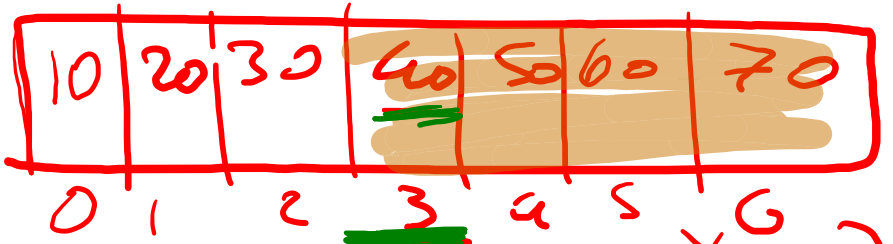
S(400)

S(6) return 3 ⇒ 3+1

index position

Search(20)

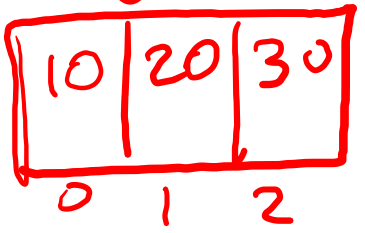
Search(100)



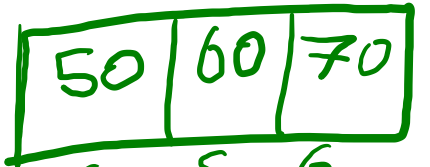
$mid = \frac{0+6}{2} = 3$

key < a[mid]

key > a[mid]



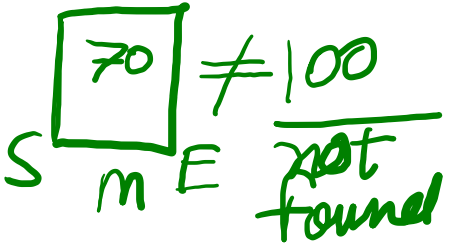
$\frac{0+2}{2} = 1$



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

key = a[mid]

key > a[mid]



$\neq 100$

~~not~~ found

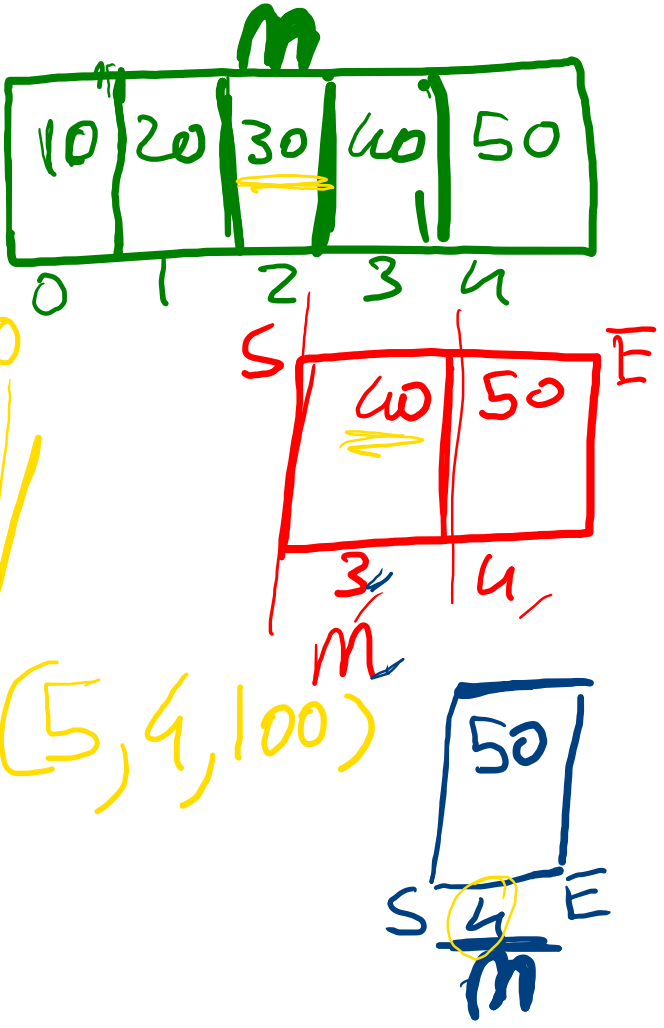
```
void binarysearch(int start,int end,int key)
 {
   if(start<=end)
   {
     int mid=(start+end)/2;
     if(a[mid]==key)
       System.out.println("found at:"+(mid+1));

     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(0,4,50)

BS(3,4,50)

BS(4,4,50)

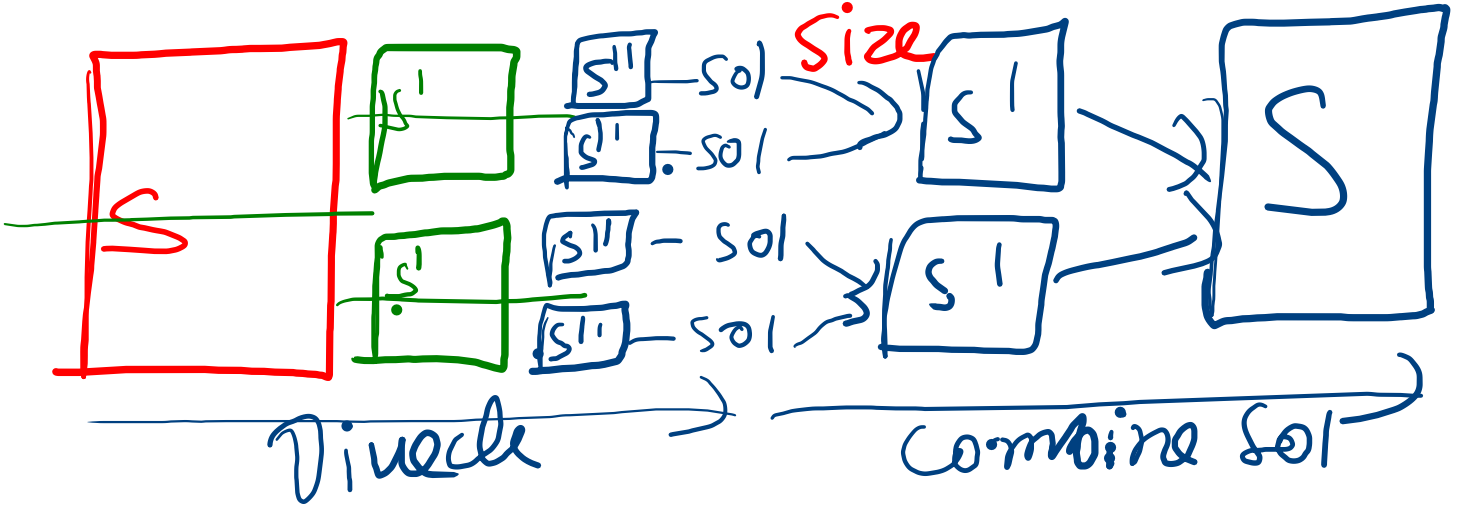
BS(5,4,100)



Types of Algorithms

1 Divide and conquer → Problem is too big then sub div till it reaches to handlable size

- **Merge sort**
- **Binary search**



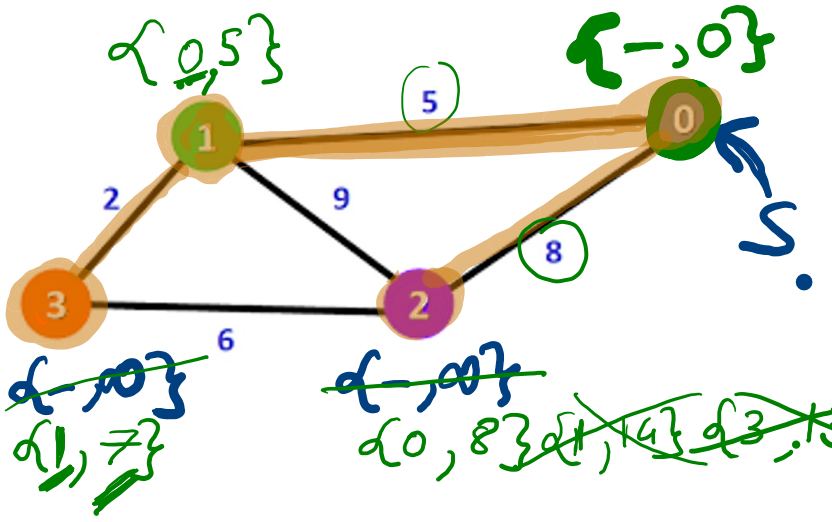
Types of Algorithms

2 Greedy Algorithm

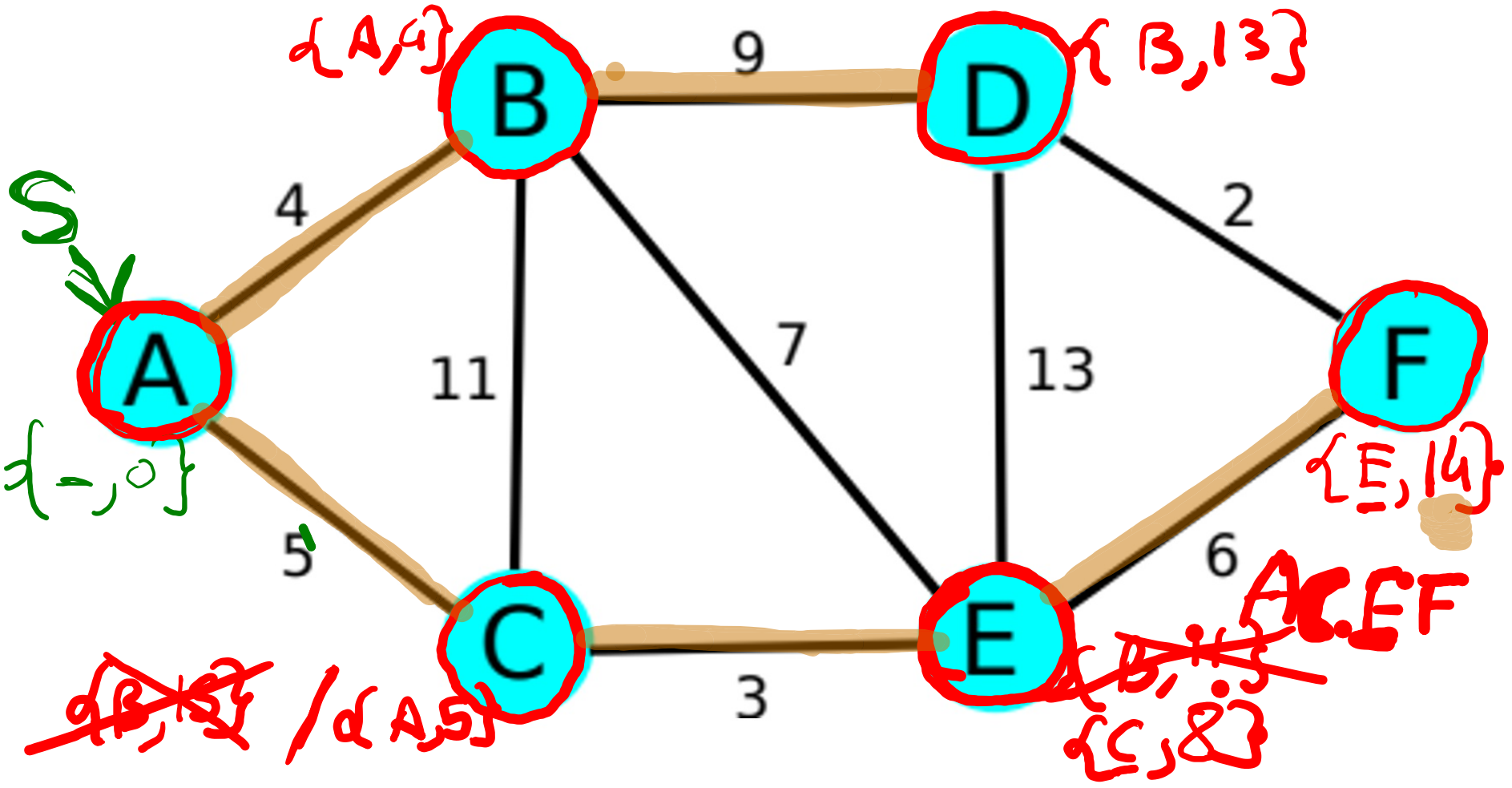
- **Talks About Optimization**
- **Optimization**
  - **Minimization-Cost**
  - **Maximization-Profit**

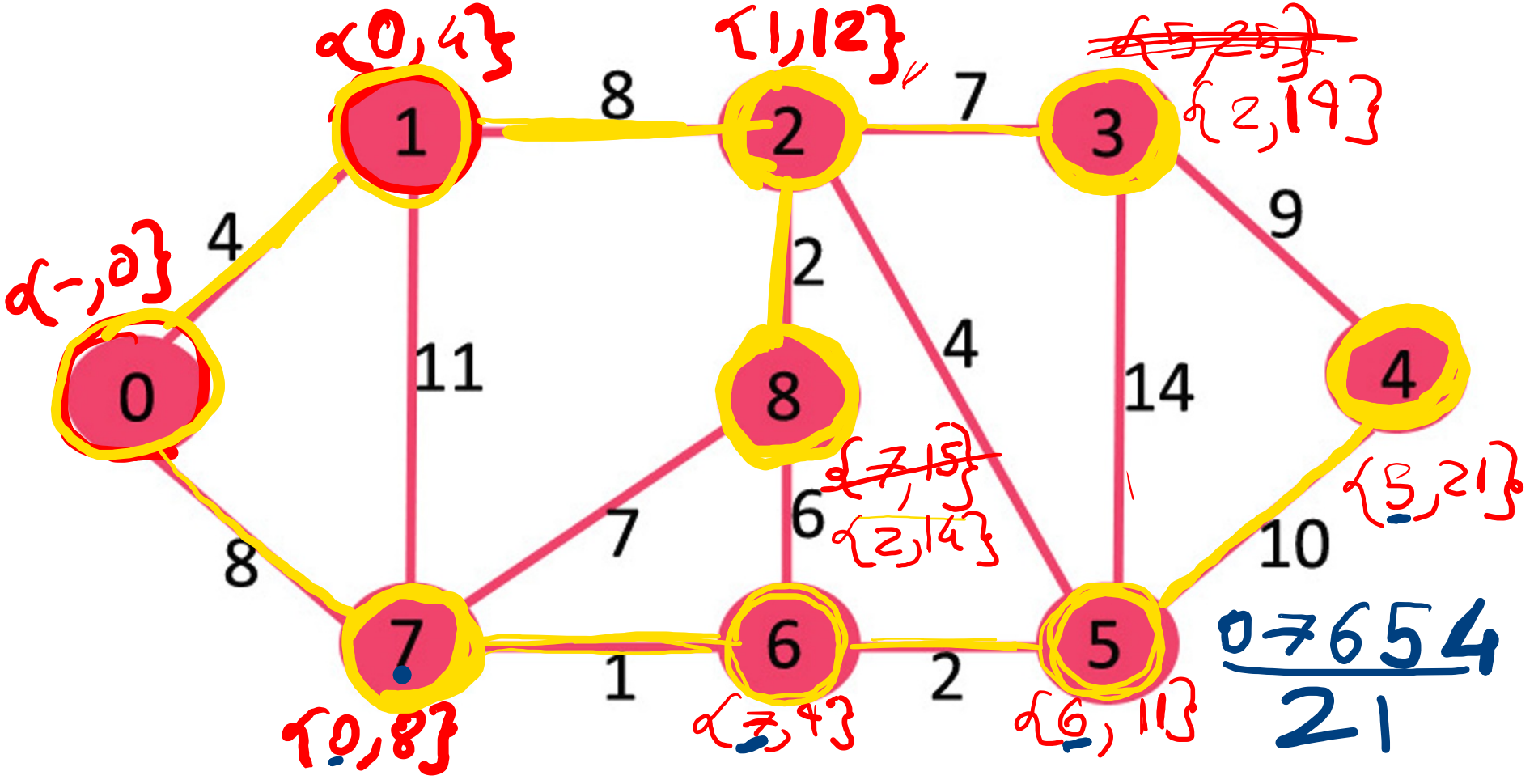
Dijkstra's Shortest Path Algorithm

- 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 {-,inf}
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


A
B
C
D
E
F
4
9
11
7
13
2
5
3
6
S
{-,0}
{A,4}
{B,13}
{E,14}
ACEF


{0,4}
{1,12}
{2,19}
{-,0}
0
1
2
3
4
5
6
7
8
4
8
11
8
2
7
4
14
9
6
7
1
2
10
{7,15}
{2,14}
{5,21}
{0,8}
{7,9}
{6,11}
0 7 6 5 4
21

Minimum Spanning Tree

App of Graph

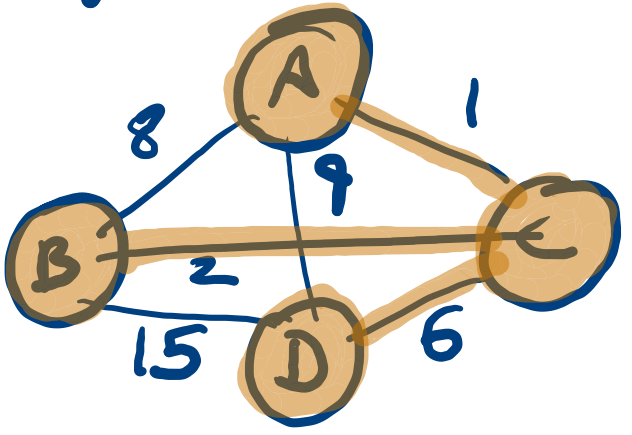
I/P ⇒ Graph

O/P ⇒ Tree

- 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 minimumcost.

1+6+2

9 cr.



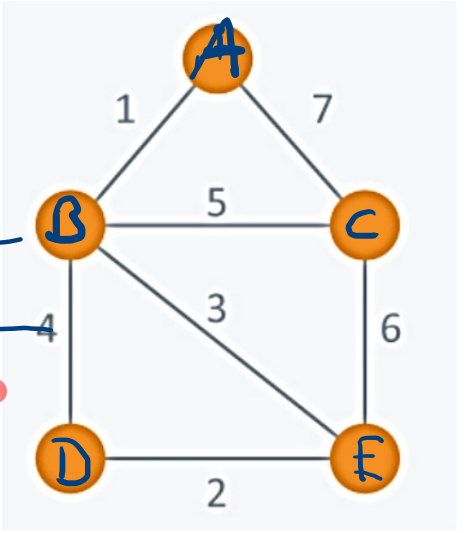
Kruskal's Algorithm

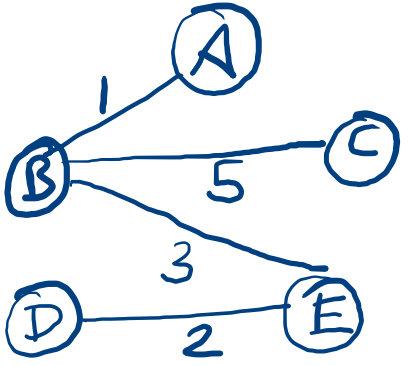
- **Algorithm Steps:**
- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

Edge based

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

MST

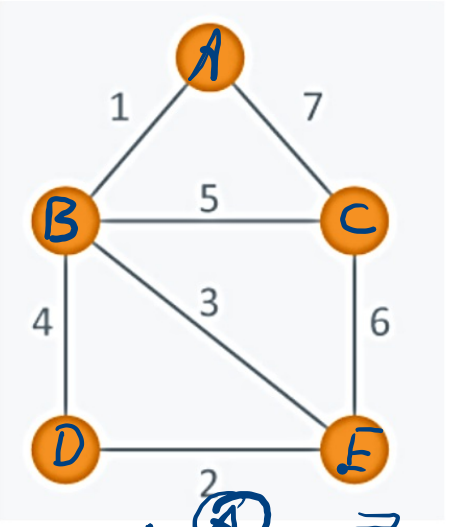




Prim's Algorithm

- **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.

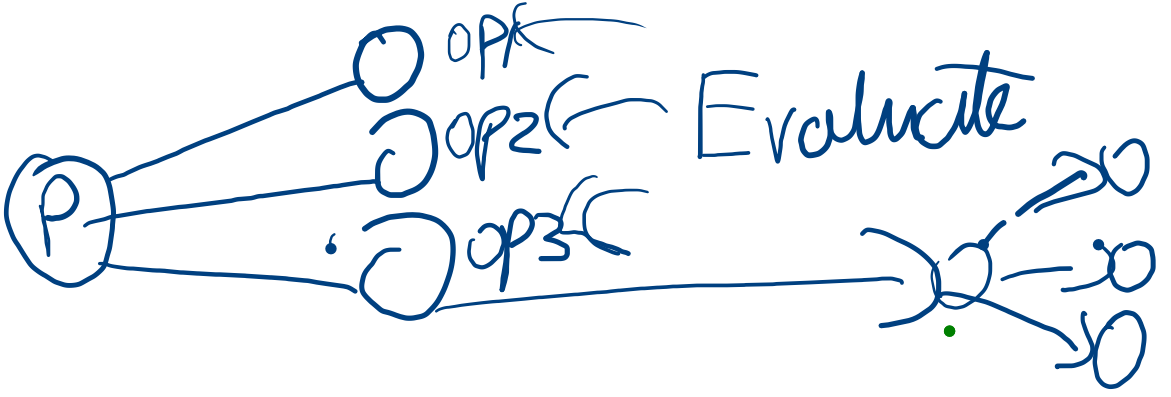
node base
Better control

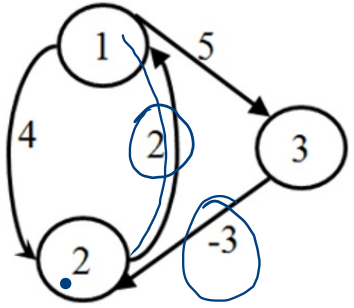


Types of Algorithms

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)**



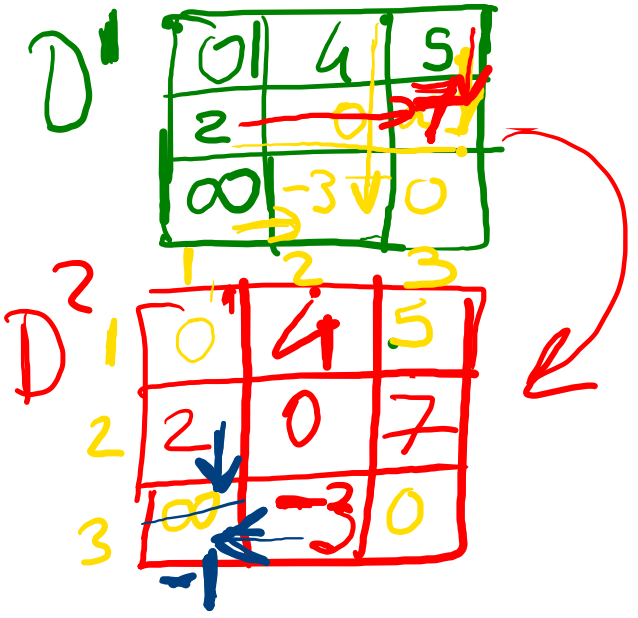


$W = D^0 =$

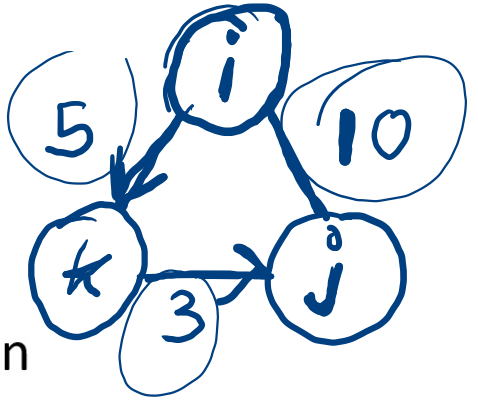
	1	2	3
1	0	4	5
2	2	0	∞
3	∞	-3	0

$P =$

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



```
Begin
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
        done
      done
    done
    display the current cost matrix
End
```



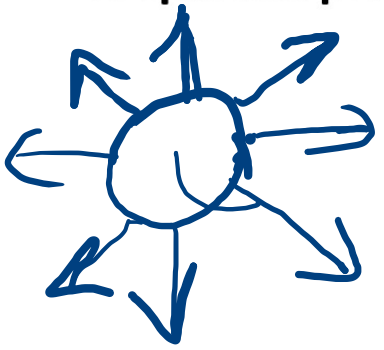
Cost(i,j) > Cost(i,k) + Cost(k,j)

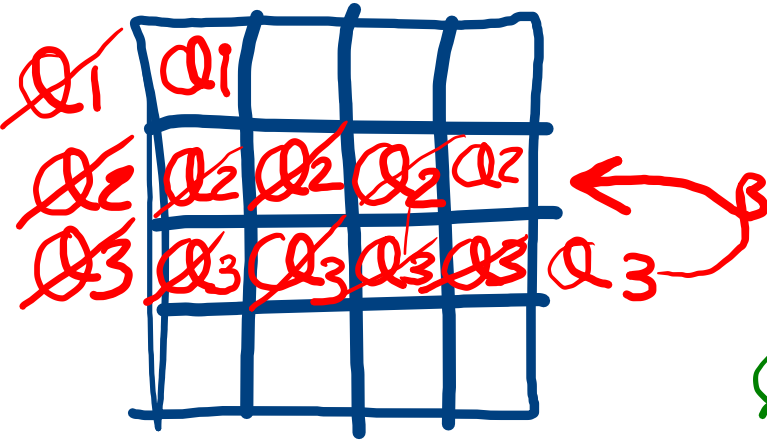
10 > 5+3

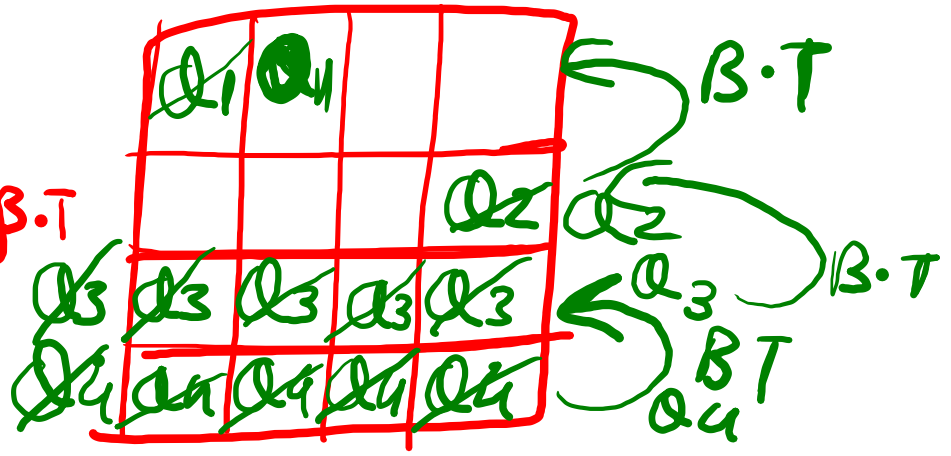
Types of Algorithms

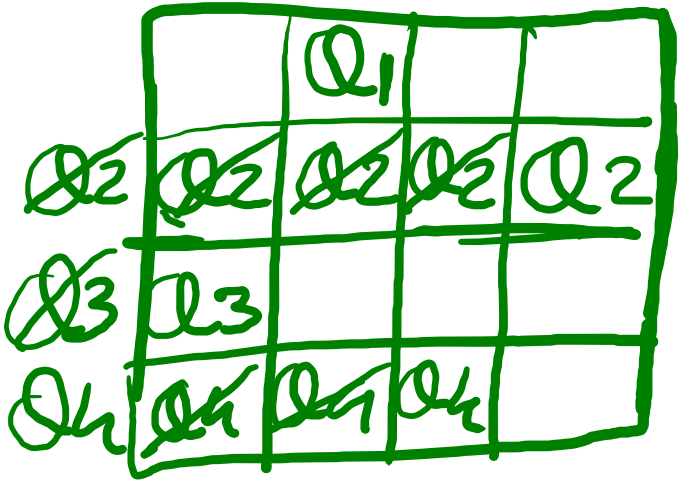
4 Back Tracking

- **Only solution needed . (May/May not be optimal)**
- **N-queens problem** → 4×4 → 4 Queens on 4×4 Board such that no one attacks other




Q1
Q2
Q3
B.T
Q3


Q1
Q4
B.T
Q2
Q2
Q3
B.T
Q3
Q4
B T
Q4


Q1
Q2
Q2
Q3
Q3
Q4
Q4

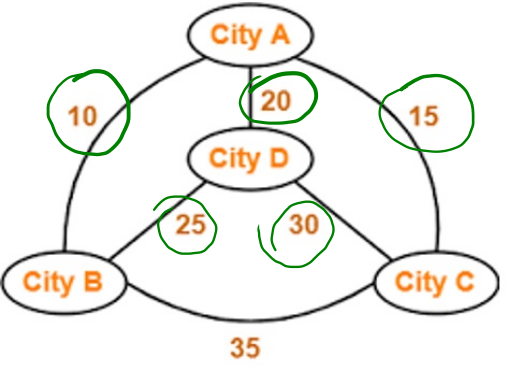
Types of Algorithms

option evaluated

5 Branch n Bound

Cost

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



Travelling Salesman Problem

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