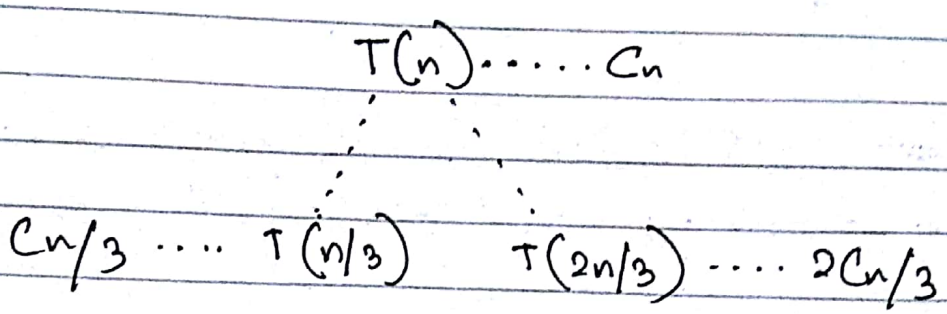
$T(n) = T(n/3) + T(2n/3) + Cn.$



$\frac{n}{(3/2)^k}$

$k = \log_{3/2} n$

$T(n) \leq Cn \log n.$

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

Master Theorem.

$T(n) = aT(n/b) + Cn^k$

$T(n) \in \Theta(n^k \log_b n)$ i) if $a = b^k$.

$\in \Theta(n^k)$ ii) if $a < b^k$.

$\in \Theta(n^{\log_b a})$ iii) if $a > b^k$.

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

$a = 1, b = 2, k = 0.$

$\therefore a = b^k.$

$T(n) \in \Theta(n^0 \log_2 n)$

$\in \Theta(\log_2 n)$

$T(n) = T(n-1) + C.$

$T(n) = 2T(\sqrt{n}) + \log n$

$n = 2^k$.

$T(2^k) = 2T(2^{k/2}) + k$.

$\downarrow$

$S(k) = 2S(k/2) + k$.

$\therefore a = 2, b = 2, k_1 = 1$.

$\therefore a = b^{k_1}$

$S(k) \in \Theta(k^1 \log_2 k)$.
$\in \Theta(k \log_2 k)$.
$\in \Theta(\log_2 n \cdot \log_2(\log_2 n))$

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

$T(n) = 3T(n/4) + n\log n$
$T(n) = aT(n/b) + f(n)$

$T(n) \in \Theta$
$\in \Theta(n^{\log_b a} \times \log_b n)$
$\in \Theta(n^{\log_b a})^{\log_b}$

$\exists$ constant > 0.

i). $f(n) \in \Omega(n^{\log_b a + \epsilon})$
ii). $f(n) \in \Theta(n^{\log_b a})$
iii) $f(n) \in O(n^{\log_b a - \epsilon})$

Divide and Conquer Techniques.

Binary Search.

- Best case $\in \Theta(1)$
- Worst case $T(n) = T(n/2) + C.$
  $\in \Theta(\log n)$
- Average case

$S_c \rightarrow$ Cases / Scenarios

No. of comparisons.

Let $n = 2^k - 1.$

$S_1 = 1.$

$S_2 = 2.$

$S_3 = 4.$

$\therefore S_c = 2^{c-1}$

$$\therefore T_{avg}^{Bsearch} = \frac{1}{n} \sum_{c=1}^{k} S_c * C.$$

$$= \frac{1}{n} \sum_{c=1}^{k} 2^{c-1} * C.$$

$$= \frac{1}{n} \left[\sum_{c=1}^{k} C * 2^c - \sum_{c=1}^{k} C * 2^{c-1} \right]$$

$$= \frac{1}{n} \left[\sum_{c=1}^{k} C * 2^c - \sum_{c=0}^{k-1} (C+1) * 2^C \right]$$

$$= \frac{1}{n}\left[k * 2^k - \sum_{c=0}^{k-1} * 2^c\right]$$

$$= \frac{1}{n}\left[k * 2^k - (2^k - 1)\right]$$

$$= \frac{1}{n}\left[k * 2^k - 2^k + 1\right]$$

$$= \frac{1}{n}\left[k * 2^k - 2^k + k - k + 1\right]$$

$$= \frac{1}{n}\left[2^k(k-1) - 1(k-1) + k\right]$$

$$= \frac{1}{n}\left[(2^k - 1)(k-1) + k\right]$$

$$= \frac{1}{n}\left[n(k-1) + k\right]$$

$$= (k-1) + k/n$$

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

$$\in O(\log_2 n)$$

$$= \frac{1}{n}\left[\sum_{c=1}^{k} c * 2^c - \sum_{c=0}^{k-1} (c+1) * 2^c\right]$$

$$= \frac{1}{n}\left[k * 2^k - \sum_{c=0}^{k-1} 2^c\right] - \sum_{c=1}^{k-1} c * 2^c$$

$$= \frac{1}{n}\left[k * 2^k - (2^k - 1)\right] - \sum_{c=0}^{k-1} 2^c$$

$$= \frac{1}{n}\left[k * 2^k - 2^k + 1\right]$$

$$= \frac{1}{n}\left[k * 2^k - 2^k + k - k + 1\right]$$

Merge Sort.

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

$a = 2, b = 2, k = 1.$

$a = b^k$

$$T(n) \in \theta(n^1 \log_2 n)$$

$$\in O(n \log_2 n)$$

Quick Sort

$$T(n) = T(n-k) + T(k-1) + (n+1)$$

Best case :- $k = n/2$.

Worst case :- $k = 1$.

For Average Case.

$$T(n) = (n+1) + \frac{1}{n}\sum_{k=1}^{n}[T(n-k) + T(k-1)]$$

$$nT(n) = n(n+1) + 2[T(0) + T(1) + \dots + T(n-1)]$$

$$(n-1)T(n-1) = (n-1)n \cdot + 2[T(0) + T(1) + \dots + T(n-2)]$$

$$nT(n) - (n-1)T(n-1) = 2n + 2T(n-1).$$

$$\therefore nT(n) = 2n + (n+1)T(n-1). \quad \text{Divide by } n(n+1).$$

~~$nT(n) = 2n$~~

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

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

$$= 2\sum_{k=3}^{n+1} 1/k + 1 \quad < \quad 2\sum_{k=1}^{n} 1/k + 1 < 2(\log_e n) + 1$$

Priority Queue

Enqueue (A, size, x, totalsize) → O(log₂ n) → Upper Bound 3.5 = 3.

```
{
    size ++; i = size;
    if (size > totalsize)
        overflow
    A[i] = x;
    parent = ⌊(i - 1)/2⌋;
    while ((A[parent] < A[i]) && (parent >= 0))
    {
        swap (&A[parent], &A[i]);
        i = parent;
        parent = ⌊(i-1)/2⌋          i-1/2
    }
}
```

Dequeue (A, size) → O(log₂ n)

```
{
    if (size == 0)
        Underflow
    X = A[0];
    swap(A[0], A[size-1]);
    size--;
    Heapify (A, size, 0);
}
```

Build heap

Height of a heap $(h) = \lfloor \log_2 n \rfloor$

No. of nodes @ height $h = \left\lceil \frac{n}{2^{h+1}} \right\rceil$

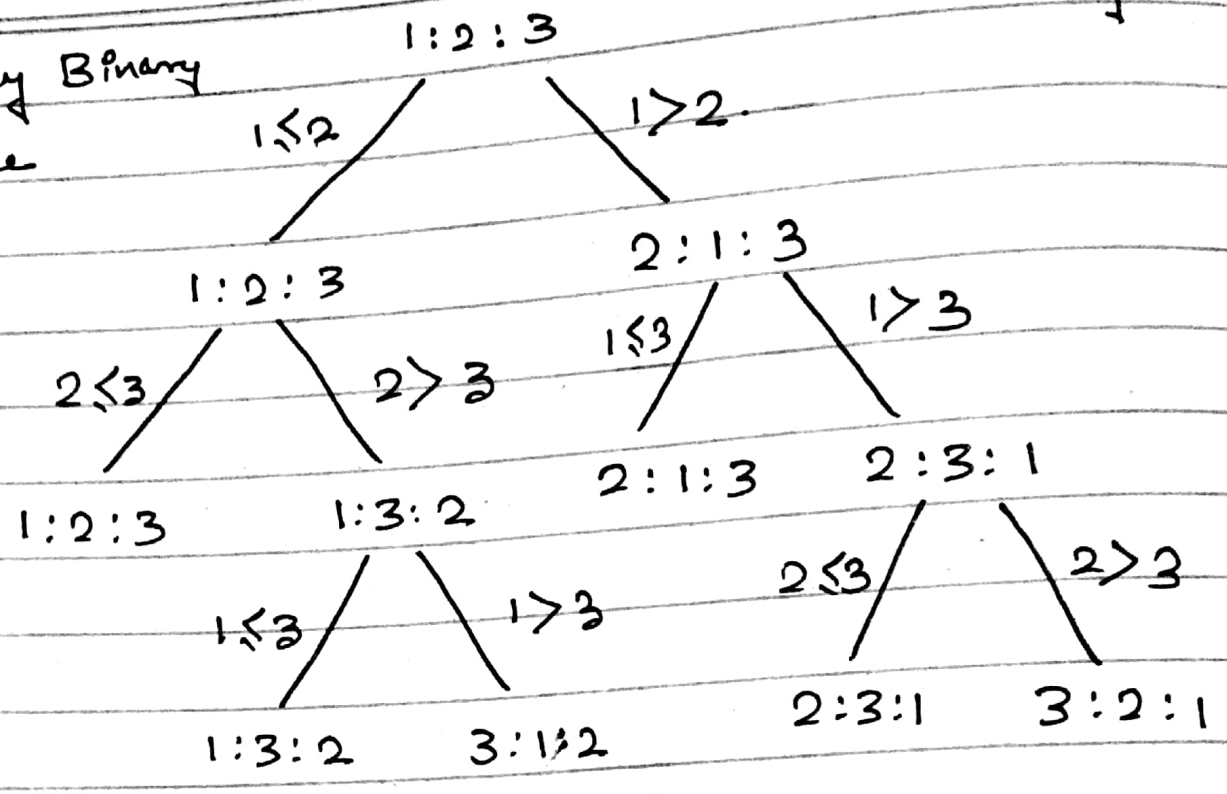
$$T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^{h+1}} < \sum_{h=0}^{\infty} \frac{h}{2^h} \rightarrow \frac{1/2}{(1-1/2)^2}$$

$$\in O(n)$$

$a_i \leq a_j$ or $a_i > a_j$

Fully Binary Tree



leaves have all permutations.

$$n! \leq L \leq 2^h$$

$$\therefore n! \leq 2^h$$

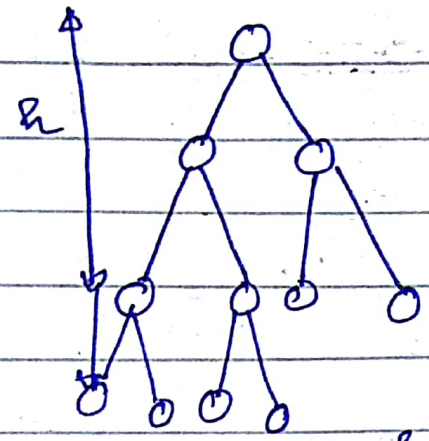
$$\log_2 n! \leq h. \Rightarrow \Theta(n \log n) \leq h.$$

$$2^n < n! < n^n.$$

$$\Rightarrow \log(n!) \in \Theta(n \log n).$$

∴

Worst case of Max heapify happens if we traverse the entire height of the heap which is only if any subtree atmost have $\frac{2n}{3}$ no. of nodes at lowest level, it is half filled.



Left Subtree should have (h-1) height.

Right subtree should have (h-2) height.

Left Subtree $\rightarrow 2^h - 1$

Right Subtree $\rightarrow 2^{h-1} - 1$.

$$\therefore \frac{2^h - 1}{\underbrace{2^h - 1}_{\text{Left}} + \underbrace{2^{h-1} - 1}_{\text{Right}} + \underbrace{1}_{\text{Rt}}}$$

$$\lim_{h \to \infty} \frac{2 \cdot 2^{h-1} - 1}{3 \cdot 2^{h-1} - 1} = \frac{2}{3}$$

Greedy Algorithm

(Huffman coding) → Data Compression.

- Optimal Substructure
- Greedy Choice
- Optimality ??

	a	b	c	d	e	f	
freq.	45	13	12	16	9	5	Σ=100
Same length code.	000	001	010	011	100	101	
Variable length code/ Huffman	0	101	100	111	1101	1100	

Huffman code Properties

(*) No code is a prefix of any other code → prefix code.

0 → left child

(*) Each leaf represents the character.

$|C| \rightarrow |C-1|$

No. of leaves → No. of internal Nodes.

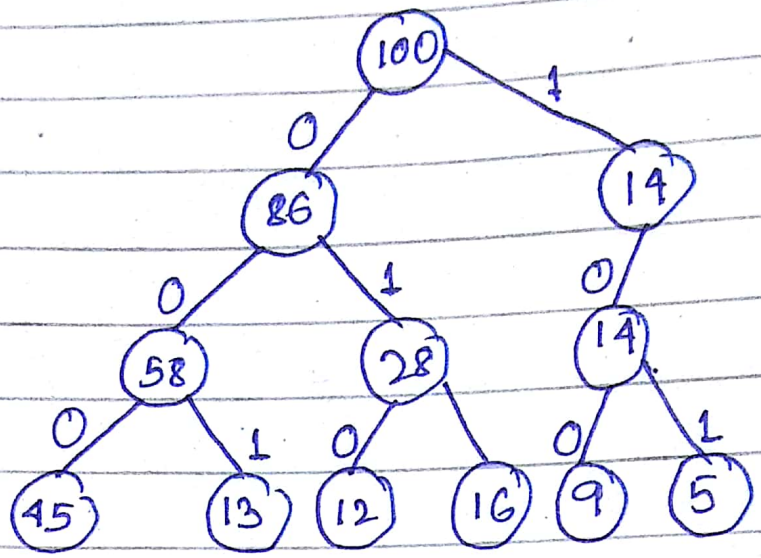
Data File → 1,00,000 characters

Size → 3,00,000

Size → $\frac{100000}{100}(45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4)$

$= 2,24,000$

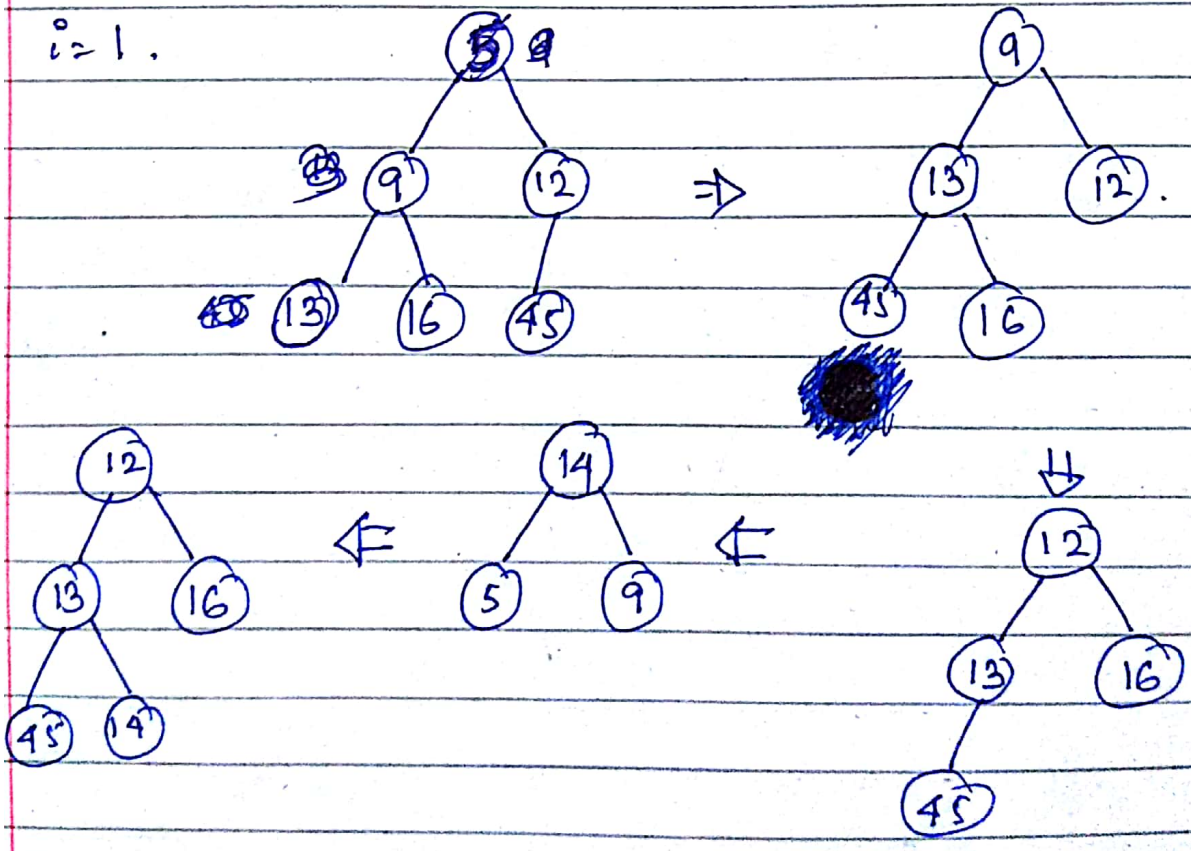
Data File Size $\sum_{i}^{|C|} C_{freq} * C_{depth}$.



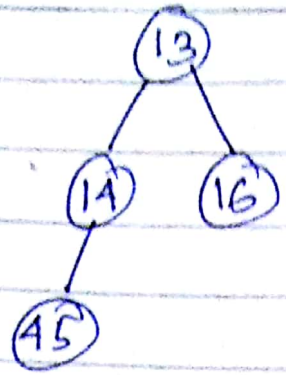
```
Huffman (C)
{
    n = |C| // 6.
    Q = C // Min Priority Queue.        | n or nlogn
    for(i=1; i<=n-1; i++)               (n-1)
    {
        createNewNode(z).
        z.left = x = Extract_Min(Q);    } 2logn.
        z.right = y = Extract_Min(Q);
        z.freq = x+y;
        Insert(Q, z);                   logn.
    } 3(n-1)logn.
    return Q;
}   n + 3(n-1)logn ≈ nlogn.
```

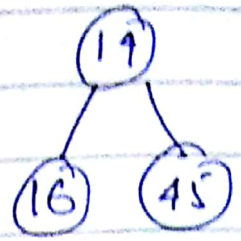
0	1	2	3	4	5
f:5	e:9	c:12	b:13	d:16	a:45

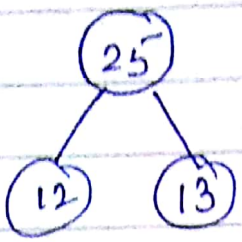
i = 1.

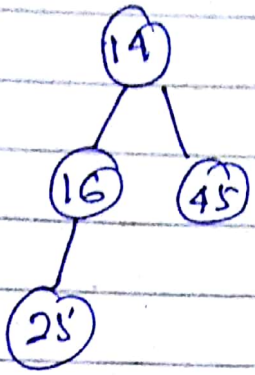


i=2.

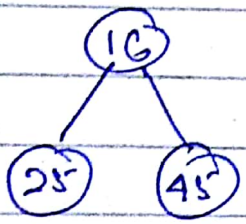


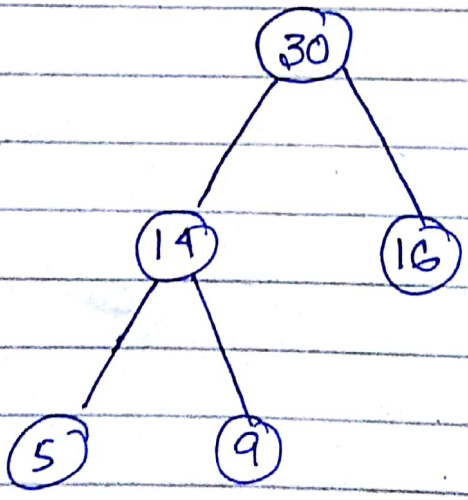


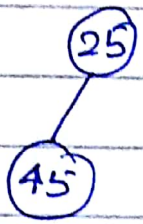


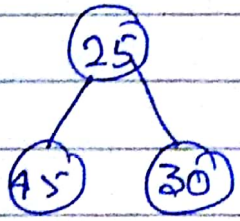


i=3

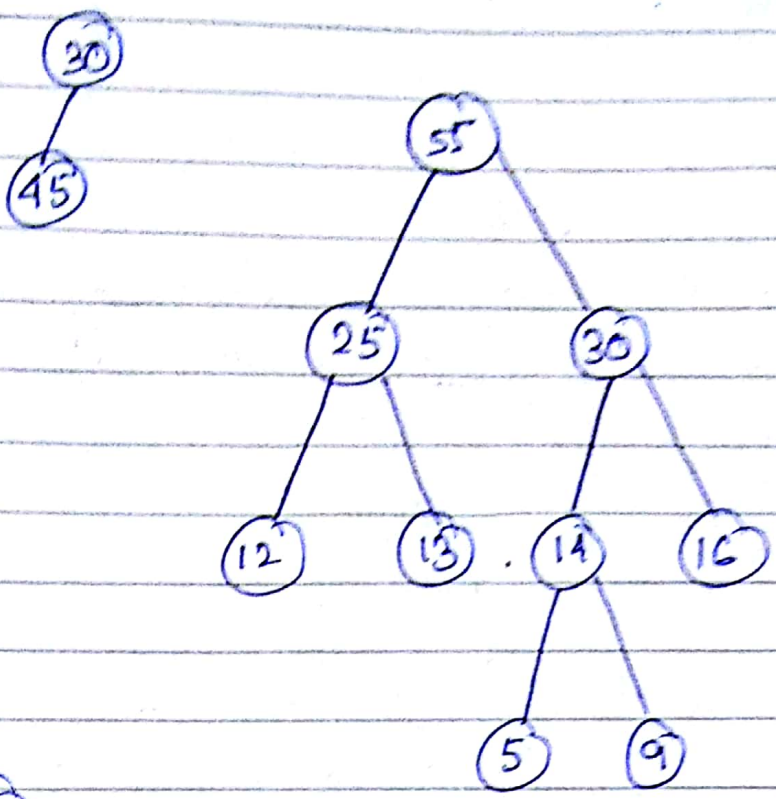






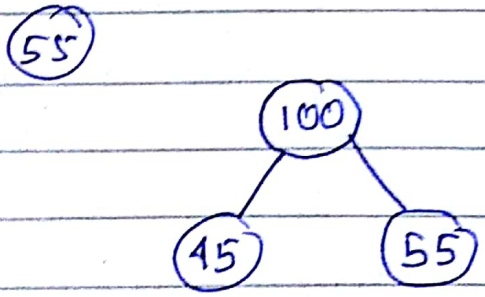


i = 4.



45

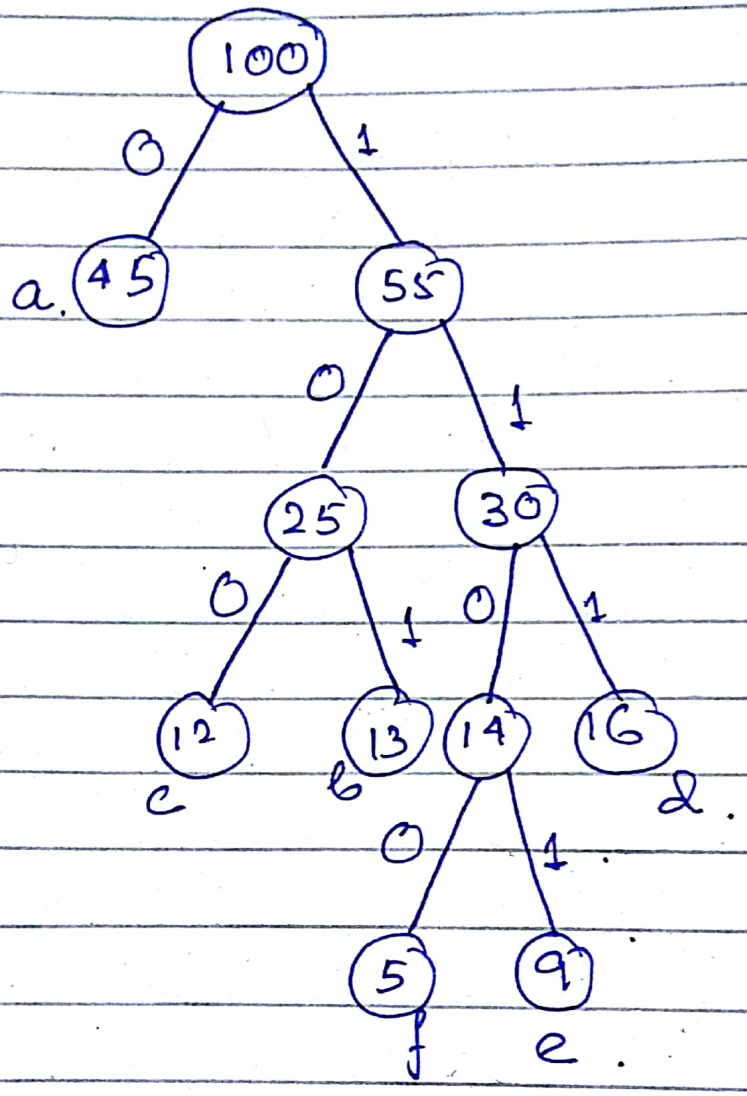
i = 5.



100

100

i = 5.



Greedy Technique.

i. Choose the solution, which is best in current scenario.
ii. Hope to get optimal result.
iii. But may not guarantee the optimality due to local optima and not global optima.
iv. Then how? Based on the greedy choice.

Fractional Knapsack.

W ← Max^m permissible weight.
P_i ← Profit of ith item.
W_i ← Weight of ith item.
x_i ← Fraction of ith item

Objective

$$\text{maximize} \left(\sum_{i=1}^{n} P_i x_i \right)$$

Such that

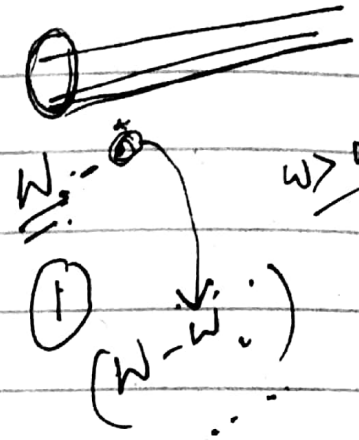
$$\sum_{i=1}^{n} x_i W_i \leq W$$

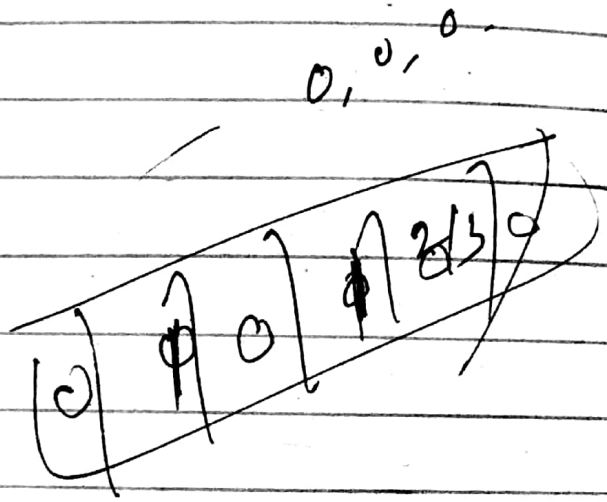
Sort the items in non-increasing order of profit/weight

$P_i/w_i \geq P_{i+1}/w_{i+1}$ ....

Now based on that order

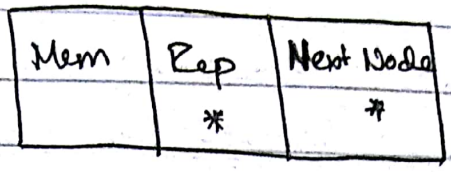
```
for(i=1; i<=n; i++)
    x_i = 0;
u = w;
for i ← 1 to n.
{   if (w_i <= u)
    {
        x_i = 1;
        u = u - w_i;
    }
    else
        break;
}
if (i<=n)
{
    x_i = u/w;
}
```

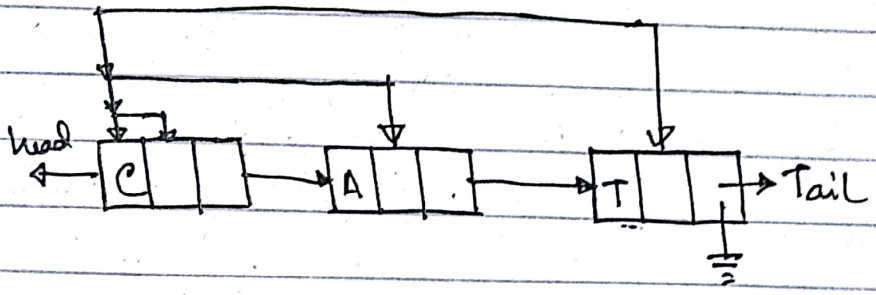





```
for(i=0; i<num; i++)
{
    for(j=i+1; j<num; j++)
```

Union Find



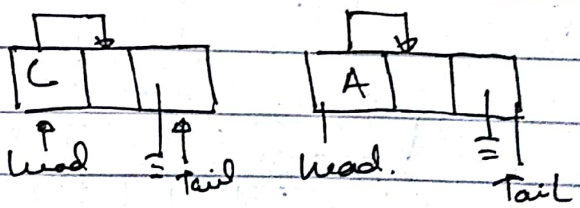


3 Operations

i. MAKESET(U) $\in O(1)$

It will create a single node.

MAKESET(C).



MAKESET(A).

ii. FINDSET(U) $\in O(1)$.

It returns the representative of the member U.

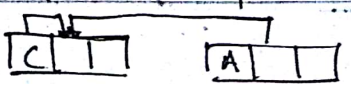
iii. UNION(U & V).

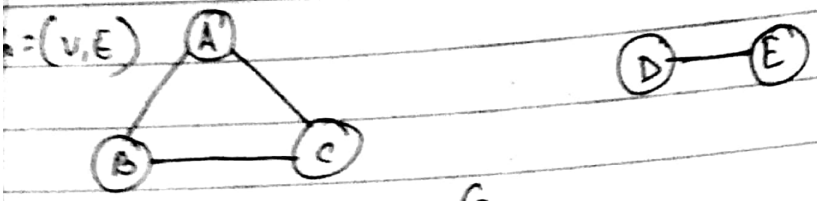
U → Tail = V → Head.

V → rep = U → rep

UNION(S1, S2).

Weighted Union Heuristic





G.

For each $v \in V$

MAKESET(v);

For each $(u,v) \in E$

if(FIND-SET(u) ≠ FINDSET(v))

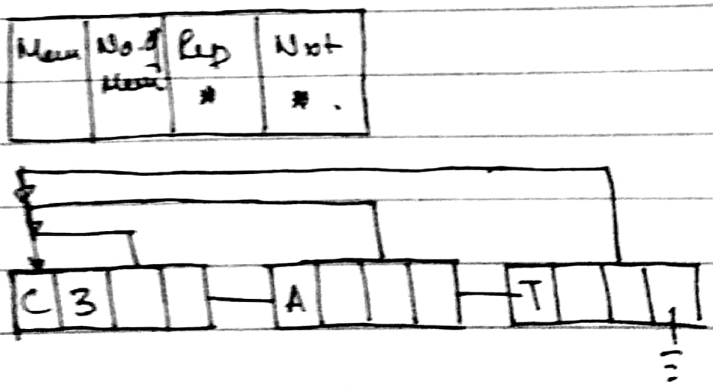
UNION(u,v);

{A} {B} {C} {D} {E}.

{A,B} {C} {D} {E}.

{A,B} {C} {D,E}.

{A,B,C} {D,E}.



MAKE SET (x_1)
⋮
MAKESET (x_n)
] $\in O(n^3 + n^2)$.

UNION (x_2, x_1).
UNION (x_3, x_2).
⋮
UNION (x_n, x_{n-1})

$\sum_{i=1}^{n-1} i$ $(n+n)$ $< n \cdot \lfloor \log_2 n \rfloor$ $(n + n\log_2 n)$

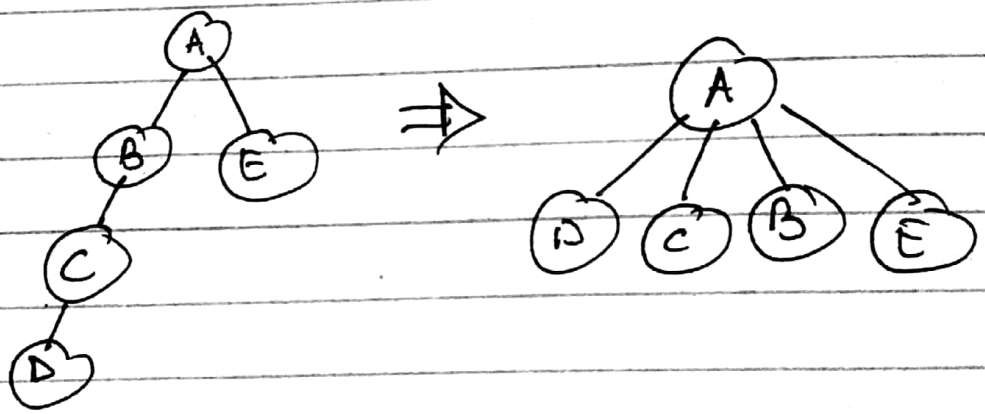
UNION (x_1, x_2) UNION (x_0, x_2)
UNION (x_1, x_3) UNION (x_3, x_4).
⋮ ⋮
UNION (x_1, x_n) UNION (x_1, x_3)
⋮
UNION (x_n, x_{n+1})

Path Compression.

Here entire set is represented by a tree & the members will point to the representative which is the root node. Each node will have extra info, i.e. rank = height of the node.

```
MAKESET(x)
{
    create a new x;
    x.p = x;  // p = parent.
    x.rank = 0;
}
```

```
FINDSET(x)   // Path compression.
{
    if (x != x.p)
    {
        x.p = FINDSET(x.p);
    }
    return x.p;
}
```



```
UNION(x, y)
{
        LINK(FINDSET(x), FINDSET(y));
}
LINK(x, y)
{
        if (x.rank > y.rank)
        {
            y.p = x.p.
        }
        else
        {
            x.p = y.p ;
        }
        if (x.rank == y.rank)
        {
            y.rank + = 1 ;
        }
}
```

$$C(S,T) = \sum_{u \in S} \sum_{v \in T} C(u,v)$$

$$|f| = f(S,T).$$

$$= \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{v \in S} \sum_{v \in T} f(v,v).$$

Correctness Proof of Prim's and Kruskal.

1. Let U be an actual MST.
2. Let T be a MST found by Prim's / Kruskal

Now, we have to prove $U \triangleq T$

Let f be an Min^m Weighted edge in T.
Now if f included in U, then obviously there will be a cycle. Now U, must have an edge e, which is removed from U & include in T then cycle is removed from U [as T does not have e].

$U + \{f\} - \{e\}$ → Resultant MST.

if $\{e\} > \{f\}$.
Then U is not MST.
if $\{e\} < \{f\}$
then for the greedy approach, $\{e\}$ will be picked before $\{f\}$.

Correctness Proof of Bellman Ford.

P. In Bellman Ford from source (s) to vertices ($v \in V$) is a simple path if K is the length of that path, then $K \leq |V| - 1$.

So for $|V|-1$ iterations, we have relaxed each edge, sufficient

$key[v] = \delta(s, v)$ ← Shortest Path

ii. There does not exist any -ve cycle, Bellman Ford returns TRUE

$$key[v] \leq \delta(s, u) + wt(u, v).$$

iii. There exist a -ve cycle, still algorithm returns true.

Let -ve cycle is in the form $\{v_0, \ldots v_k\}$ $k > 0$.

Then $\sum_{i=1}^{k} wt(v_{i-1}, v_i) < 0$.

if Bellman Ford is giving me true result, then

$$\sum_{i=1}^{k} key[v_i] \leq \sum_{i=1}^{k} key[v_{i-1}] + \sum_{i=1}^{k} wt(v_{i-1}, v)$$

Dijkstra's Correctness

From

If any shortest path found then any subpath of that path is also shortest; so if ∃ any shorter subpath, it will replace the subpath.

```
int power(int x, int n)
{
    if(n==0)
        return 1;
    if (n==1)
        return x;
    if(n%2==0)
        return power(x, n/2) * power (x,n/2);
    else
        return x*power(x,n/2)* power (x,n/2);
}
```

$$T(n) = C \quad , \; n=0 \text{ or } n=1.$$

$$= 2T\left(\frac{n}{2}\right) + C \quad , \; n>1.$$

$$= 2.\left(2T\left(\frac{n^2}{2^2}\right) + C\right) + C$$

$$= 4T\left(\frac{n}{4}\right) + 2C + C.$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^2 C + 2C + C.$$

∴ For kth term.

$$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} C + \ldots 2^2 C + 2C + C.$$

Let $2^k = n$.

$$2^k C + 2^{k-1} C + \cdots + 2^2 C + 2C + C.$$

$$= C\left(\frac{2^k - 1}{2-1}\right) + C.$$

$= (n-1)C + C$

$= nC$

$\therefore \in O(n)$.

```
int power(int x, int n).
{
    if(n==0)
      return 1;
    if(n==1)
      return x;
    int a = power(x, n/2);
    if(n%2==0)
      return a*a;
    else
      return x*a*a;
}
```

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

$= \{T(n/2^2) + C\} + C$

$= T(n/4) + 2C$

$= T(n/2^3) + 3C$

$\therefore$ For kth Term.

$= T(n/2^k) + kC.$ $\quad [2^k = n]$

$= T(n/n) + C\log_2 n$.

$= T(1) + C\log_2 n$

$= c + c\log_2 n = c(\log_2 n + 1)$

$\therefore \in O(\log_2 n)$.

Substitution Technique for Solving Recurrence Relation.

$T(n) = 2T(\lfloor n/2 \rfloor) + n$.

Let us assume $T(n) \in O(n\log n)$

$T_n \leq Cn\log n$.

$\therefore T(n/2) \leq C\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor$

$\leq 2C\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor$

$\leq Cn(\log n - \log 2) + n$.

$\leq Cn(\log n - 1) + n$.

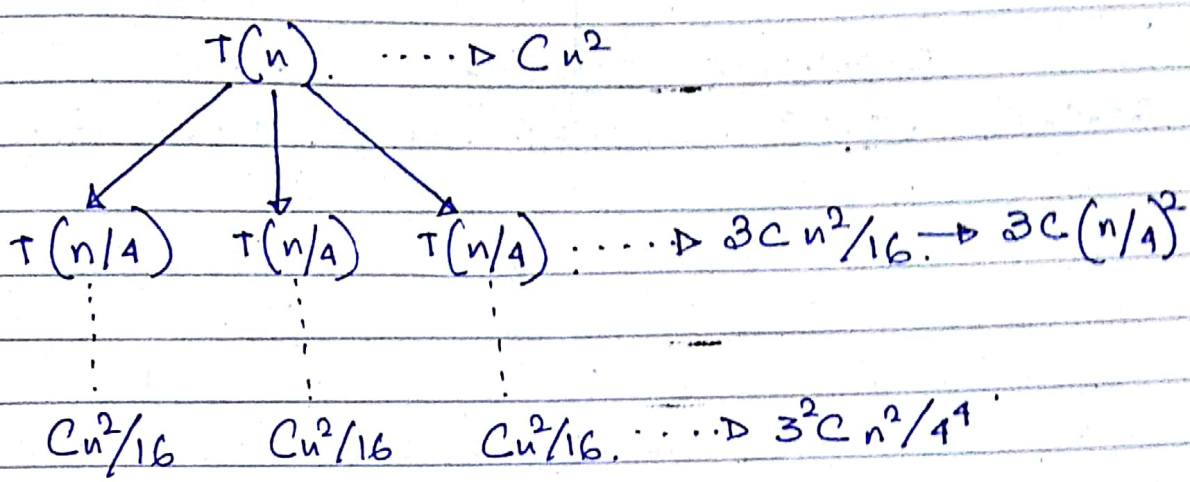
$\leq Cn\log n - Cn + n$.

$\therefore$ It is less than $n\log n$.

$\therefore \in O(n\log n)$.

Recurrence Tree For Solving Recurrence Relation.

$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.



Height = $\log_4 n$.

For i^{th} level $\rightarrow 3^i$ children.

For leaf nodes

$3^i \ldots \Theta(1)$.

$i = \log_4 n = n^{\log_4 3}$.

$\Rightarrow \sum_{i=0}^{\log_4 n - 1} 3^i \frac{Cn^2}{4^{2i}} + n^{\log_4 3}$.

$\sum_{i=0}^{\log_4 n - 1} 3^i \frac{Cn^2}{4^{2i}} + n^{\log_4 3} < \sum_{i=0}^{\infty} \frac{3^i Cn^2}{4^{2i}} + n^{\log_4 3}$

$\approx \left(\frac{1}{1 - \frac{3}{16}}\right) Cn^2 + n^{\log_4 3}$.

$= \frac{16}{13} Cn^2 + n^{\log_4 3}$.

$\therefore \frac{16}{13} Cn^2 \leq \sum_{i=0}^{\log_4 n - 1} \frac{3^i Cn^2}{4^{2i}} \ldots$

$\therefore \in O(n^2)$.

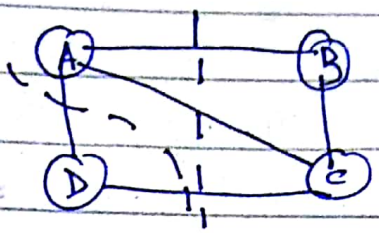
MST · Let T be an acyclic subset of $(T \subseteq E)$ an undirected graph $(G(V, E))$ which connects all the vertices such that $\sum_{(u,v) \in T} wt(u,v)$ is minimized.

Greedy approach → Kruskal → Union find.
→ Prim's → Priority Queue

Anotation:- Let A be always a subset of a MST of a graph (G)

Safe edge :- It is the edge which is added to the set A the Anotation will not fail.

Cut :- Cut is the partition of the Graph (G).
{S, (V-S)}



Lightweight edge :- It is the min^m weighted edge (u,v) which crosses the cut & it means one of its vertex is @ {S} and other @ {V-S}

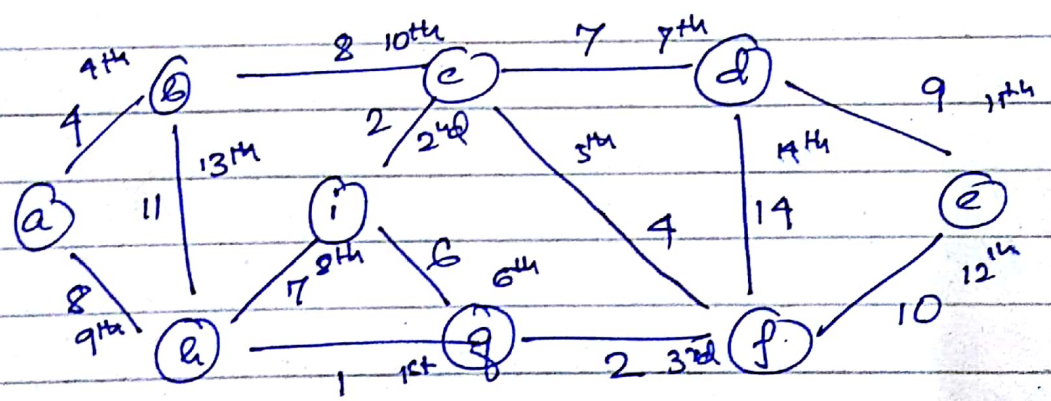
Theorem :- If G is an undirected graph where A is a subset of a MST of graph G, that represents the cut ({S}, {V-S}) then lightweight edge is the safe edge.
(Greedy Choice)

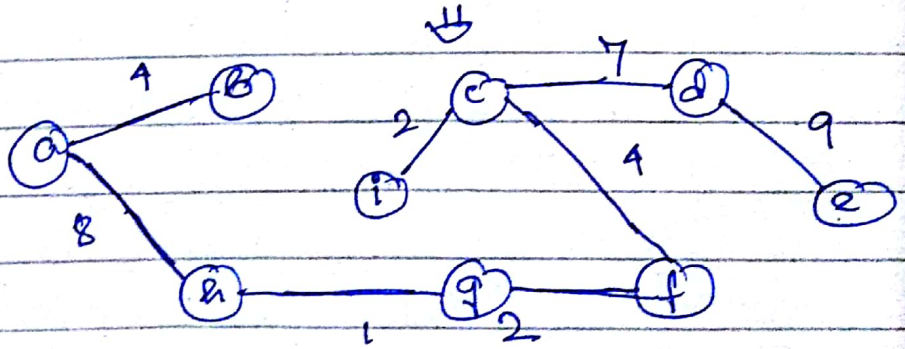
```
Kruskal(G)
{
    A = ϕ                                   ⟼ |E| log |E|
    Sort the edges in non-decreasing order of their
    weight, i.e w_i ≤ w_{i+1} ≤ w_{i+2}.
    Now based on the order
    for each (u,v) ∈ E
    {   if (FIND_SET(u) ≠ FIND_SET(v))
        {
            A = A ∪ {u,v};
            UNION {u,v}; ⟼ log|V|
        }
    }
}
```

|E| log |V|

E < V2

∴ |E| log |V|.





6 8 10

Prim's Algorithm

i. A priority queue (Q) of size $|V|$, which consists a set of vertices that are not included in the MST with a associated key value.
key [v] : cost (u,v) where v is already in the MST.

ii. $\pi[v]$ = Parent of v

iii. Generic MST (A): $(v, \pi[u]) \mid [v \in V - Q - \{r\}]$ (r = root)

Algo stops : $(v, \pi[v]) \mid [v \in V]$

Prim's Algorithm

```
Prim(G, {r})
{
    for each v ∈ V                          } |V|
    {
        key[v] = ∞;
        π[v] = NIL;
    }
    key[r] = 0;
    Build(Q); // of size |V| → |V|.
    while(Q != ϕ)                           |V|
    {
        u = ExtractMin(Q)  → log |V|
        for each v ∈ Adjacent[u] (Degree of u)
        {
            if(v ∈ Q && key[v] > wt(u,v))
            {
                π[v] = u;
                key[v] = wt(u,v); // Calling of Heap decrease key.
                                  ↓
                                  log|V|.
            }
        }
    }
```

Σ degree of vertices = 2|E|

$\therefore T \in O(\log |V| (|V| + |E|))$

Single Source Shortest Path (Dijkstra).

Greedy Technique

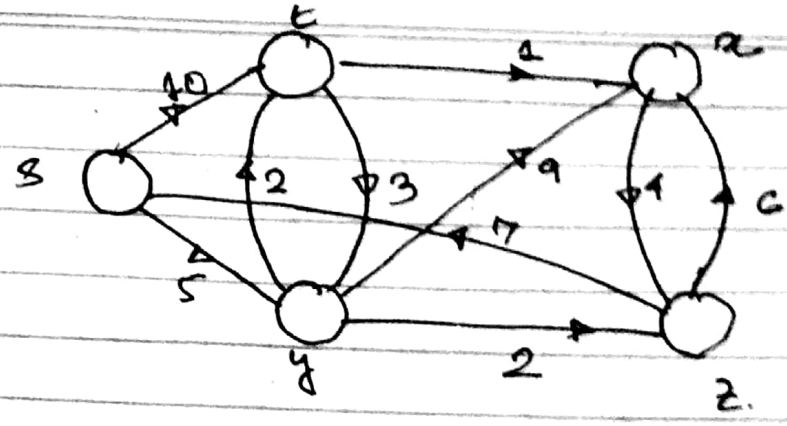
```
Initialize_sre(G, w, s)
{
for each v∈V
 {
    key[v] = ∞;
    π[v] = NIL;
 }
key[s] = 0;
}
```

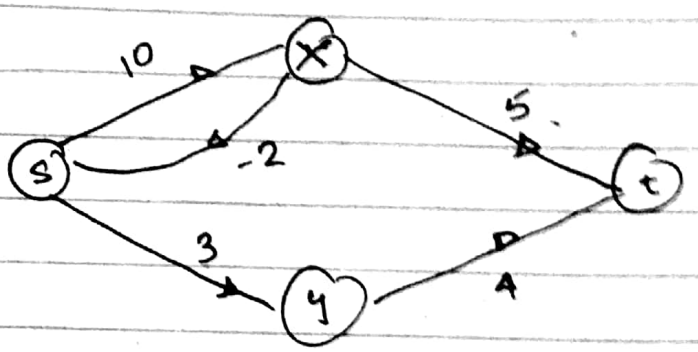
2. Relax (u, v, w).

```
if {key[v] > key[u] + wt(u,v)}
 {
   key[v] = key[u] +
            wt(u,v);
   π[v] = u;
 }
}
```

Greedy Technique

```
Dijkstra (G, c, w)
{
   Initialize_sre (G, w, s);
   Build(Q);
   S = ϕ;
   while (Q != ϕ)
   {
      u = Extract Min (Q);
      S = S ∪ u;
      for each v ∈ Adjacent[u];
            relax (u, v, w);
   } }
}
```






```
Bellman Ford (G, s, w).
{
  Initialize_source(G, w, s);
  for i ← 1 to |V| - 1
  { for each (u,v) ∈ E
      relax(u, v, w);
  }
  for each (u,v) ∈ E
  {
      if (key[v] > key[u] + wt(u,v))
          return FALSE;
  }
  return TRUE;
}
```

So when ∃ a -ve edge, we do not u dijkstra, we prefer Bellman for.

$O(|V| * |E|)$.