Design & Analysis of Algorithm

* जिस algorithm की time complexity & space complexity कम होगी उसको prefer करेंगे।

Time Complexity:- Count the no. of primitive operation (steps) as a function of input size

→ Average(a[], n){

sum = 0; —— c_1

for(i=0; i<n; i++){ → $c_2 n$

sum = sum + a[i]; → $c_3 n$

}

Average = sum/n; → c_4

}

Time = $c_1 + (c_2 + c_3) n + c_4$

= order of n

Comparision of Function:- → CSE में आने वाले function non-negative ही

① $n^{\log n}$ vs $n^{\sqrt{n}}$

$\log(n^{\log n}) = \log n \cdot \log n$

$\log(n^{\sqrt{n}}) = \sqrt{n} \log n$

common part को ignore करेंगे अब

$\log n < \sqrt{n}$

$n = 2^8$

$\log_2 2^8 < \sqrt{2\times2\times2\times2\times2\times2\times2\times2}$

$8 < 16$

$\therefore n^{\log n} < n^{\sqrt{n}}$

3^n Vs $2^{\sqrt{n}\log n}$

$\log(3n^{\sqrt{n}}) = \sqrt{n}\log n + \log 3$ $\quad \therefore 3n^{\sqrt{n}} > 2^{\sqrt{n}\log n}$

$\log(2^{\sqrt{n}\log n}) = \sqrt{n}\log n \log_2 2$

$2^{\sqrt{n}\log n} = 2^{\log_2 n^{\sqrt{n}}} = n^{\sqrt{n}}$

~~$\log n$ $\log n$~~

~~$\log 3 + \log n$ <> $\log n$~~

Q. $n^2\log n$ Vs $n(\log n)^{10}$

n common & eliminate करो

$n\log n$ V $(\log n)^{10}$

$\log(n\log n) = \log n + \log(\log n)$ $\quad \log((\log n)^{10})$

$10\log(\log n)$

$\therefore n^2\log n > n(\log n)^{10}$

Q $10, \sqrt{n}, n, \log_2 n, \frac{100}{n}$ Grade 2017

$\frac{100}{n} < 10 < \log_2 n < \sqrt{n} < n$

Q. $2^n, n^{3/2}, n\log n, n^{\log n}$

$\log(2^n) = n\log_2 2 = n$

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

$\log(n\log n) = \log n + \log(\log n)$

$\log(n^{\log n}) = \log n \cdot \log n$

~~$n^{\frac{3}{2}}$~~

~~$2^n > n^{3/2} > n^{\log n} > n\log n$~~

$2^n > n^{\log n} > n^{3/2} > n\log n$

** constant < Logarithmic < polynomial < Exponential
$(\log n)$ $\quad n^k$ $\quad c^n$

$1 < \log\log n < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 \ldots < 2^n < 3^n <$

→ Big-oh Notation (O) - upper Bound

$f(n) = O(g(n))$ iff

$f(n) \leq c \cdot g(n)$ for some value of $c > 0$ && for all $n \geq n_0$

→ Big omega Notation (Ω) - Lower Bound

$f(n) = \Omega(g(n))$ iff

$f(n) \geq c \cdot g(n)$ for some value of $c > 0$ && for all $n \geq n_0$

Q $f(n) = n$, $g(n) = 3n + 2$

एक c की value बताइए ..

$f(n) \geq \frac{1}{10} \cdot g(n) \; \forall \; n \geq 1$

$\therefore f(n) = \Omega(g(n))$

also $g(n) = \Omega(f(n))$ → $g(n) \geq 5 f(n) \; \forall \; n \geq 1$

→ Theta Notation (Θ) - Average Bound

$f(n) = \Theta(g(n))$ iff

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for some values of c_1 & c_2 for all values of $n \geq n_0$

$f(n) = 3n + 2$, $g(n) = n$

$0 \leq 5g(n) \leq f(n) \leq 1 g(n)$

$\therefore f(n) = \Theta(g(n))$

* $f(n) = 3n^2 + 4n + 2$

$g(n) = n^2$

*1 $f(n)$ & $g(n)$ are asymptotically equal दोनों function की highest power 2 है।

o $f(n) = n$ $g(n) = n^2$

$n \geq c_1 n^2$ $n \leq c_2 n^2$

⇓

Never possible $f(n) \neq \Omega(g(n))$ ∴ $f(n) \neq \Theta(g(n))$

→ small-o-Notation

$f(n) = o(g(n))$ iff $f(n) < c \cdot g(n)$

all value of $c > 0$ && for all value of $n \geq n_0$

or $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ आ० गया तो small o satisfy हो रहा है

$f(n) = n$ $g(n) = n^2$ $f(n) < c \cdot n^2$ $\forall c > 0$, $n \geq 1$

∴ $f(n) = o(g(n))$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{n}{n^2} = 0$

* $f(n)$, $g(n)$ & $f(n) = o(g(n))$ ← small-o → reverse is not true
then $f(n) = O(g(n))$ ← Big oh

o $f(n) = \log n$

$g(n) = n$ $\lim_{n \to \infty} \frac{\log n}{n} = \lim_{n \to \infty} \frac{1}{n} = 0$

∴ $f(n) = o(g(n))$ मतलब $\log n < c\, n$ $\forall c > 0$

→ Small Omega Notation (ω)

$f(n) = \omega(g(n))$ iff $f(n) > c\, g(n)$

for all value of $c > 0$ && for all value of $n \geq n_0$

or

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ ⇒ $f(n) = \omega(g(n))$

* $f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$ but reverse is not true

@ $f(n) = n^2$, $g(n) = n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} : \lim_{n\to\infty} \frac{n^2}{n} = \infty$$

*** $f(n)$ & $g(n)$ का highest power same है तो $f(n)$ & $g(n)$ are asymptotically equal.

जब तक होले ना small o है तब तक Big oh मानते है।

→ $\Omega(n^2) = \Omega(n) = \Omega(1)$.

↳ इसको इसके lower function से replace कर सकते है।

$O(n^2) = O(n^3) = O(n^4)$

↳ इसको इसके bigger function से replace कर सकते है।

$\Theta(n^2)$ को सिर्फ $\Theta(n^2)$ ही लिख सकते है

$f(n) = n^2 + 2n + 3$

↓

$O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$, $O(n^2 \log n)$, $\omega(n)$

↓ small o

* small $o(n)$ को इससे बड़े function से replace कर सकते है & $\omega(n)$ को इससे छोटे function से replace कर सकते है

@ 1. $100 n \log n = O\left(\frac{n \log n}{100}\right)$ ✓

2) $\sqrt{\log n} = O(\log \log n)$ ✗ $\log\log n < \sqrt{\log n}$

3) if $0 < x < y$ then $n^x = o(n^y)$ ✓ $n^x < n^y \Rightarrow x < y$

4) $2n \neq O(nk)$ ✗ $2n = O(n)$

5) $(n+k)^m = \Theta(n^m)$ ✓ $(n+k)^m$ में highest power वाला term n^m है।

6) $2^{n+1} = O(2^n)$ ✓

7) $2^{2n+1} = O(2^n)$ ✗ $\quad 2^n \cdot 2^n + 1 > 2^n$

Q $x = \sum_{i=0}^{n} i^3$ $\quad x = \left[\frac{n(n+1)}{2}\right]^2$

1) $\Theta(n^4)$ ✓ $\quad$ 2) $\Theta(n^5)$ $\quad x = \frac{n^2 (n+1)^2}{4}$ $\quad \Omega(n^4)$ $\Theta(n^4)$

3) $O(n^5)$ ✓ $\quad$ 4) $\Omega(n^3)$ ✓ $\quad \Rightarrow O(n^4)$

Assume $f(n) \geq 1$

Q 1) If $f(n) = O(g(n))$
then $2^{f(n)} = O(2^{g(n)})$ ✗

2) If $f(n) = O(g(n))$
then $f(n) * h(n) = O(g(n) * h(n))$ ✓

3) $f(n) = O(f(n))$ ✓

4) $f(n) = O(f(n/2))$ ✗ → $f(n) = 2^{2n}$, $f(n/2) = 2^n$, $2^{2n} > 2^n$

5) $f(n) = O((f(n))^2)$

6) $\frac{1}{n} = \Theta(1)$, $\frac{1}{n} = O(1)$, $\frac{1}{n} = \Omega(1)$ ✗

$\frac{1}{n} \leq c \cdot 1$ True

$\frac{1}{n} \geq c \cdot 1$ False

→ Properties of Asymptotic Notation

Reflexive: $f(n) = O(f(n))$

Symmetry: $f(n) = O(g(n)) \Leftrightarrow g(n) = O(f(n))$

Transitivity: $f(n) = O(g(n))$ & $g(n) = O(h(n))$ then $f(n) = O(h(n))$

Notation	Reflexivity	Symmetry	Transitivity
Big O	✓	×	✓
Big Ω	✓	×	✓
Theta Θ	✓	✓	✓
Small o	×	×	✓
Small ω	×	×	✓

1. $n^2 \cdot 2^{3\log_2 n} = \Theta(n^5)$ ✓

$n^2 \cdot 2^{\log_2 n^3} = n^2 \cdot n^3 = n^5$

2. $\frac{4^n}{2^n} = \Theta(2^n)$ ✓

3) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$ ✓

$f(n) = n^2$
$g(n) = \log n$
$n^2 + \log n = \Theta(n^2)$

4) $n! = \Theta(n^n)$ ✓

5) $f(n) = O(g(n))$ && $g(n) \neq O(f(n))$
$g(n) = O(h(n))$ && $h(n) = O(g(n))$

$f(n) = n$
$g(n) \neq n$
$g(n) = n^2$
$h(n) = 1 < n^2$

a) $f(n) + g(n) = O(g(n))$ ✓

b) $f(n) = O(h(n))$ ✓

c) $h(n) = \Theta(f(n))$

d) $f(n) \times g(n) = O(h(n) * h(n))$ ✓

$n * n^2 = O(k n^4)$

Q1 $f(n) + \text{small } o(f(n)) = \Theta(f(n))$ ✗

$n + n^2 \neq \Theta(n)$

Q2 $o(f(n)) \cap \omega(f(n)) = \phi$ ✓ (small o)

$n^2 \cap \log n = \phi$

$f(n) = n$

$o(f(n)) = n^2$

$\omega(f(n)) = \log n$

* $\log_2^* 16 = \log_2^* 2^4$

$= \log_2^* 4$

$= \log_2 2$

$= 1$ step

} 3 $\therefore \log_2^* 16 = 3$

1 जब आएगा तब stop कर जाएगा

$\log_2^* n$ = no. of step जब तक 1 ना आ जाए।

3) If $f(n) = \log(\log^* n)$ && $g(n) = \log^*(\log n)$

then $f(n) = O(g(n))$ ✗

$n = 2^{2^{2^{2^{\cdots}}}}$ 8th

$\therefore f(n) = \log(\log_2^* 2^{2^{2^{\cdots}}})$

$= \log(8)$

$= 3$

$g(n) = \log^*(\log_2 2^{2^{2^{\cdots}}})$

$= \log_2^*(2^{2^{2^{\cdots}}})$

$= \infty$

$\therefore \log(\log^* n) < \log^*(\log n)$

$\therefore f(n) \neq O(g(n))$

4) If $f(n) = \sin n$ && $g(n) = \cos n$ then what the relation b/w $f(n)$, $g(n)$

$f(n) \neq O(g(n))$

$f(n) \neq \Omega(g(n))$

$\Rightarrow f(n)$ & $g(n)$ is non-comparable

Q $O(n^2) + \Omega(n^2) + \Theta(n^2) = ?$

1) $O(n^2)$ 2) $\Omega(n^2)$ 3) $\Theta(n^2)$ 4) NOT

Sol $O(n^2)$ functions less than or equal to n^2

$\Omega(n^2)$ function ~~less than~~ Greater than or equal to n^2

$\Theta = n^2$

$O(n^2) + \Omega(n^2) + \Theta(n^2)$

$\leq n^2 + \geq n^2 + n^2 \Rightarrow O(n^2)$ because n^3 से लिख नहीं $O(n^2)$ वाली लिख सकते हैं।

$\Omega(n^2) \Rightarrow$ इस case में लिख सकते हैं।

→ Linear Search

Best case = 1 = constant ⇒ $O(1)$, $\Omega(1)$, $\Theta(1)$

Best case ≤ Average case ≤ worst case

Average case $\frac{n+1}{2}$ = $O(n)$, $\Omega(n)$, $\Theta(n)$

worst case : $n = O(n)$, $\Theta(n)$, $\Omega(n)$

Q Let $w(n)$ and $A(n)$ denote respectively, the worst case and Average case running time of an algorithm executed on an input size n which of the following is ALWAYS TRUE?

I) $A(n) = \Omega(w(n))$

2) $A(n) = \Theta(w(n))$

3) $A(n) = O(w(n))$

4) $A(n) = o(w(n))$

worst case $w(n)$

Average case $A(n)$

$A(n) \leq w(n) \Rightarrow A(n) = O(w(n))$

$A(n) = w(n)$ हो सकता है

$\therefore A(n) = \text{small } o(w(n))$

→
```
for (int i=n; i<=n; i++) { → (1)
    Printf ("Hello"); — (1)
}
```

Time complexity. $O(1)$

→
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j=j+i)
        Printf ("Hello");
```

$i=1$ $j=1$ to n

$i=2$ $j = n/2$ times

$i=3$ $j = n/3$ times

Time $= n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} + \cdots + \frac{n}{n}$

$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}\right]$

$= n \log n$

④
```
int Fun(int n){
  for(i = n/2 ; i<=n; i++){ ---- n/2
      for(j=2 ; j<=n; j = j*2){ - log n
          k = k + n/2 ;
      }
  }
  return k
}
```
Time complexity : $O(n \log n)$

⑤
```
int F(int n) {
    int count = 0;
  for (i=0; i<n; i++) {
      for (j=i; j>0; j--) {
          count = count+1;
      }
  }
  return count;
}
```
i = 0 , j = 0 times
i = 1 , j = 1 times
i = 2 , j = 2 times
:
i = n-1 , j = (n-1) times

Time complexity : $1 + 2 + \cdots + (n-1)$

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

$= O(n^2)$

⑥
```
for (i = 2; i<= n; i = pow(i,2))
    printf("%d", i);
```
pow(2,2) : 2^{2^1}

pow(4,2) : $4^2 = 2^{4} = 2^{2^2}$

pow(16,2) : $16^2 = 2^{4^2} = 2^{2^4}$

$2^{2^k} > n \Rightarrow 2^k \log 2 > \log n$

$2^k > \log n$

$k > \log(\log n)$

Time complexity : $O(\log \log n)$

② F(int n) {

```
int i=1, s=1;
while (s<=n) {
    i = i+1;
    s = s+i;
}
}
```

1st	2nd	3rd	4th
i=2	i=3	i=4	i=5
s=3	s=6	s=10	s=15

~~3+6+10+15 ... <= n~~

$S_n = 3 + 6 + 10 + 15 + \cdots + T_n$

$S_n = 0 + 3 + 6 + 10 \cdots + T_{n-1} + T_n$

$T_n = 3 + 3 + 4 + 5 + \cdots (n+1)$

$T_n = 3 + \frac{(n+1)(n+2)}{2}$

$T_n = \frac{(n+1)(n+2)}{2} - 3$

~~$\frac{(k+1)(k+2)}{2} - 3 <= n$~~

i=2	i=3	i=4
s=3	s=6	s=10
]	↓	↳
1+2	1+2+3	1+2+3+4

$\frac{k(k+1)}{2} > n$

$k^2 > n$

$k = O(\sqrt{n})$

③ Fact(n) {

```
if (n<=1)
    return 1; → c
else
    return (n * Fact(n-1))
}
```

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

$T(n) : c + c + c + c + c + \cdots T(n-k)$

$: kc + 1$

$= (n-1)c + 1$

$n - k = 1$

$k = n - 1$

Time complexity: $O(n)$

Q $T(n) = \begin{cases} 1 & n == 1 \\ 2T(n/2) + n & n > 1 \end{cases}$

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

$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$

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

$= 2^3 T\left(\frac{n}{2^3}\right) + 3n \quad = 2^K T\left(\frac{n}{2^K}\right) + Kn$

$= n + \log_2 n$

$\frac{n}{2^K} = 1$

$n = 2^K$

so, time complexity $= O(n \log n)$

Q $T(n) = \begin{cases} 1 & n == 1 \\ 8T(n/2) + n^2 & n > 1 \end{cases}$

Time complexity: $O(n^3)$

$T(n) = 8T(n/2) + n^2$

$= 8\left[8T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right] + n^2$

$= 8^2 + \text{[illegible]}$

$= 8^2 T\left(\frac{n}{2^2}\right) + 3n^2$

$= 8^2\left[8T\left(\frac{n}{8}\right) + \frac{n^2}{16}\right] + 3n^2$

$= 8^3 T\left(\frac{n}{2^3}\right) + 7n^2 \quad = 8^3 T\left(\frac{n}{2^3}\right) + n^2 + 2n^2 + 4n^2$

$= 8^4 T\left(\frac{n}{2^4}\right) + 8n^2 + 4n^2 + 2n^2 + n^2$

$T(n/2): 8T\left(\frac{n}{4}\right) + \frac{n^2}{4}$

$T(n/4): 8T\left(\frac{n}{8}\right) + \frac{n^2}{16}$

$\frac{n}{2^K} = 1$

$n = 2^K$

$S = \frac{a(r^n - 1)}{r - 1}$

$T(n) = 8^K T\left(\frac{n}{2^K}\right) + 2^{K-1} n^2 + \cdots + n^2$

$\frac{2^K - 1}{1} = 2^K - 1$

$= n^3 + n^2(1 + 2 + 2^2 + \cdots 2^{K-1})$

$= n^3 + n^2(n-1) \Rightarrow 2n^3 - n^2$

Q $T(n) = \begin{cases} 1 & n==1 \\ 2T(n/2) + \frac{n}{\log n} & n>1 \end{cases}$

$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n/2}{\log n/2}\right] + \frac{n}{\log n}$

$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n/2}{\log n/2}$

$T(n/4) = 2T\left(\frac{n}{8}\right) + \frac{n/4}{\log n/4}$

$= 2^2 T\left(\frac{n}{2^2}\right) + \frac{n}{\log n/2} + \frac{n}{\log n}$

$= 2^K T\left(\frac{n}{2^K}\right) + \frac{n}{\log \frac{n}{2^{K-1}}} + \cdots + \frac{n}{\log n}$

$\frac{n}{2^K} = 1$

$= n + n\left[\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\log(n-1)} + \frac{1}{\log n}\right]$

$= n + n \log\log n$

Time complexity : $O(n \log\log n)$

Q $T(n) = \begin{cases} 2 & 0 < n \le 2 \\ 2T(\sqrt{n}) + c & n > 2 \end{cases}$

$T(n) = 2T(n^{1/2}) + c$

$= 2\left[2T(n^{1/4}) + c\right] + c$

$= 2^2 T(n^{1/4}) + 2c + c$

$= 2^2\left[2T(n^{1/8}) + c\right] + 2c + c$

$= 2^3 T(n^{1/2^3}) + 2^2 c + 2^1 c + 2^0 c$

$= 2^K T(n^{1/2^K}) + 2^{K-1} c + \cdots 2^1 c + 2^0 c$

$T(n^{1/2}) = 2T(n^{1/4}) + c$

$T(n^{1/4}) = 2T(n^{1/8}) + c$

$n^{\frac{1}{2^K}} = 2$

$\frac{1}{2^K} \log_2 n = 1$

$\log n = 2^K$

$K = \log\log n$

$= 2\log n + c(2^k - 1)$

$= 2\log n + c(\log n - 1)$ Time Complexity $O(\log n)$

$= O(\log n)$

Q $T(n) = \begin{cases} 2 & 0 < n \leq 2 \\ 2T(\sqrt{n}) + \log n & n > 2 \end{cases}$ Time Complexity $O(\log n \cdot \log\log n)$

Time $= 2\log n + \log n \cdot \log\log n$

Tips:→ इस तरह के question में expanded form में ही रहने दो odd कर के मत लिखो असानी से solve हो जायेगा।

→ Masters Theorem

$T(n) = aT(n/b) + \Theta(n^k \log^p n)$

$a \geq 1, b > 1, k \geq 0$ and p is a real number

1) if $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

2) if $a = b^k$ then

a) $p > -1$ then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

b) $p = -1$ then $T(n) = \Theta(n^{\log_b a} \log\log n)$

c) $p < -1$ then $T(n) = \Theta(n^{\log_b a})$

3) if $a < b^k$ then

a) if $p \geq 0$ then $T(n) = \Theta(n^k \log^p n)$

b) if $p < 0$ then $T(n) = \Theta(n^k)$

① $T(n) = 4T(n/2) + n$

$a = 4 \geq 1$, $b = 2 > 1$, $K = 1 \geq 0$, $P = 0$ → real no

$b^K = 2$ $\quad a = 4 \Rightarrow a > b^K$

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4})$

$= \Theta(n^2)$

② $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$a = 2 \geq 1$, $b = 2 > 1$, $K = 1 \geq 0$ $P = -1$ → real no

$b^K = 2$ $\quad a = 2$ $\quad a = b^K$, $P = -1$

$T(n) = \Theta(n^{\log_b a} \log\log n)$

$= \Theta(n \log\log n)$

③ $T(n) = 2T(n/2) + \frac{n}{\log^2 n}$ $\quad a = 2 \geq 1$, $b = 2 > 1$, $K = 1 \geq 0$

$P = -2$ → real no.

$b^K = 2$ $\quad a = b$ $\quad P = -2$ $\quad P < -1$

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

④ $T(n) = 2T(n/2) + n^2$

$a = 2 \geq 1$, $b = 2 > 1$, $K = 2 \geq 0$, $P = 0$ → real no.

$b^K = 4$ [illegible]

$T($[illegible]$) = \Theta($[illegible]$) = \Theta(n \log n)$

$a < b^K$ $\quad P = 0$ $\quad P \geq 1^0$

$T(n) = \Theta(n^K \log^P n)$

$= \Theta(n^2)$

Q Check whether Master Theorem is applicable or not

(i) $T(n) = 2^n T(n/2) + n \rightarrow 2^n$

(ii) $T(n) = 64 T(n/2) - n^2$ $\quad -n^2$

(iii) $T(n) = T(n/2) + \sin n \rightarrow$ यहाँ +, - होता रहेगा

(iv) $T(n) = 2T(n/2) + \frac{1}{n}$ $\quad \frac{1}{n} = n^{-1}$ $k = -1 \not\geq 0$ ✗

(v) $T(n) = 0.5 T(n/2) + n$ $\quad a = 0.5$, $a \not\geq 1$

(vi) $T(n) = \sqrt{2} T(n/2) + \log n$ $\quad a = \sqrt{2} = 1.414 \geq 1$

$b = 2$, $k = 0$, $p = 1$

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

$a = \sqrt{2} \geq 1$, $b = 2 > 1$, $k = 0 \not> 0$, $p = 1$, real no

$b^k = 2^0 = 1$ $\quad a > b^k$ $\quad T(n) = \Theta(n^{\log_2 2^{1/2}})$

$= \Theta(\sqrt{n})$

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

$T(2^m) = T(2^{m/2}) + \log 2^m$ $\quad n = 2^m \Rightarrow m = \log n$

$T(2^m) = T(2^{m/2}) + m$ $\quad T(2^m) = S(m)$

$S(m) = S(m/2) + m \log^0 m$

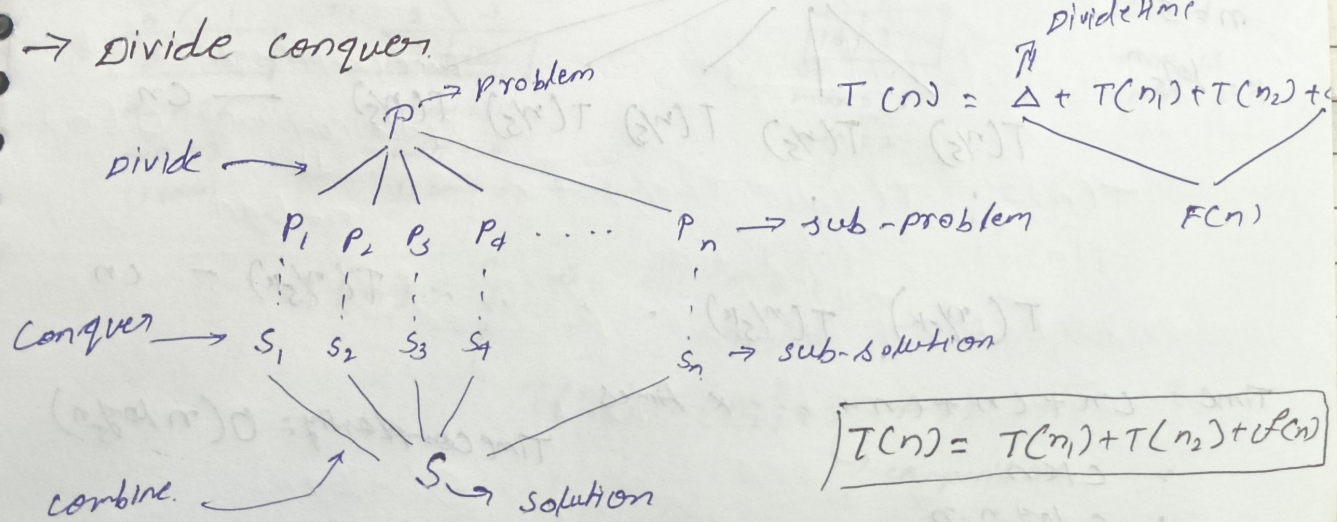
$a = 1 \geq 1$ $\quad b = 2 > 1$, $k = 1 \geq 0$ $\quad p = 0$

$b^k = 2^1 = 2$ $\quad a < b^k$ $\quad p \not> 0$ $\quad T(n) = \Theta(m^k \log^{p+1} m)$

$T(n) = \Theta(m \log^0 m) = \Theta(m)$

$= \Theta(\log n)$

→ Divide conquer

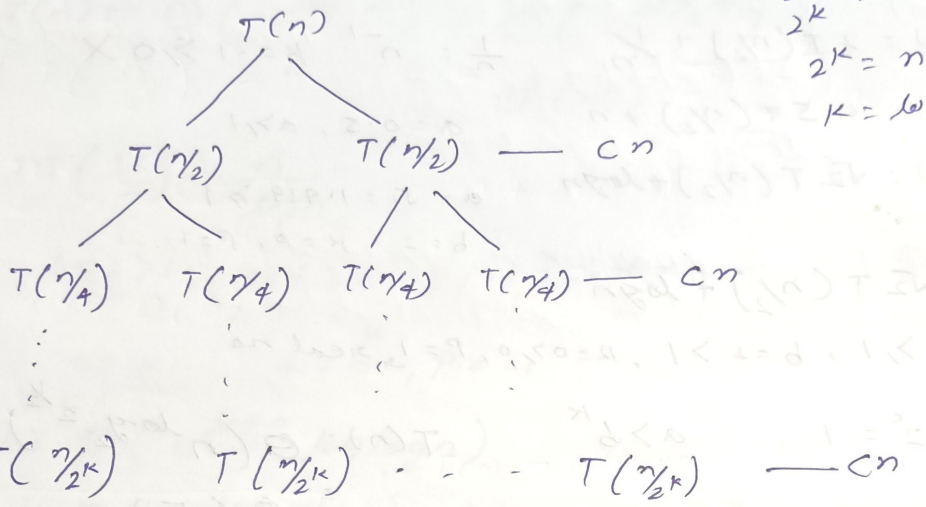


$T(n) = T(n_1) + T(n_2) + cf(n)$

→ Recursive Tree Method (Prefer Back substitution method)

① $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2) + cn \end{cases}$

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



$\frac{n}{2^k} = 1$

$2^k = n$

$k = \log n$

Time = $\underbrace{cn + cn + cn + cn \cdots cn}_{k \text{ times}} = kcn$

$= \log_2 cn$

$= cn \log n$

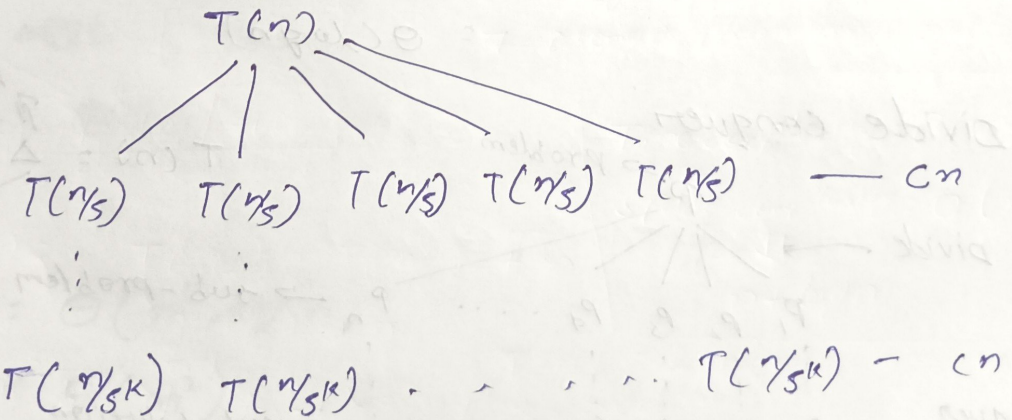
Time Complexity = $O(n \log n)$

② $T(n) = \begin{cases} 1 & \text{if } n=1 \\ 5T(n/5) + cn \end{cases}$

$T(n) = T(n/5) + T(n/5) + T(n/5) + T(n/5) + T(n/5) + cn$

$n = 5^k$

$k = \log_5 n$



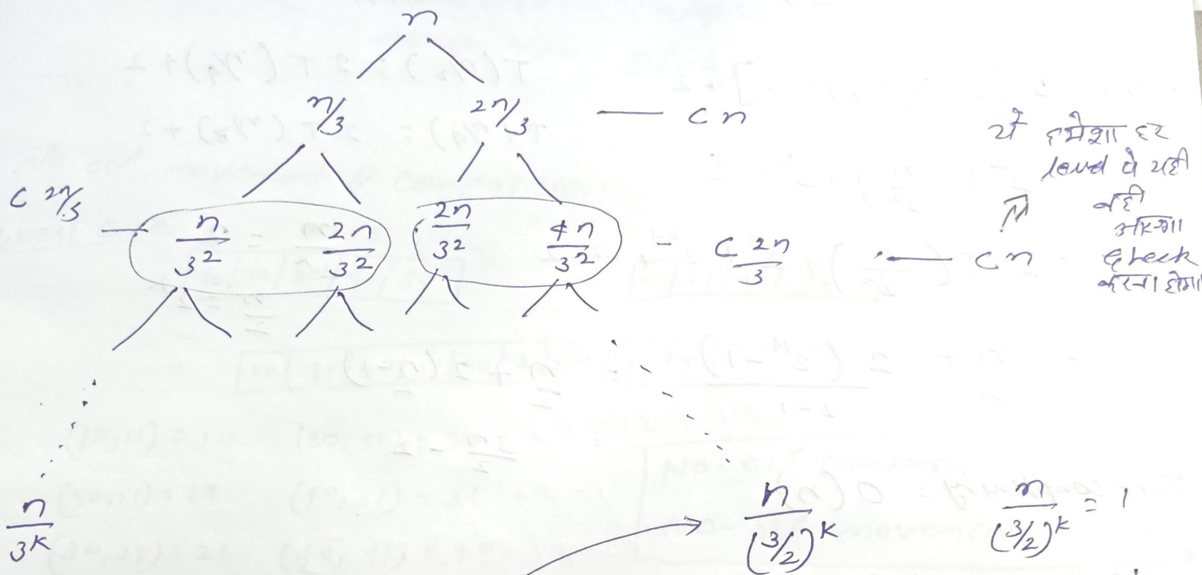
Time = $cn + cn + cn + \cdots k$ times

$= ckn$

$= c \log_5 n \cdot n$

Time complexity = $O(n \log_5 n)$

① $T(n) = T(n/3) + T(2n/3) + cn$



$\frac{n}{(3/2)^k} = 1$

$n = (3/2)^k$

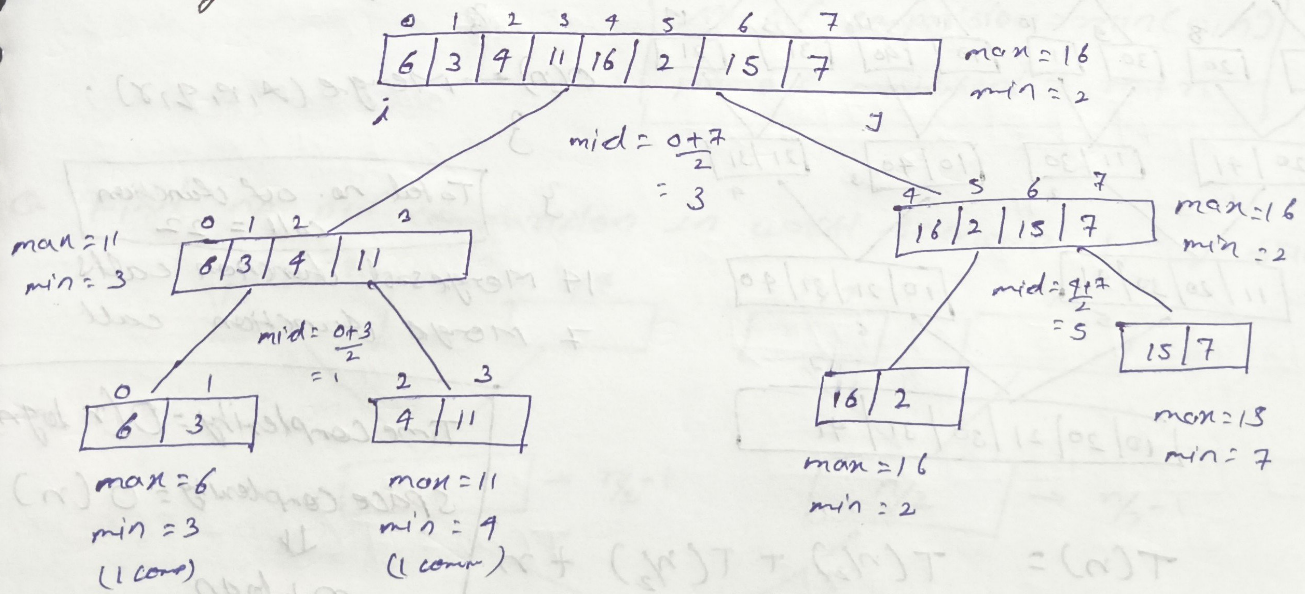
$\log_{3/2} n = k$

Height of tree $= \log_{3/2} n$

so, Time complexity: $kcn = c \log_{3/2} n \cdot n$

$= O(n \log_{3/2} n)$

→ Finding maximum & minimum element using Divide & Conquer



$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(n/2) + T(n/2) + 2 & n > 2 \end{cases}$$

↳ comparision

$T(n) = 2T\left(\frac{n}{2}\right) + 2$ इसपे Master's theorem नहीं लगा पाओगे।

$T(n) = 2\left[2T(n/4) + 2\right] + 2$

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

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

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

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

$T(n/4) = 2T(n/8) + 2$

$\frac{n}{2^k} = 2$

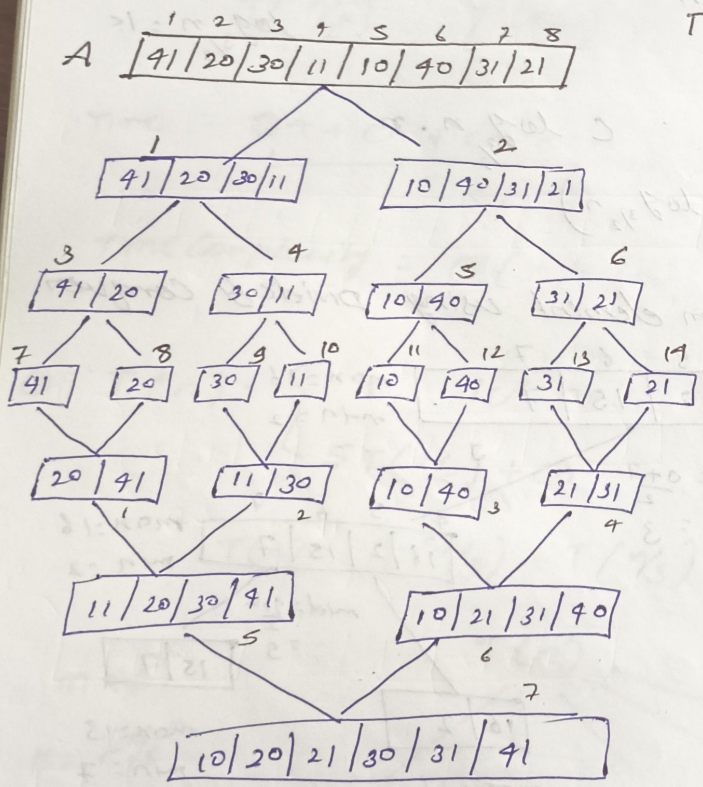
$\frac{n}{2} = 2^k$

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

$= \frac{3n}{2} - 2$

Time complexity = $O(n)$

→ Merge Sort




```
T(n) - MergeSort(A, P, r) {
    if (P < r) {
        c ← q = ⌊(P + r)/2⌋ - floor
        T(n/2) ← MergeSort(A, P, q)
        T(n/2) ← Merge sort(A, q+1, r)
        O(n) ← Merge(A, P, q, r).
    }
}
```

Total no. of function call = 22

14 Mergesort function calls

7 Merge function call

Time complexity = $O(n \log n)$

Space complexity = $O(n)$ ⇓ $n + \log n$

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

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

$a = 2 > 1$ $b = 2 > 1$ $K = 1 > 0$ $P = 0$ → real no

$b^k = 2^1 = 2$ $a = b^k$ $P > -1$

$T(n) = \Theta\left(n^{\log_b a} \log^{P+1} n\right) = \Theta(n \log n)$

Merge sort ——— Best case - $O(n \log n)$
worst case - $O(n \log n)$
Average case - $O(n \log n)$

→ No. of movement & comparision in ~~merge sort~~ Merging

worst case

m = 5				
10	20	30	40	50

n = 4			
11	21	31	41

10	11	20	21	30	31	40	41	50	

(10,11) = 10 (30,31) = 30
(20,21) = 20 (40,31) = 31
(30,21) = 21 (40,41) = 40
(20,11) = 11 (50,41) = 41

No. of comparision = m+n-1
No. of movement = m+n

Best case

m			
10	20	30	40

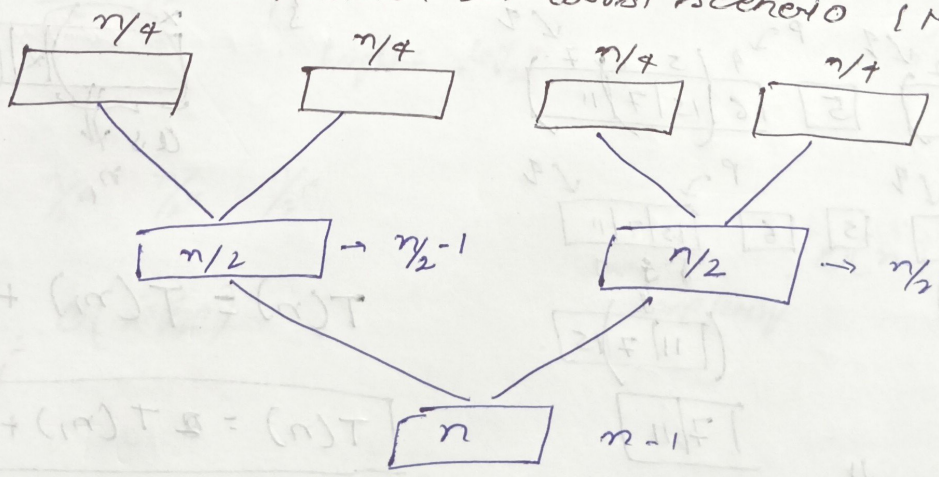
n				
50	60	70	80	90

(10,20) = 10 (30,50) = 30
(20,50) = 20 (40,50) = 40

10	20	30	40	50	60	70	80	90

No. of comparision = min(m,n)
No. of movement = m+n

Q Find no. of comparision in worst scenario (Merge करना)



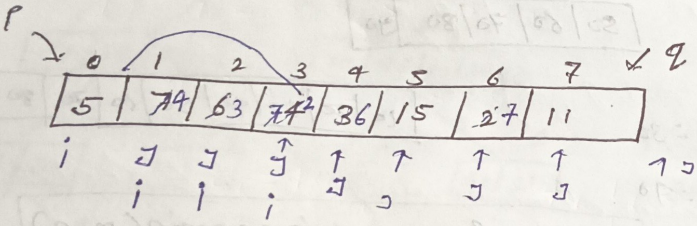
No. of comparision = $\frac{n}{2} - 1 + \frac{n}{2} - 1 + n - 1$

$= 2n - 3$

→ Quick sort

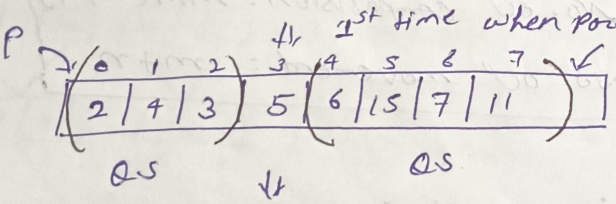
Pivot element को पकड़ते हैं और उसको उसकी सही जगह पर पहुँचा देते हैं।

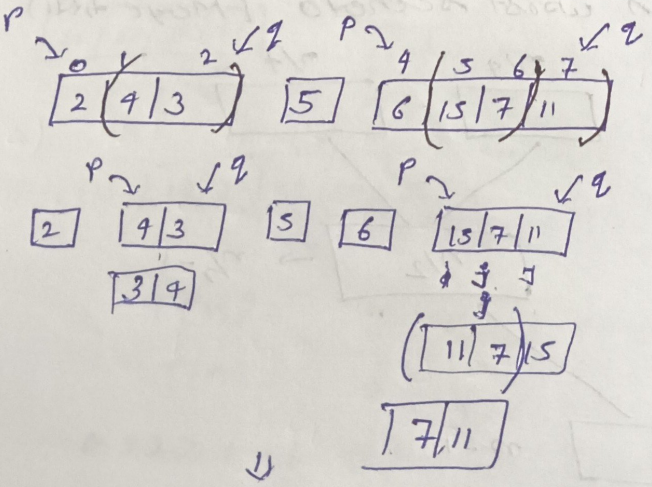
```
Partition (A, P, q) {
    X = A[P];
    i = P;
    for (J = P+1 ; J ≤ q ; J++) {
        if (A[J] ≤ X) {
            i = i+1;
            swap(A[i], A[J])
        }
    }
    swap(A[i], A[P])
}
```

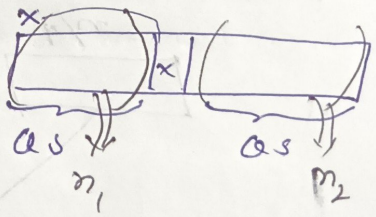



```
Quicksort (A, P, q) {
    if (P < q) {
        m = Partition (A, P, q)
        quicksort (A, P, m-1)
        quicksort (A, m+1, q)
    }
}
```

O(n)







$$T(n) = T(n_1) + T(n_2) + O(n)$$

$$T(n) = T(n_1) + T(n_2) + cn$$

2, 3, 4, 5, 6, 7, 11

$n_1 = n_2 = n/2$ * Partition कैसे हो रहा है कि half-half part में divide करे।

$\boxed{T(n) = 2T(n/2) + cn}$ ⇒ Time Complexity : $O(n \log n)$ ⇓ Best case

worst case :- → Array जब sorted order में हो

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

$= T(n-2) + cn - c + cn$ $\quad T(n-1) = T(n-2) + c(n-1)$

$= T(n-3) + cn - 2c + cn - c + cn$ $\quad T(n-2) = T(n-3) + cn - 2c$

$= T(n-3) + c(n-2) + c(n-1) + cn$ $\quad n-k = 0$, $n = k$

$= T(n-k) + c(n-k+1) + \cdots + c(n-1) + cn$

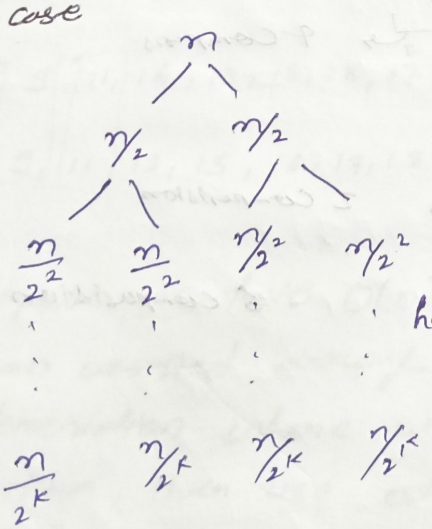
$= c[1 + 2 + 3 + \cdots + n-1 + n]$

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

Time Complexity : $O(n^2)$

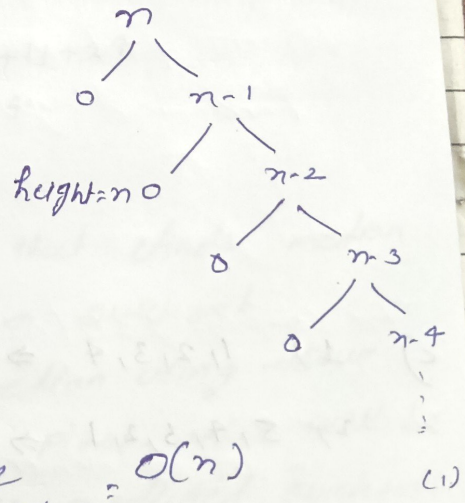
For space complexity :-

Best case



space complexity : $O(\log_2 n)$

worst case



space complexity : $O(n)$

(1)

Q) Quicksort is run on two input shown below to sort in ascending order

a) 1) 1, 2, 3, 4 ---- (n-1), n ⇒ C_1

2) n, (n-1), ---- 3, 2, 1 ⇒ C_2

a) $C_1 < C_2$ b) $C_1 > C_2$

✓ c) $C_1 = C_2$ d) NOT

1, 2, 3, 4, --- (n-1), n

5, 4, 3, 2, 1 → 4 comparision

1, 4, 3, 2, → 3 comparision

4 3 2 → 2 comm

3 2 → 1 comparision

4+3+2+1

10 comparision

1, 2, 3, 4, 5 → 4

2, 3, 4, 5 → 3

3, 4, 5 → 2

4, 5 → 1

No. of comparision = 4+3+2+1 = 10

दोनों case में No. of comparision same है।

b) 1) 1, 2, 3, 9, 5 ⇒ C_1

2) 4, 1, 5, 3, 2 ⇒ C_2

a) $C_1 < C_2$ ✓ b) $C_1 > C_2$

c) $C_1 = C_2$ d) NOT

$C_1 = 10$

4, 1, 5, 3, 2 → 4 comparis.

2, 1, 3 → 2 comparision

(1) 2 (3) 6 comparision

c) 1) 1, 2, 3, 4 ⇒ C_1

2) 5, 4, 3, 2, 1 ⇒ C_2

1, 2, 3, 4 : 3+2+1

C_1 = 6

C_2 = 5, 4, 3, 2, 1

: 4+3+2+1

= 10

$C_1 < C_2$

Q suppose we are sorting an array of 8 integers using Quicksort and we have just finished the first Partitioning with the array looking like this

2 5 1 7 9 12 11 10

2, 5, 1, [7], [9], 12, 11, 10

1, 2, 5, [7], [9], 10, 11, 12

True

The pivot could be

a) either 7 or 9

b) 7 but not 9

c) not 7 but 9

d) Neither 7 nor 9

Q Suppose we are sorting an array of eight integers using Quicksort and we have just finished the first partitioning with the array looking like this. Find sum of all possible values that could have been used as a pivot

9, 11, 16, 13, 15, 18, 17, 24

Sol. [9], [11], 16, 13, 15, 18, 17, [24]

[9], [11], 13, 15, 16, 17, 18, [24]

sum = 9+11+24

= 44

Q Suppose we have O(n) time algorithm that finds median of an unsorted array. Now consider a Quicksort implementation where we first find median using above algorithm, then use median as pivot. What will be the worst case time complexity of this modified Quicksort.

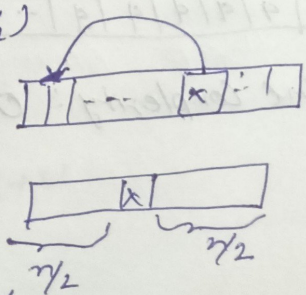
Sol. $T(n) = O(n) + O(1) + O(n) + T(n/2) + T(n/2)$

Median find किया

Swap किया First element से

Partition लगाया

दो equal part में divide हो गया



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

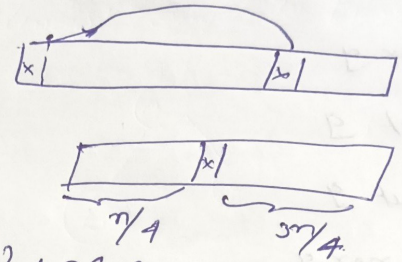
Q. In Quicksort, for sorting n elements, the $(n/4)^{th}$ smallest element is selected as pivot using an $O(n)$ time algorithm what is worst time complexity of the Quicksort

a) $\Theta(n)$

b) ✓ $\Theta(n \log n)$

c) $\Theta(n^2)$

d) $\Theta(n^3)$



$$T(n) = O(n) + O(1) + O(n) + T(n/4) + T(3n/4)$$
$$= O(n) + 2T(n/4) + T(3n/4)$$
$$\therefore \Theta(n \log_{4/3} n)$$

→ Randomized quicksort

Quicksort में worst case में Time complexity $O(n^2)$ आया था। उसी को reduce करेंगा Randomised quick sort.

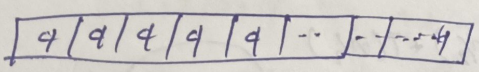
यहाँ पर pivot element किसी को भी select कर लेंगे। through Randomised generating function

```
RQS(A, P, q) {
    if (P < q) {
        r = RGF(P, q)        ← random pivot choose करेंगा
        swap(A[P], A[r])     ← First element को swap कर दिया
        m = partition(A, P, q)   ← अब quicksort लगा दिया
        RQS(A, P, m-1)       ← same as quick sort
        RQS(A, m+1, q)
    }
}
```

Time complexity = $O(n \log n)$

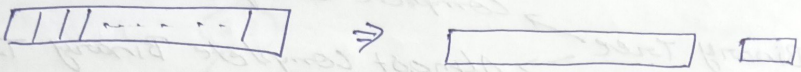
worst case

↳ जब सारे element same हो तब



Time complexity: $O(n^2)$

Q An array of 25 distinct element is to be sorted using quicksort. Assume that the pivot element is chosen uniformly at random. The probability that the pivot element gets placed in the worst possible location in the first round of partitioning (rounded off to 2 decimal places) is —

Sol. 

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

ये तभी होगा जब max या min element pivot first round में

$$\therefore \text{probability} = \frac{2}{25} = 0.08$$

→ Matrix Multiplication

- Normal method $O(n^3)$
- Divide & conquer $O(n^3)$

$$T(n) = 8T(n/2) + 4n^2$$

Normal method ज्यादा अच्छा है because space complexity कम है कोई extra space नहीं लिए Divide & conquer में extra space occupie करेंगी stack

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \quad \begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix}$$

Strassen's Matrix Multiplication

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$Q = (A_{21} + A_{22}) B_{11}$$

$$R = A_{11} (B_{12} - B_{22})$$

$$S = A_{22} (B_{21} - B_{11})$$

$$T = (A_{11} + A_{12}) B_{22}$$

$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P + S - T + V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

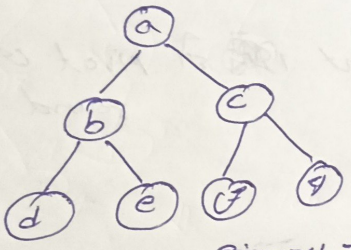
$$C_{22} = P + R - Q + U$$

$$T(n) = 7T\left(\frac{n}{2}\right) + 18n^2$$

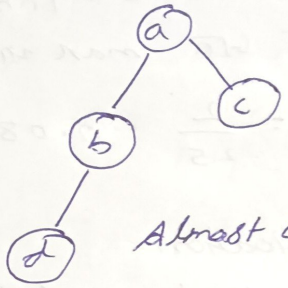
$$O(n^{2.81})$$

Heap

Binary Heap ⇒ Binary Tree → complete Binary Tree
Binary Tree → Almost complete Binary Tree



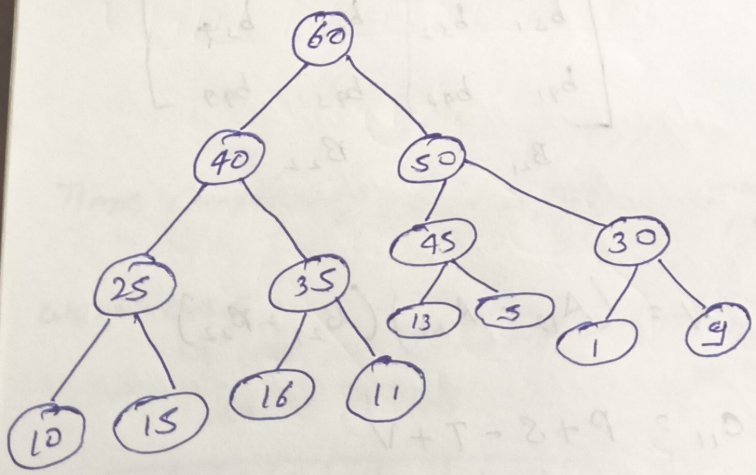
Complete Binary Tree



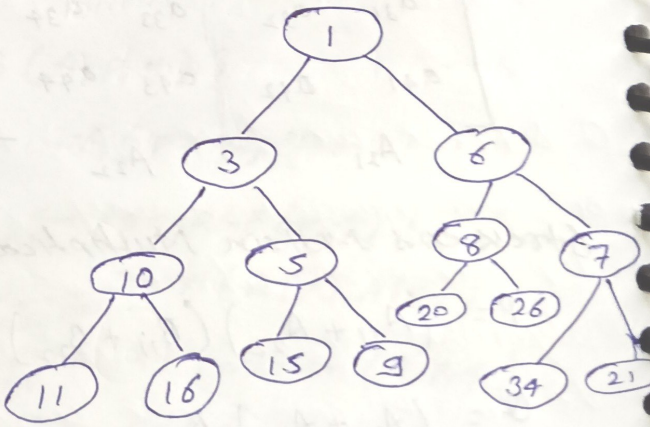
Almost complete Binary Tree

Array representation करने से tree का array full है तो Binary Heap है नहीं तो Binary heap नहीं है।

Max heap
Root node maximum है subtree में Root node maximum है.



Min heap
Root node minimum है subtree में root node minimum है

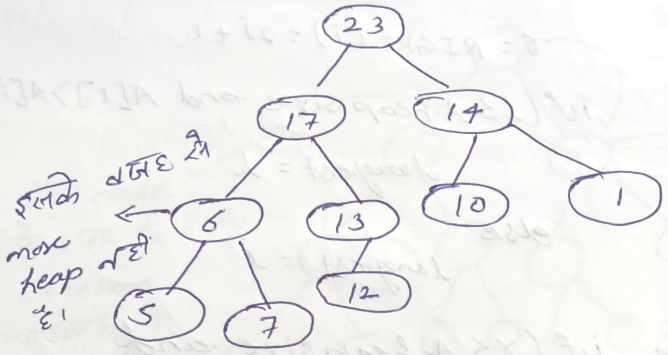


Heap, Binary search Tree से अलग है

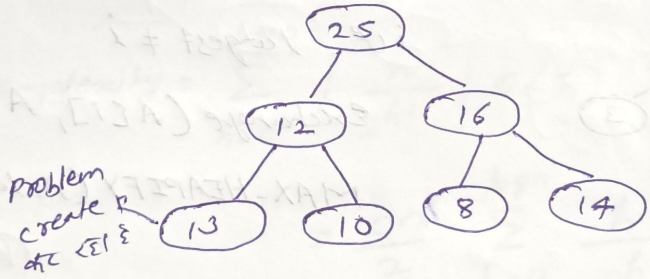
Min heap में minimum element find करने में constant time लगेगा
Max heap में maximum element find करने में constant time लगेगा।

Check whether given Tree is Binary heap or not

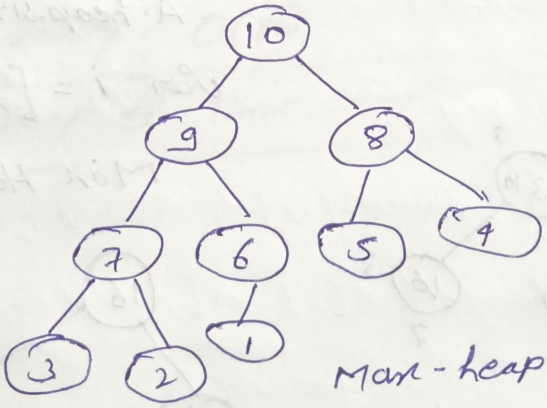
1) 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 ✗



2) 25, 12, 16, 13, 10, 8, 14 ✗



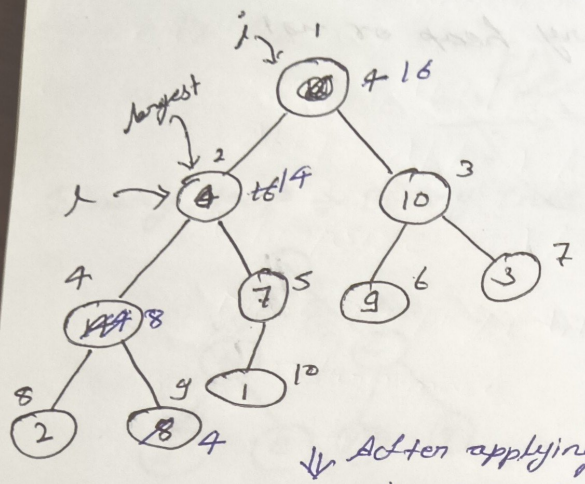
3) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 ✓



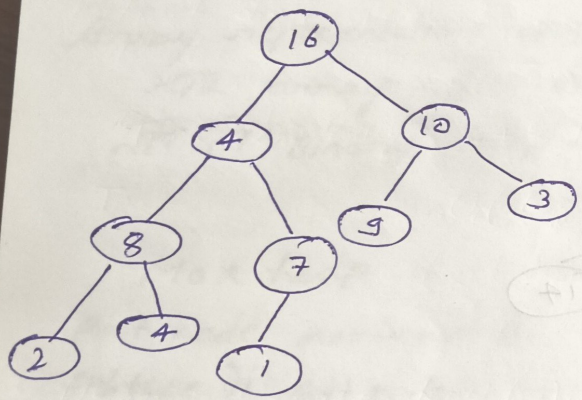
Max-heap है

→ Max Heapify

Max heapify function की
Time complexity $O(\log n)$



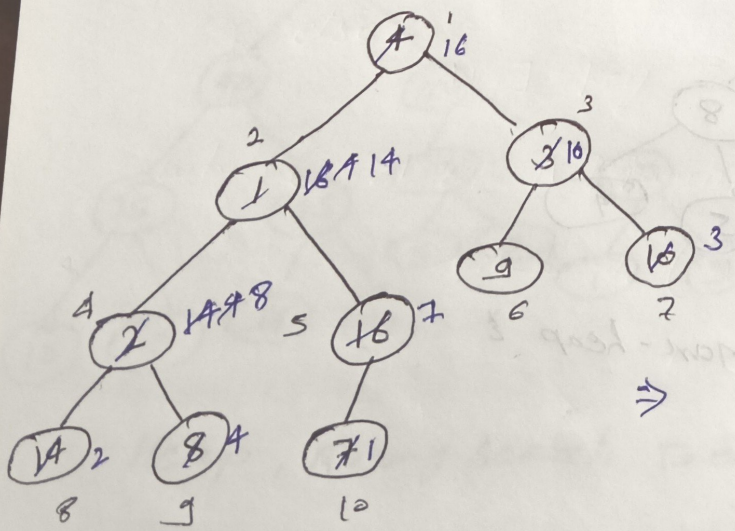
⇓ After applying heapify function



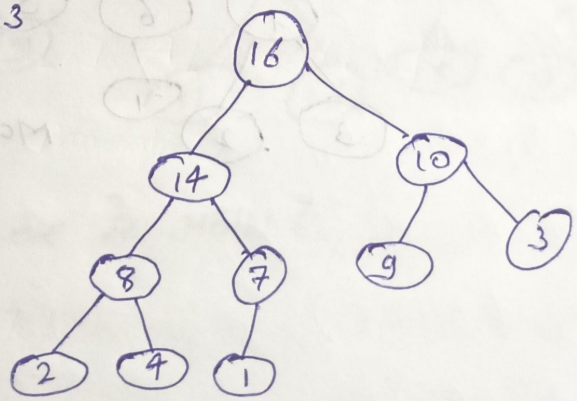

```
MAX - HEAPIFY(A,i)
    l = LEFT(i) = 2i
    r = RIGHT(i) = 2i+1
if (l ≤ A.heapsize and A[l] > A[i])
        largest = l
    else
        largest = i
if (r ≤ A.heapsize and
                A[r] > A[largest])
        largest = r
if largest ≠ i
    exchange(A[i], A[largest])
    MAX-HEAPIFY(A, largest)
```

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7

```
Build Max Heap(A)
    A.heapsize = A.length
for i = [A.length/2] down to 1
    Max Heapify(A,i)
```

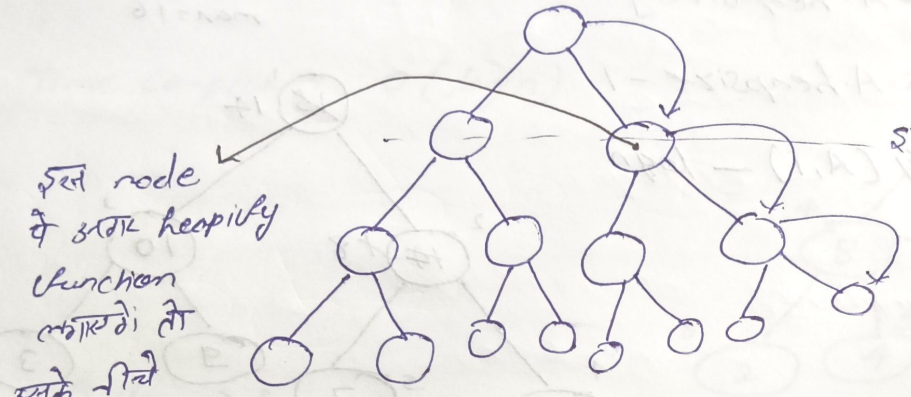


⇒



→ Given array को descending order में sort कर देंगे और tree construct कर देंगे तो Max heap बन जाएगा
इस approach से time complexity $O(n \log n)$

→ No. of node in level जिसकी height h है $= \left\lceil \frac{n}{2^{h+1}} \right\rceil$ ceil



Time complexity $= \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = \sum_{h=0}^{\log n} \frac{n}{2^h * 2} * c \cdot h$

$= \frac{cn}{2} \sum_{h=0}^{\log n} \frac{h}{2^h}$

$\leq \frac{cn}{2} \left(\sum_{h=0}^{\infty} \frac{h}{2^h} \right) \Rightarrow 2$

∴ Time complexity : $O(n)$

∴ Build Max-Heap की Time complexity $O(n)$

Heap Extract max (A)

{

if (A.heapsize < 1)

"Head underflow" ⇒ max element को delete कर रहे हैं तो error देगा।

else

max = A[1]

A[1] = A[A.heapsize]

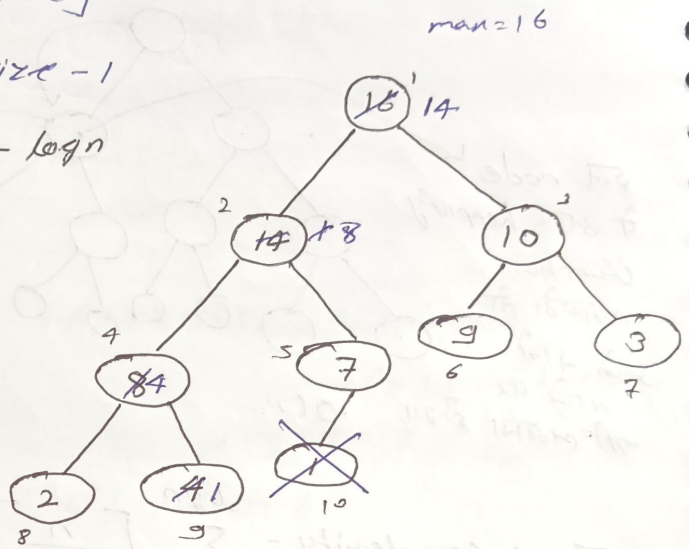
A.heapsize = A.heapsize - 1

Max Heapify (A, 1) — logn

return max

}

जैसे हम max element extract करने जायेंगे।



Time complexity = c + logn

= O(logn)

Heap Increase key (A, i, key) {

if (key < A[i])

"error"

else

A[i] = key

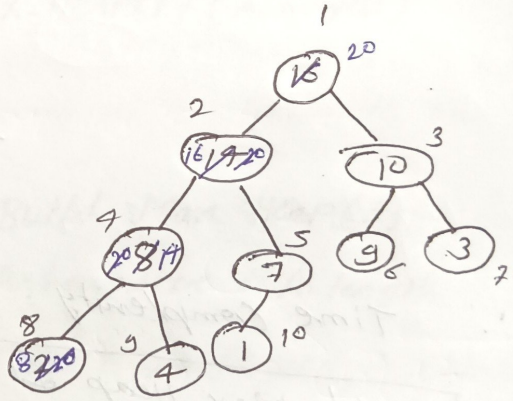
while (i > 1 && A[parent(i)] < A[i]) {

exchange (A[i], A[parent(i)])

i = parent(i)

}

}



Time complexity : O(logn)

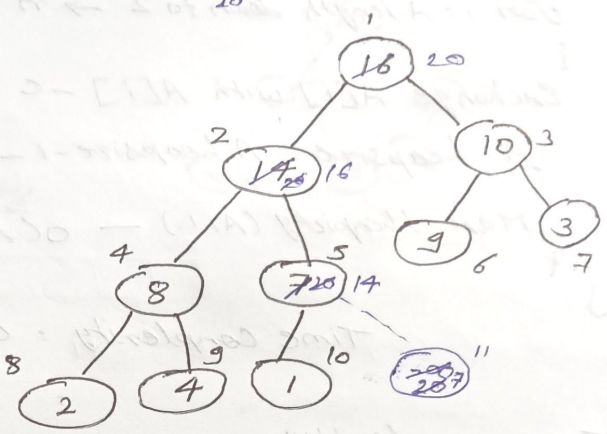
Max Heap Insert (A, key) {

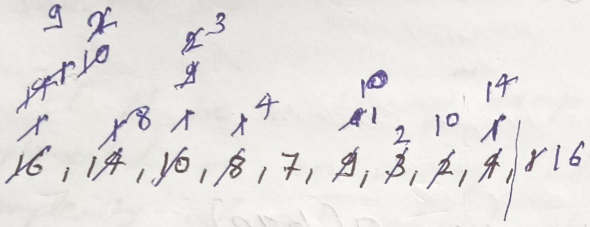
A.heapsize = A.heapsize + 1

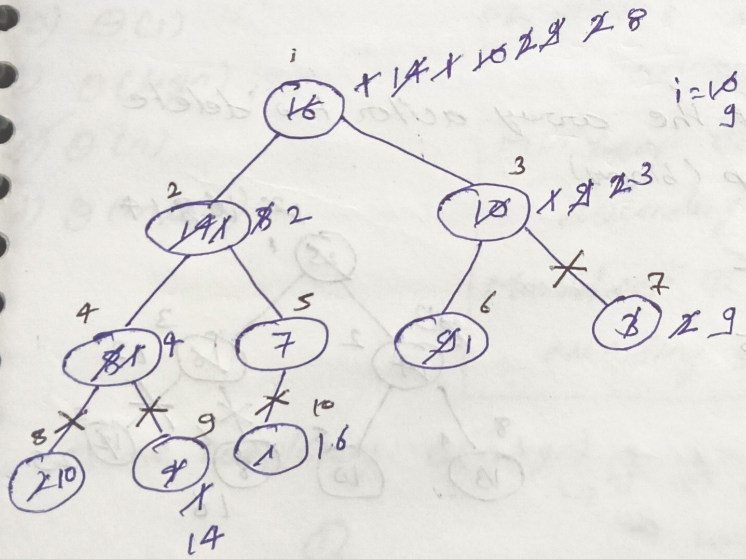
A[A.heapsize] = $-\infty$

Heap Increase key (A, A.heapsize, key)

Time complexity = O(log n)







Heap Sort (A) {

Build Max Heap (A)

for i = A.length down to 2

{

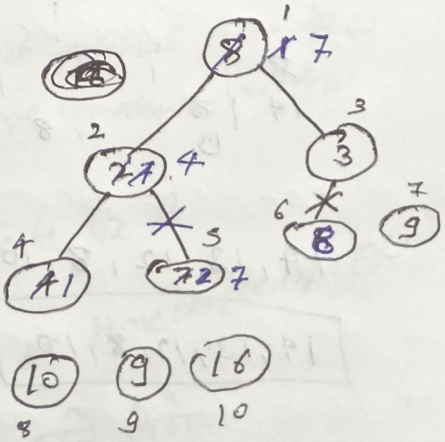
Exchange A[1] with A[i]

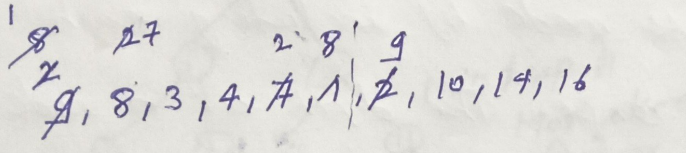
A.heapsize = A.heapsize - 1

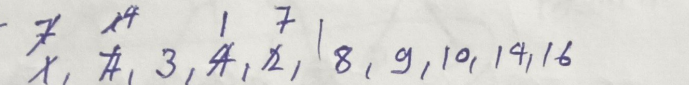
Max Heapify (A, 1)

}

}

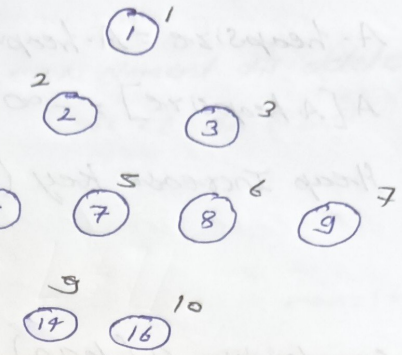






1 iteration में max element unsorted array की अपनी exact position पर जा रहा है।

1 | 2 | 3 | 4 | 7 | 8 | 9 | 10 | 14 | 16



```
Heap sort (A) {
    Build Max heap(A) — O(n)
  for i = A.length down to 2 → n
  {
  Exchange A[1] with A[i] — c
    A. heapsize = A.heapsize - 1 — c
    Max Heapify (A, 1.) — O(logn)
  }
}
```

Time complexity : $O(n) + O(\log n)\, n$

: $O(n \log n)$

Time complexity:-

Build_Max_heap — $O(n)$

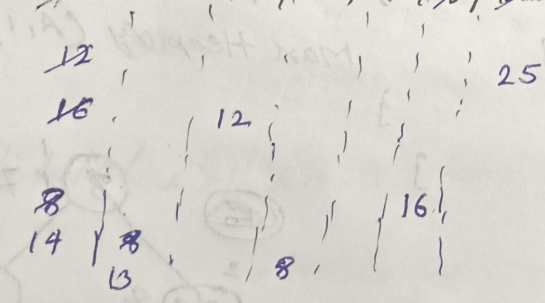
Max_heapify — $O(\log n)$

Insertion, deletion, Modify particular value — $O(\log n)$

Heap sort — $O(n \log n)$

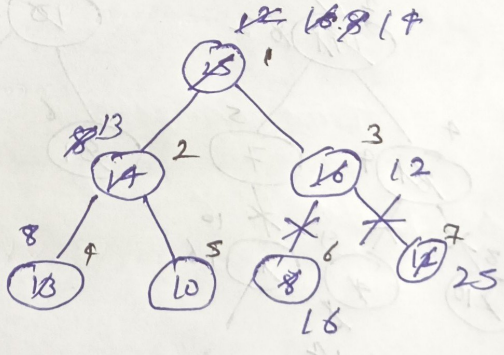
Q what is the content of the array after two delete operation on max-heap (binary)

Array: ~~25~~, 14, ~~14~~, 13, 10, ~~8~~, ~~12~~



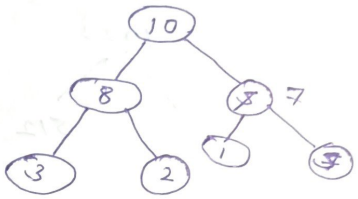
14, 13, 12, 8, 10, ~~16~~, ~~25~~

14, 13, 12, 8, 10



Q A priority queue is implemented as a Max heap, initially it has 5 elements. The level order traversal of the heap is 10, 8, 5, 3, 2. Two new element 1 & 7 are inserted into the heap in that order. The level order traversal of the heap after the insertion of the element is

Solution: 10, 8, 7, 3, 2, 1, 5



Q Lowest worst case complexity

a) Merge sort - $n \log n$
b) Quick sort - n^2
c) Bubble sort - n^2
d) Selection sort - n^2

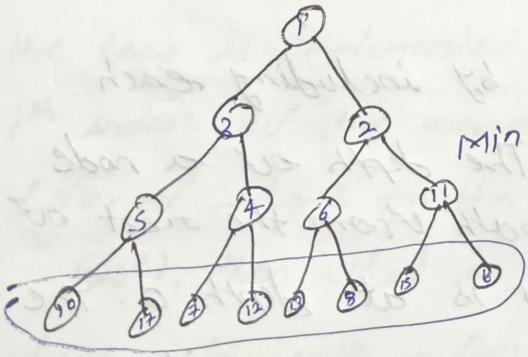
Q Let H be a binary min-Heap consisting of n-elements implemented as an array. what is the worst case time complexity of an optimal algorithm to find the maximum element.

a) $\Theta(1)$
b) $\Theta(\log n)$
c) $\Theta(n)$
d) $\Theta(n \log n)$

एक तरीका है sort करके array find कर ले maximum

Min-heap पर Heap sort लगाएंगे तो descending order में arrange हो जाएगा
Max-heap पर Heap sort लगाएंगे तो ascending order में arrange हो जाएगा

Time complexity: $\Theta(n \log n)$ ↳ यह हो सकती है but optimal case की ⇒ worst case Time complexity चाहिए



Min heap में Max element leaf node पर ही मिलेगा

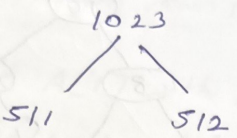
∴ Time complexity: $\Theta(n)$

जितने leaf node है उतने comparision करने होंगे

No. of leaf node $\lceil n/2 \rceil$ ⇒ $n/2+1$ to n

Q Consider the array representation of a binary ~~heap~~ min heap containing 1023 elements. The minimum number of comparision required to find the maximum in the heap —

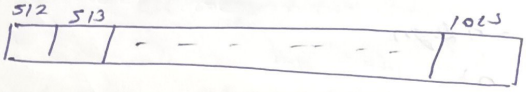
sol $n = 1023 = 2^{10} - 1$ ⇒ complete binary tree है.



1 to $\left\lfloor \frac{n}{2} \right\rfloor$ ⇒ Internal node
↳ Floor

$\left\lfloor \frac{1023}{2} \right\rfloor = 511$

$\left\lfloor \frac{n}{2} \right\rfloor + 1$ to n ⇒ Leaf node

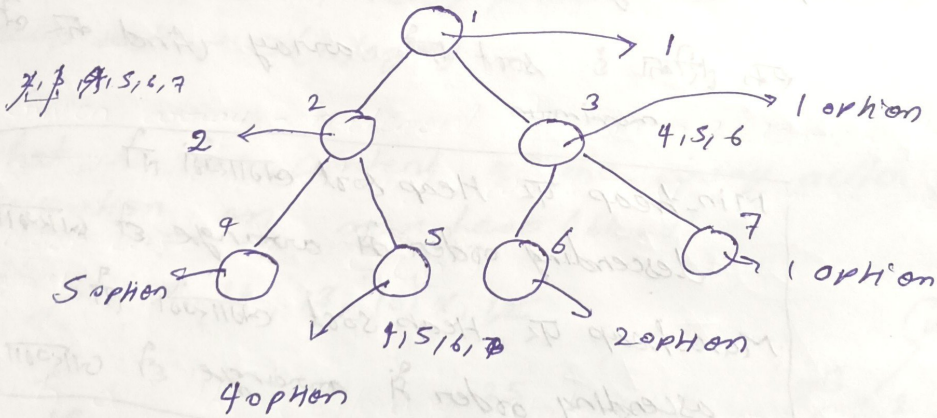


So, No. of comparision is 511

index 512 वाले को पकड़ के बाकी element से comparision करेंगे।

Q The number of possible min heaps containing each value from $\{1,2,3,4,5,6,7\}$ exactly once is —

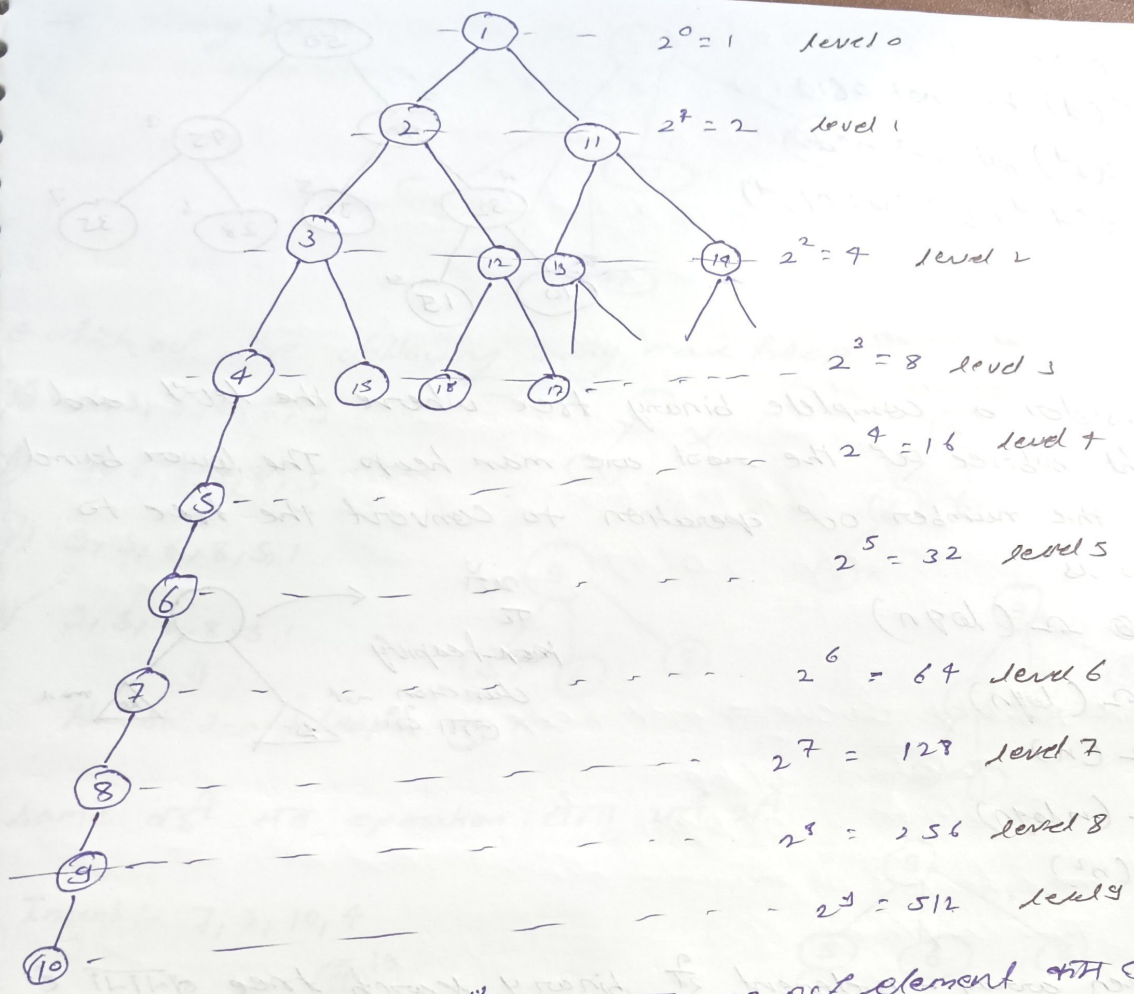
sol



No. of possible min heap: $1 \times 2 \times 5 \times 4 \times 2 \times 1 \times 1$

$= 80$

Q A complete binary Heap is made by including each integer in $[1, 1023]$ exactly once. The depth of a node in the heap is the length of the path from the root of the heap to that node. Thus, the root is at depth 0. The maximum depth at which integer 9 can appear is —



so, depth = 8

यहाँ पर अगर no. of element काम होता तब depth = 8 नहीं आता because 9 को अगर depth = 8 पर place कर देते तब binary Heap की property violate हो जाती

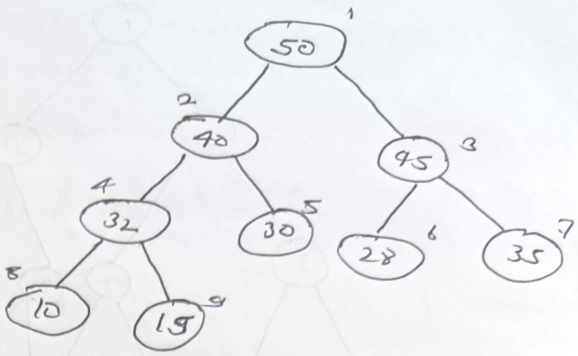
Q An operator delete(i) for binary heap data structure is to be designed to delete the item in the i^{th} node. Assume that the heap is implemented in an array and i refers to the i^{th} index of the array. If the heap tree has depth d (No. of edges on the path from the root to the farthest leaf), then what is the time complexity to refix the heap efficiently after the removal of the element.

a) $O(1)$

b) $O(d)$ but not $O(1)$

c) $O(2^d)$ but not $O(d)$

d) $O(d2^d)$ but not $O(2^d)$



Q consider a complete binary tree where the left and right subtree of the root are max heap. The lower bound for the number of operation to convert the tree to heap is

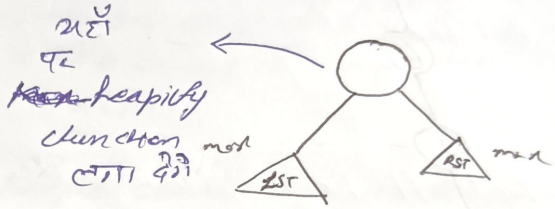
Sol ⊗ $\Omega(\log n)$

a) $\Omega(\log n)$

b) $\Omega(n)$

c) $\Omega(n \log n)$

d) $\Omega(n^2)$



→ Given array - n element से binary search tree बनाना है इसकी time complexity ~~worst~~ Best case में $\Theta(n \log n)$

~~Best~~ worst case में $\Theta(n^2)$

Q which of the following statement are True?

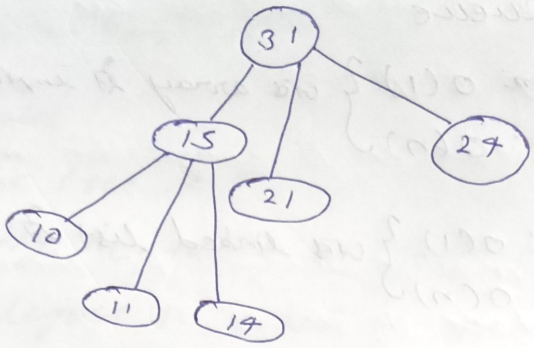
a) The smallest element in a max-heap is always at a leaf node

b) The second largest element in a max-heap is always a child of root node

c) A max-heap can be constructed from binary search tree in $\Theta(n)$ time

d) A binary search tree can be constructed from a max heap in $\Theta(n)$ time

→ 3-ary heap



Q which of the following 3-ary max heap.

a) 1, 3, 5, 6, 8, 9

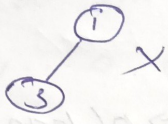
b) 9, 6, 3, 1, 8, 5

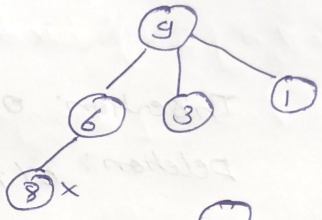
c) 9, 5, 6, 8, 5, 1

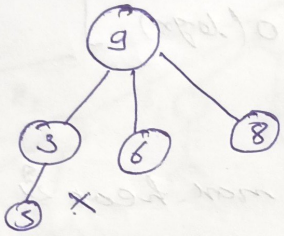
d) 9, 5, 6, 8, 3, 1

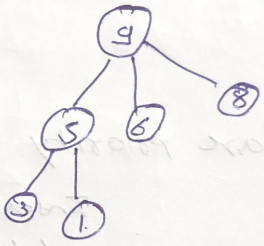
⇓

Almost 3-ary heap



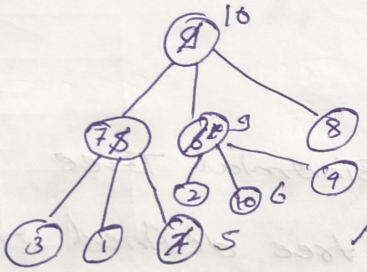


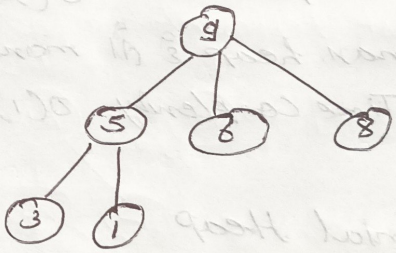




same वही सब operation होगा यहाँ भी

Insert :- 7, 2, 10, 4





After Insertion level wise traversal becomes: 10, 7, 9, 8, 3, 1, 5, 2, 6, 4

सब कुछ Binary Heap की तरह है यहाँ पर बस at most 2 children की जगह यहाँ almost 3 children है।

→ Priority Queue

→ max priority queue → जो maximum element है उसकी priority सबसे ज्यादा होगी।

→ min priority queue → min वाले की priority सबसे ज्यादा

Max priority Queue

Insertion: O(1) } जब array से implement करेंगे
Deletion : O(n) }

Insertion : O(1) } जब linked list से करेंगे।
Deletion : O(n) }

O(height) [Insertion : O(n) } जब Binary search Tree से करेंगे।
Deletion : O(n) }

Insertion : O(h) ⇒ O(logn) } जब AVL Tree से करेंगे। या Red Black tree से
Deletion : O(h) ⇒ O(logn) }

Max priority Queue ⇒ max-heap से बनाते है

Insertion: O(logn)
Deletion : O(logn)

max. heap है तो maximum element find करने का Time complexity O(1)

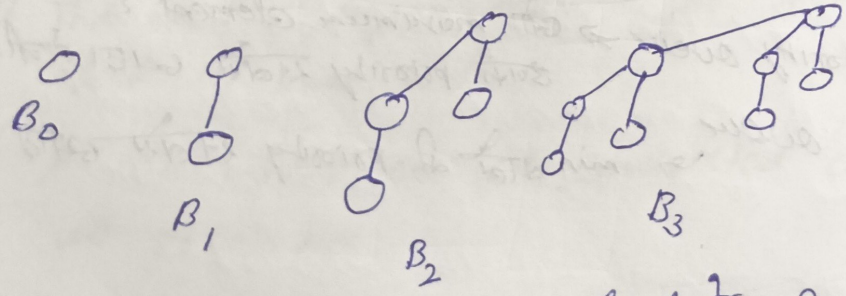
→ Binomial Heap

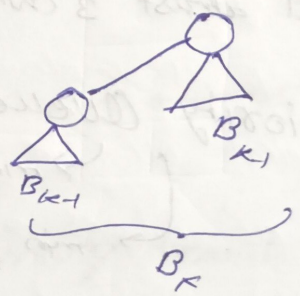
Binomial Heap is a collection of Binomial Tree

The Binomial Tree B_k is an ordered tree defined recursively

The Binomial Tree B_0 consist of a single node

The Binomial Tree B_k consist of two Binomial Tree B_{k-1} that are linked together

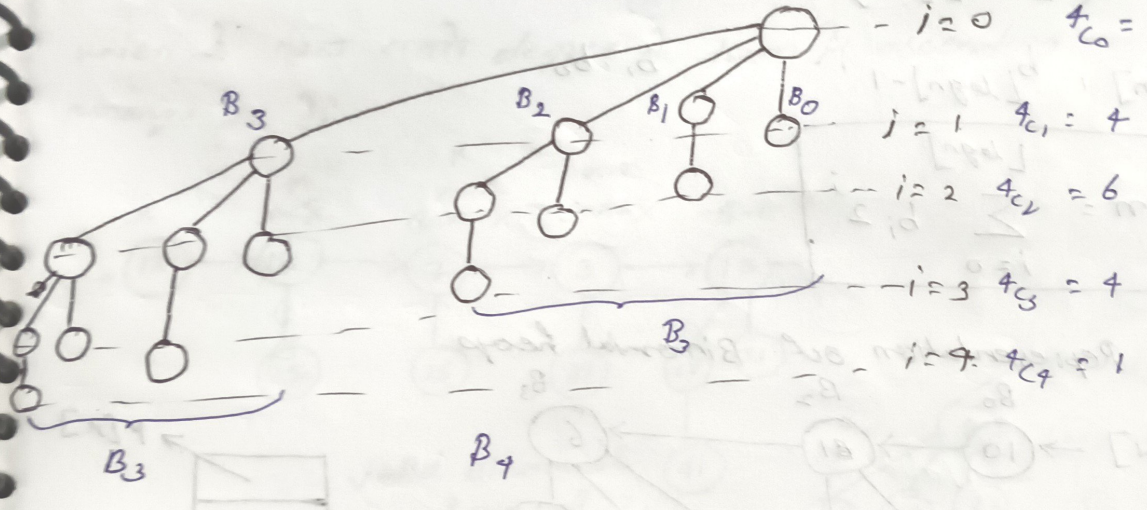




Leaf में ही करते है ये ध्यान रहना

Properties of Binomial Tree (B_k)

1) There are 2^k nodes
2) The height of the tree is k
3) There are exactly ^kC_i nodes at depth i for $i = 0, 1, \cdots k$
4) The root has degree k, which is greater than that of any other node.
5) if i the children of the root are numbered from left to right by $k-1, k-2, \cdots 0$. child i is the root of a subtree B_i



→ n = no. of node given

n = 13 ⇒ 1 1 0 1
B_3 B_2 B_1 B_0
✓ ✓ × ✓

ऐसा Binomial Heap बनाना है जिसके 13 node हो।
उसमें include B_0, B_2, B_3 Tree

n = 15 1 1 1 1 ⇒ B_0, B_1, B_2, B_3 binary Heap use करके बनाएँगे।
B_3 B_2 B_1 B_0

1 1 1 1 ⇒ इसमें no. of node कितना होगा। simple है
Tree use हुआ है B_3 B_2 B_1 B_0

$2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1$
$= 15$

A binomial heap H is a set of binomial tree that satisfies the following heap properties

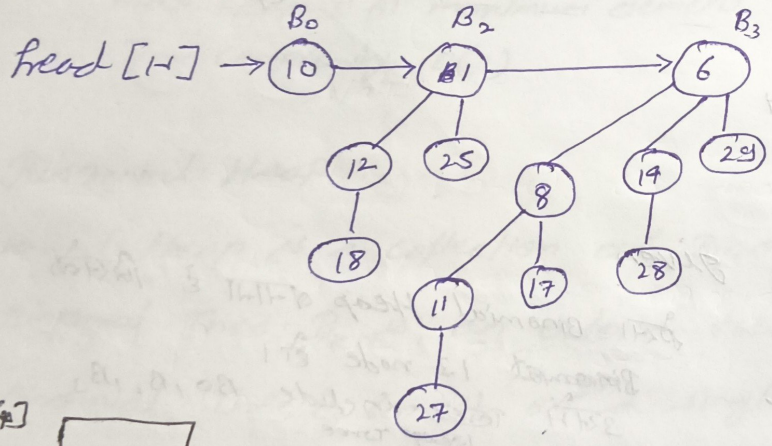
1) Each binomial tree in H obeys the min/max heap property
2) For a non-negative integer k, there is atmost one binomial tree B_k in H

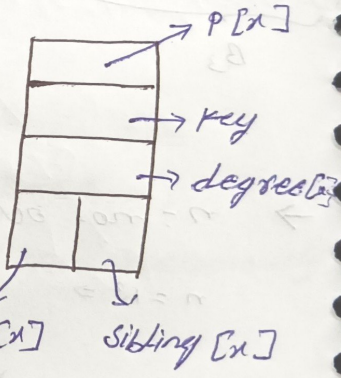
An n-node binomial heap H consist of atmost $(\lfloor \log_2 n \rfloor + 1)$ binomial trees because binary representation of n has $(\lfloor \log n \rfloor + 1)$ bits

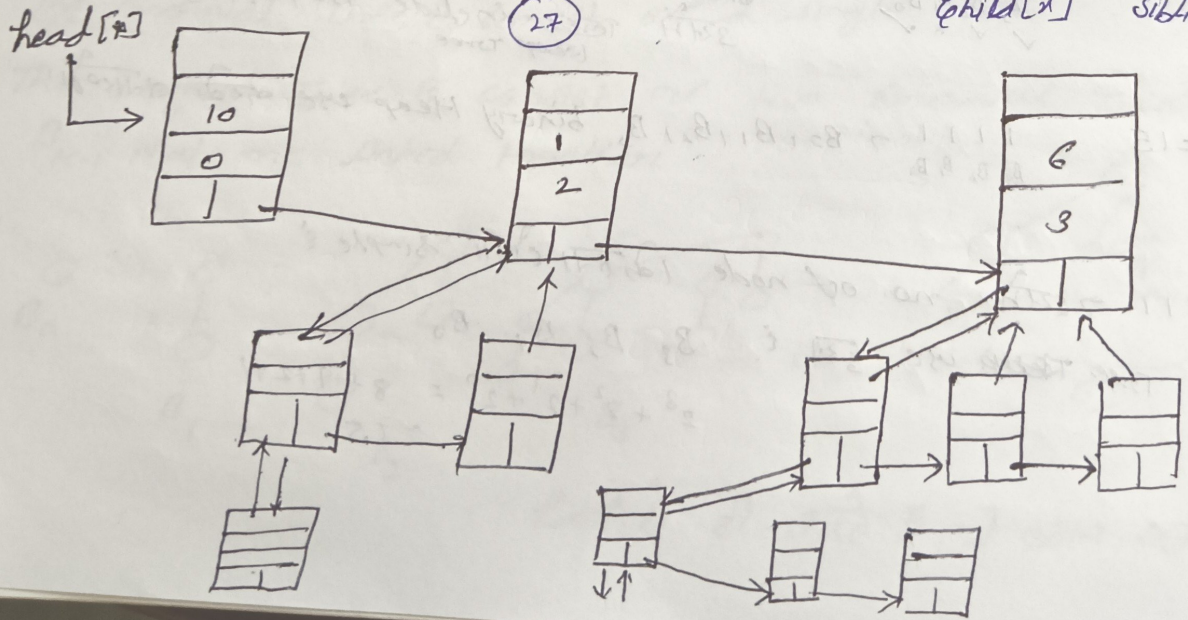
$b_{\lfloor \log n \rfloor}, b_{\lfloor \log n \rfloor - 1}, \ldots, b_1, b_0$

$$n = \sum_{i=0}^{\lfloor \log n \rfloor} b_i 2^i$$

Linked list Representation of Binomial heap

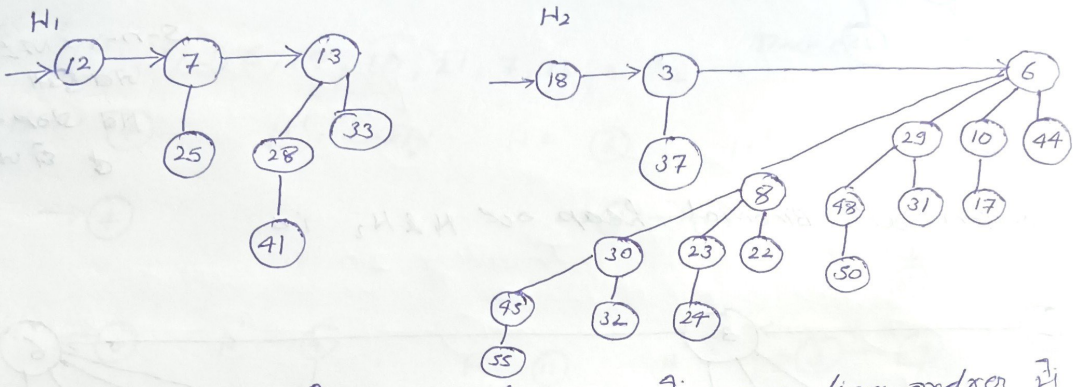




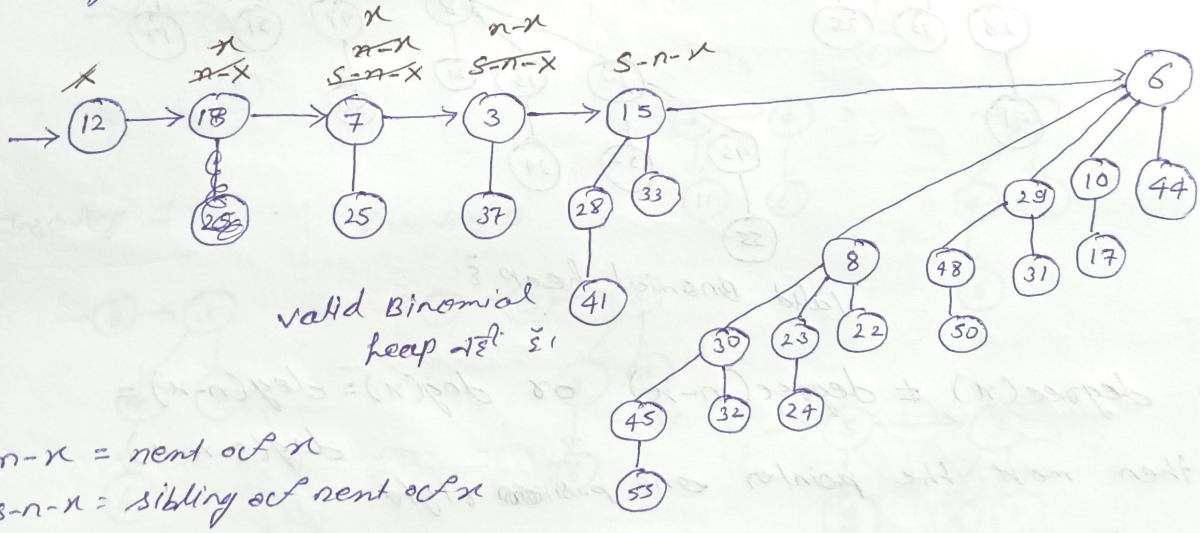


~~Heap में~~
Binomial heap में Binomial tree increasing order में होता है।

→ Binomial Heap union



union के लिए सबसे degree के term में ascending order में arrange कर देंगे

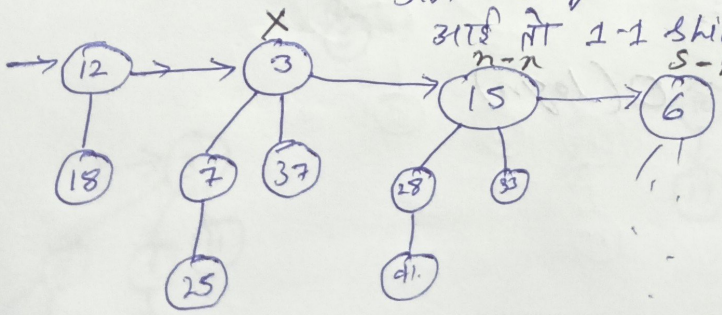


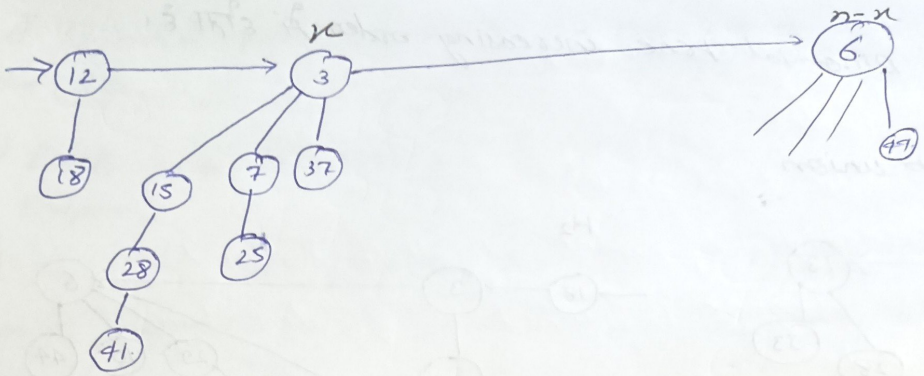
valid Binomial heap नहीं है।

n-x = next of x
s-n-x = sibling of next of x

x, n-x, s-n-x की degree निकालेंगे
अगर deg(x) = deg(n-x) ≠ deg(s-n-x) then x, n-x को combine कर देंगे, Min heap है तो दोनों में से जो min होगा उसको root node बना देंगे।

अगर deg(x) = deg(n-x) = deg(s-n-x) same आई तो 1-1 shift कर देंगे x, n-x, s-n-x

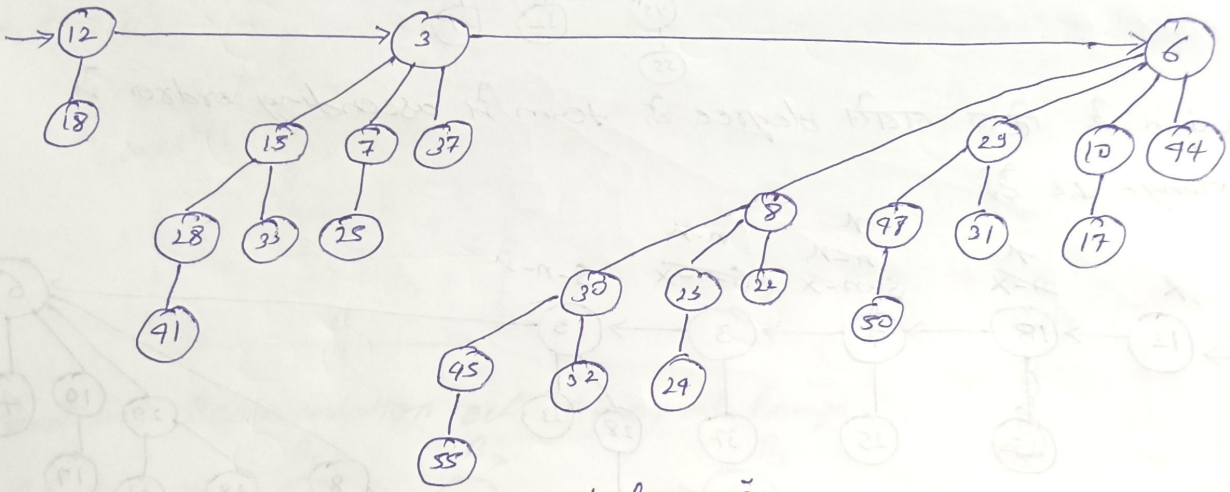




S-n-x
NIL

S-n-x = NIL
जब हुआ
तब Stop
के दो जाओ

So, union of Binomial heap of H_1 & H_2 is



Valid Binomial heap है।

* degree(x) ≠ degree(n-x) or deg(x) = deg(n-x) = deg(s-n-x)

then move the pointer one position right.

degree(x) = deg(n-x) ≠ deg(s-n-x)

तो merge कर देंगे।

H_1 = n_1 node
H_2 = n_2 node

then time complexity

$O(\log n_1 + \log n_2)$

$n_1 = n_2 = n$

then time complexity = $O(\log n)$

→ Inserting a node into Binomial min heap

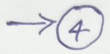
1.) Create a binomial Heap H containing new element - O(1)

2.) Apply union of two binomial min Heap H and H' - O(log n)

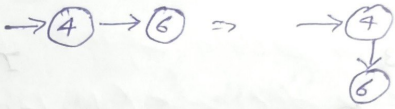
Time complexity: O(log n)

4, 6, 3, 11, 9, 5, 14, 10, 21, 7, 13, 20, 2

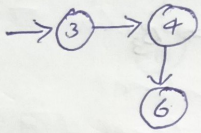
H = (4) H' = empty



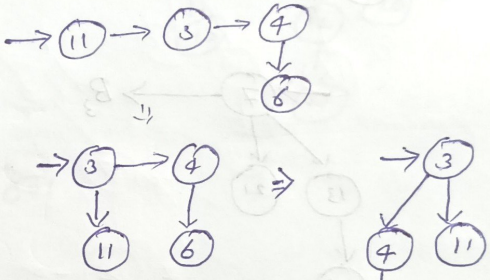
H = (6) H' = →(4)



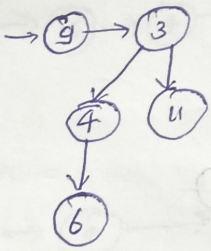
H = →(3) H' = →(4)→(6)



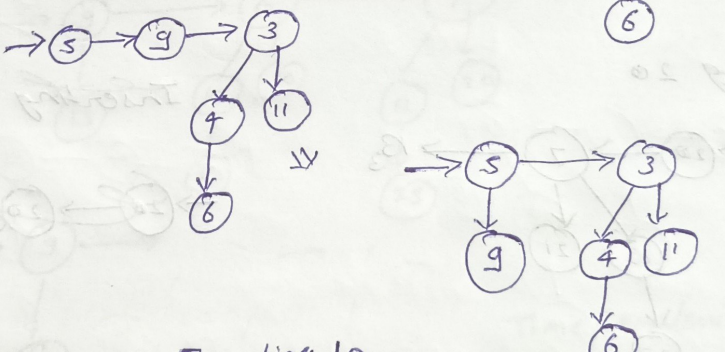
H = (11) H' = →(3)→(4)→(6)



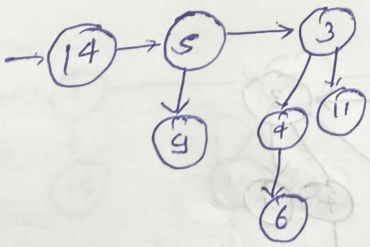
Inserting 9



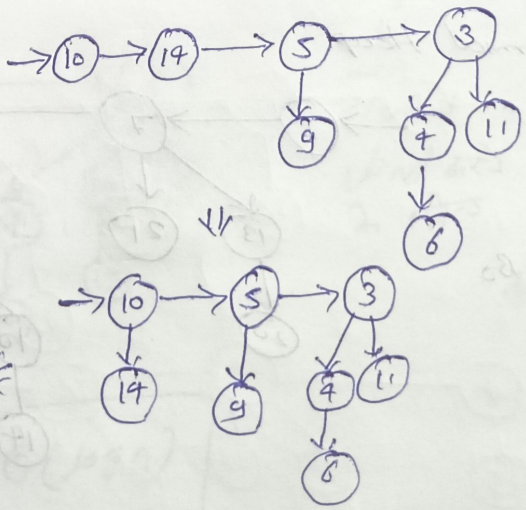
Inserting 5

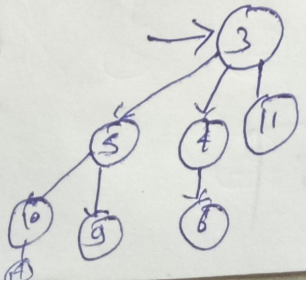


Inserting 14

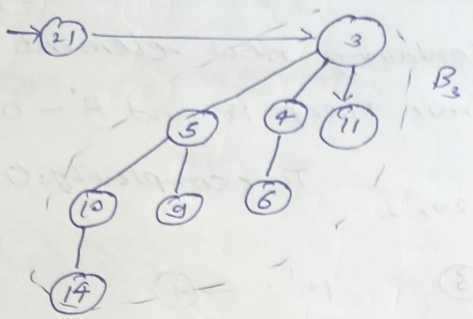


Inserting 10

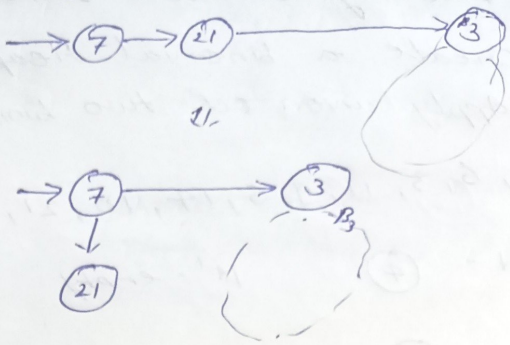




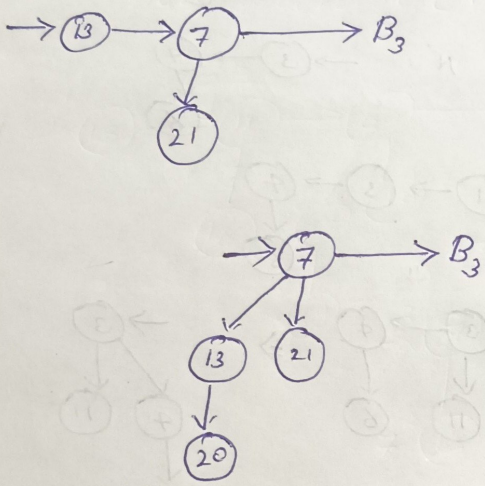
Inserting 21



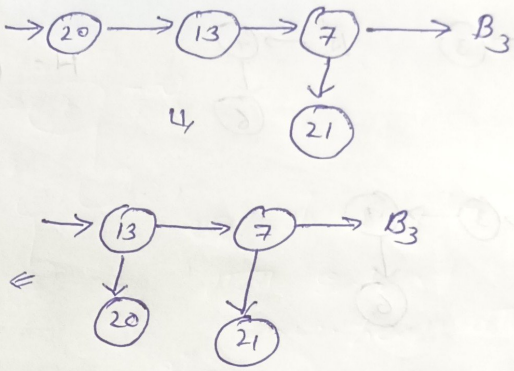
Inserting 7



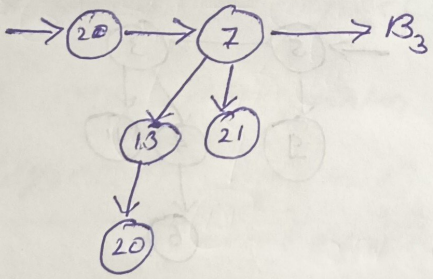
Inserting 13



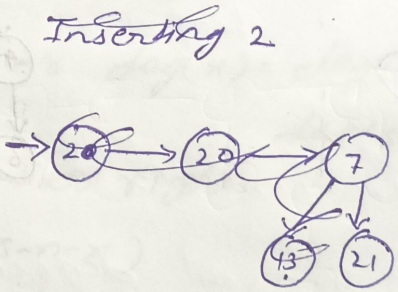
Inserting 20



Inserting 20



Inserting 2



So, Binomial Heap

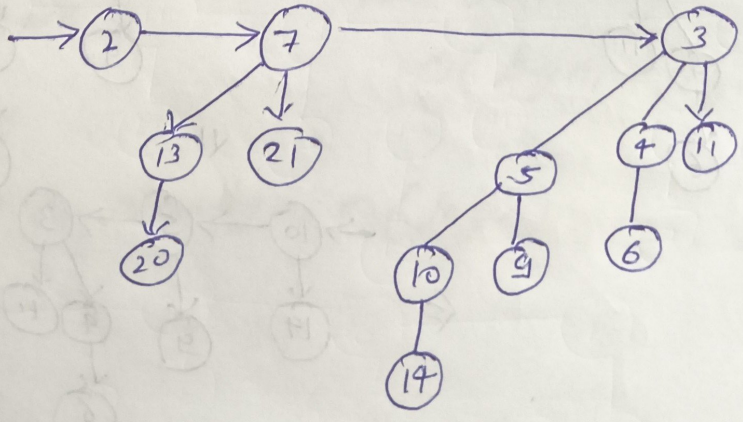
$n = 13$

$= 1101$

B_0

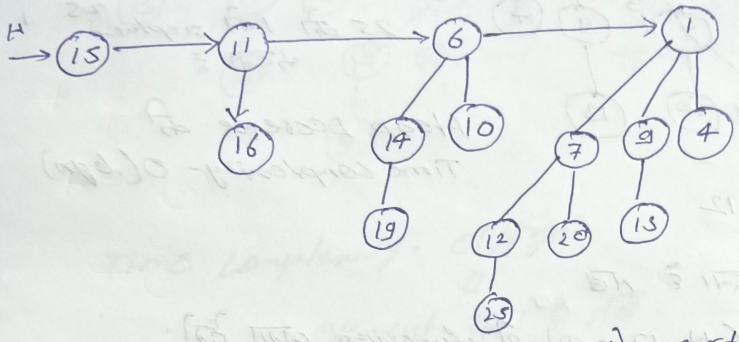
B_2

B_3



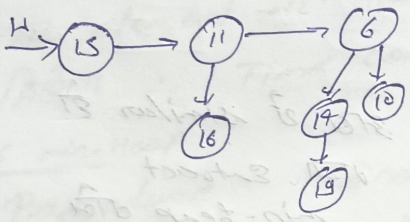
→ Extract Minimum from Binomial Heap

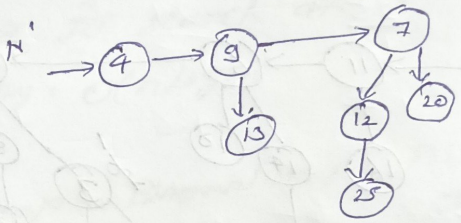
पहले minimum element search कर लेंगे → O(log n)



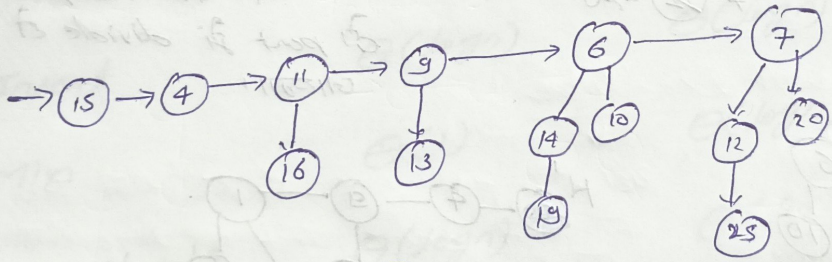
1 को delete कर देंगे तो Heap दो part में divide हो जाएगा।

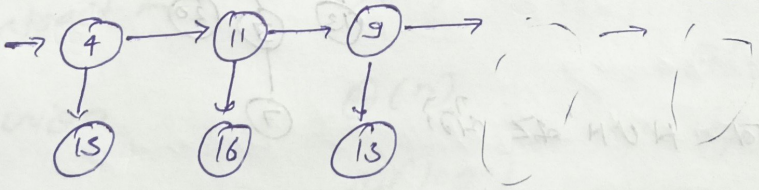
H & H'

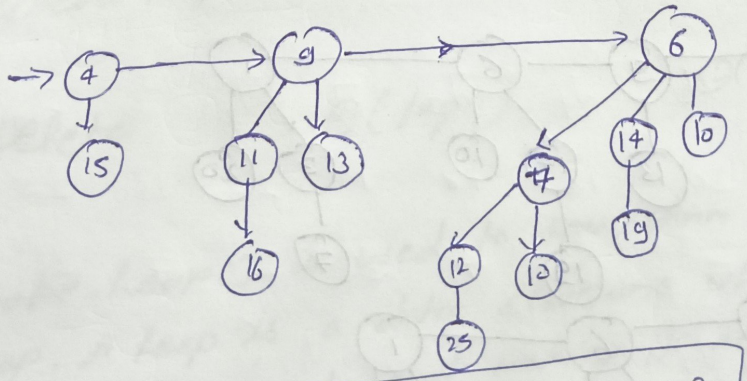




अब H & H' का union कर देंगे।







Time Complexity =

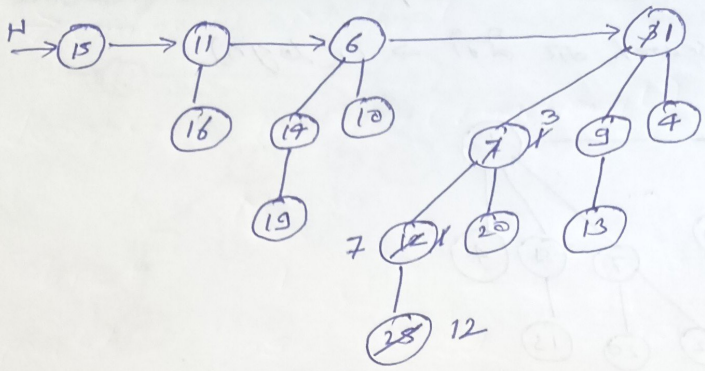
log n + log n + log n

↓ min element find करने के लिए

↓ union करेंगे

delete किए हुए को arrange करने का time

Time Complexity = O(log n)



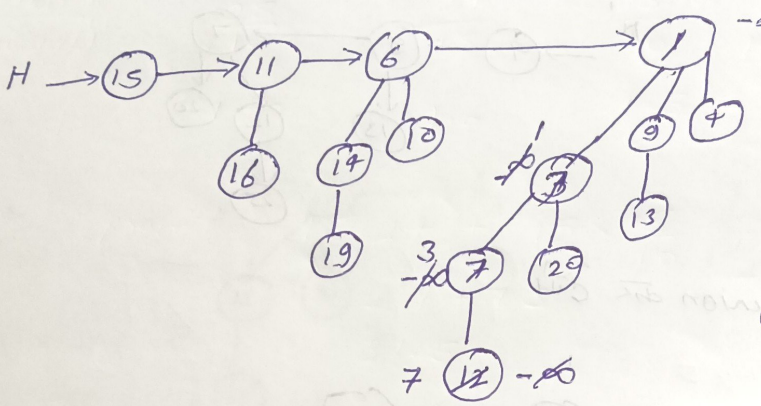
Heap Decrease (H, x, k)

25 को 1 से replace करना है

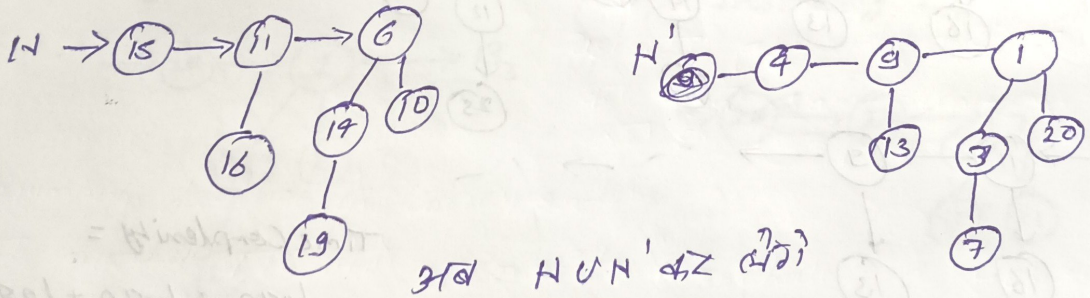
Heap Decrease की Time complexity - O(logn)

Suppose 12 को delete करना है तब

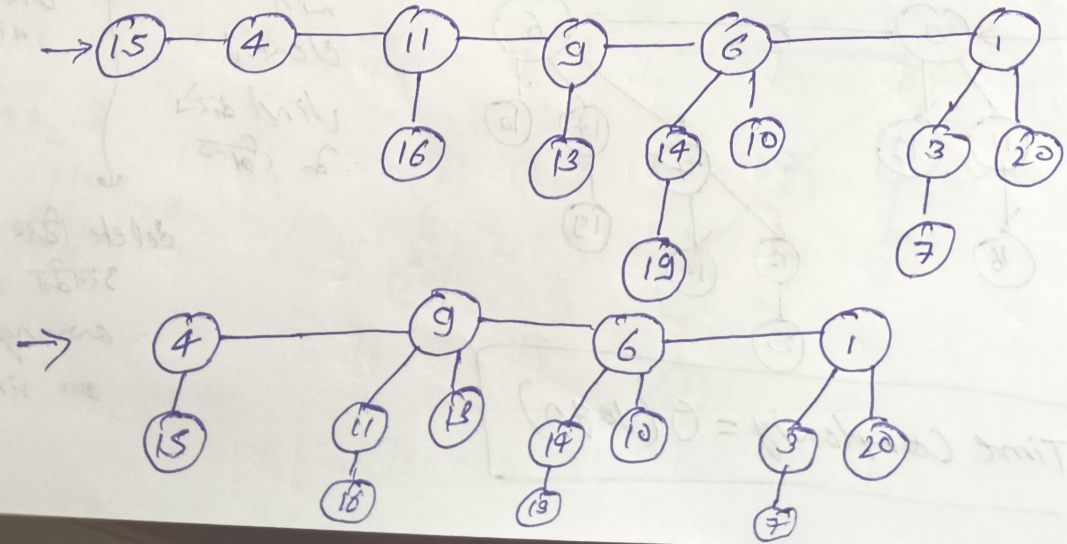
Heap decrease (H, 12, -∞) से function लगा देंगे.

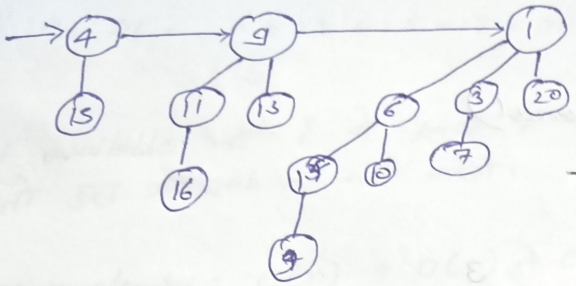


अब ये similar हो गया Extract min-heap वाले जैसा -∞ वाला remove कर देंगे Heap दो part में divide हो जाएगा।



अब HUN' कर लेंगे





Time :- $O(\log n) + O(\log n) +$

$O(\log n)$ ⇓ Heap decrease

$O(\log n)$ ⇓ Heap construct किया

$O(\log n)$ ⇓ union

Time complexity - $O(\log n)$ ⇓

Element पता है तब delete करने की ये time complexity है

अगर element नहीं पता है तब उसे search करने में $O(n)$ time लगेगा ∴ Time complexity : $O(n \log n)$

For min-Heap

Procedure	Binary Heap	Binomial Heap	Fibonacci Heap
Make Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
~~Make Heap~~	~~$\Theta(1)$~~	~~$\Theta(1)$~~	~~$\Theta(1)$~~
Insert	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Min	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
Extract min	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
union	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
Decrease key	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$

* make-heap() is used to transform a sequence into a heap. A heap is a data structure which points to highest or lowest element and making it access in $O(1)$ time.

→ Greedy Method

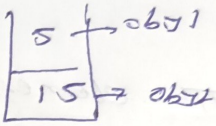
Knapsack Problem (Fractional)

M = 20

object	1	2	3
profit	25	24	15
weight	18	15	10

Profit / weight	$\frac{25}{18}$	$\frac{24}{15}$	$\frac{15}{10}$
	1.4	1.6	1.5

यहाँ पर Profit / weight को sort कर लेंगे।



∴, profit = 24 + 15 × $\frac{5}{10}$

= 24 + 7.5

= 31.5

Job Sequencing with Deadline

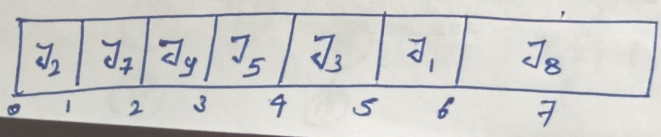
↓

यहाँ एक assumption है कि एक Job को करने same unit of time लगता है।

Maximum deadline के corresponding एक array बना लेंगे, Profit के behalf में descending order में arrange कर लेंगे

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

$J_3, J_9, J_7, J_2, \cancel{J_4}, J_5, J_8, J_1, \cancel{J_6}$



Profit = 20 + 23 + 25 + 18 + 30 + 15 + 16

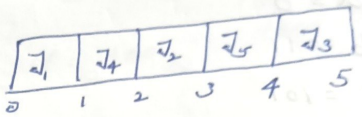
= 145

* slot available नहीं है तो left में देखेंगे, अगर left में भी जगह नहीं है तो उस Job को नहीं करेंगे।

Time complexity = $O(n) + O(1) + O(n \log n) + O(n^2)$

- $O(n)$ ⇓ maximum Deadline
- $O(n \log n)$ ⇓ sort करने के लिए according to profit
- $O(n^2)$ ⇓ Job को कहाँ place करें उसे खोजने में

Jobs	J_1	J_2	J_3	J_4	J_5
Profit	15	20	30	18	25
Deadline	5	5	5	5	5



No. of Comparision = $1+2+3+4+5 = \frac{n(n+1)}{2}$

J_3, J_5, J_2, J_4, J_1

∴ Time complexity $O(n^2)$

Huffman coding

→ used for data compression

variable length encoding

0 ← a

10 ← b

110 ← c

111 ← d

No. of bit = 50×1 + 40×2 + 6×3 + 4×3

= 50 + 80 + 18 + 12

= 160 bits

यहाँ पर कम bit used हो रहा है।

a c d a

0 110 111 0

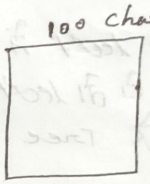
Fixed length encoding

00 ← a = 50

01 ← b = 40

10 ← c = 6

11 ← d = 4

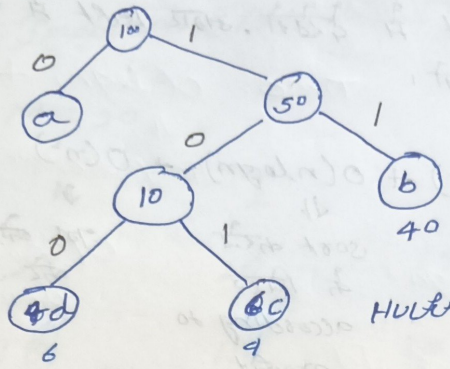


a,b,c,d को unique bit assign कर दिए इसको Fixed length encoding बोलते हैं।

Total no. of bit = 100×2 = 200 bit

a = 50
b = 40
c = 6
d = 4

50
40



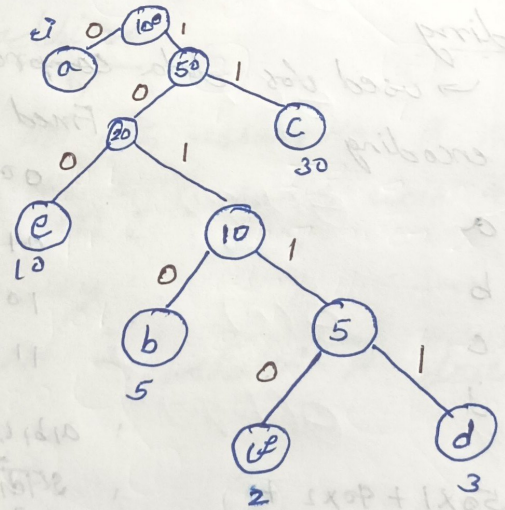
Huffman Tree

a = 0
b = 11
c = 101
d = 100

Q

a	b	c	d	e	f
50	5	30	3	10	2

2 Minimum पकड़ो, उन दोनो में जो minimum हो उसको left में place कर दो, उनके sum के correspond जो element है उसे replace कर दो 2 minimum लिये पको ये



Equal को left में रख रहे है तो left में ही रखना पूरे Tree construction में।

a = 0
b = 1010
c = 11
d = 1111
e = 100
f = 10110

Total no. of bit = 50×1 + 5×4 + 30×2 + 3×4 + 10×3 + 2×5
= 185 bit

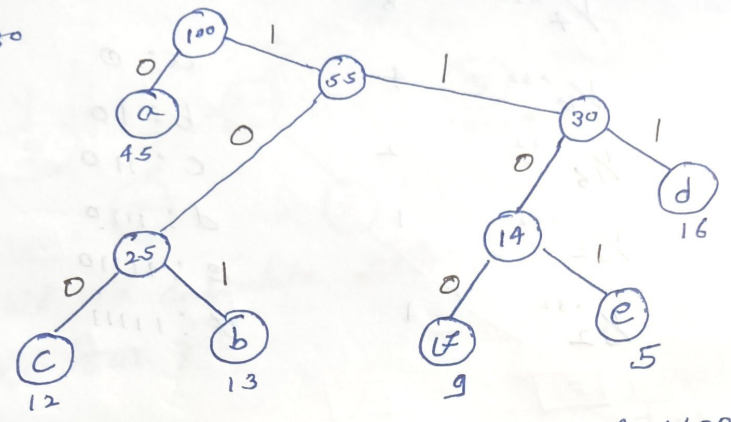
Average bit = $\frac{\text{total no. of bit}}{\text{No. of character}} = \frac{185}{100}$ = 1.85 bit/character

Q

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

25 14 30 55 100

45, ~~25~~ ~~30~~ 55



next 12, 13 के दोनों 14 से less है तो उनके लिए new tree बना लेंगे

a = 0, b = 101, c = 100, d = 111, e = 1101, f = 1100

Huffman (C) {

n = |C| = 6

Build min Heap Q with C → O(n)

for i = 1 to n-1

{ allocate a new node z

z.left = x = Extract_min(Q) → O(logn)

z.right = y = Extract_min(Q) → O(logn)

z.freq = x.freq + y.freq — O(1)

Insert(Q, z) — O(logn)

}

return (Extract_min(Q)) - O(1)

}

Time: $O(n) + (n-1)[O(\log n) + O(\log n) + O(1) + O(\log n)] + 1$

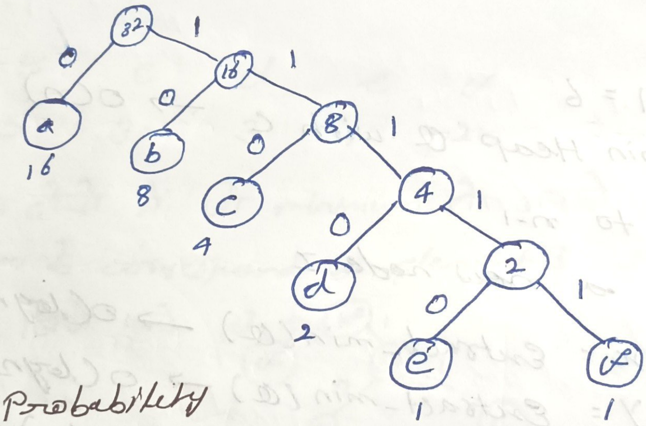
Time complexity = $O(n \log n)$

Q

Character	Probability		
a	$\frac{1}{2} \times 32$	16	a : 0
b	$\frac{1}{4} \times 32$	8	b : 10
c	$\frac{1}{8} \times 32$	4	c : 110
d	$\frac{1}{16} \times 32$	2	d : 1110
e	$\frac{1}{32} \times 32$	1	e : 11110
f	$\frac{1}{32} \times 32$	1	f : 11111

fraction में अच्छा नहीं लग रहा 32 से multiple कर रहे हैं

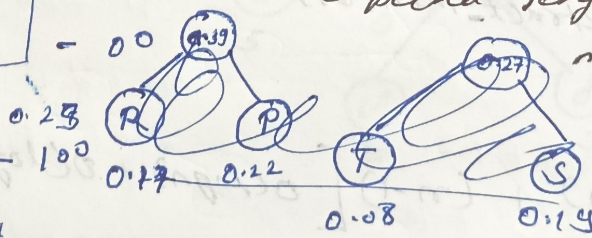
a	b	c	d	e	f
16	8	4	2	1	1



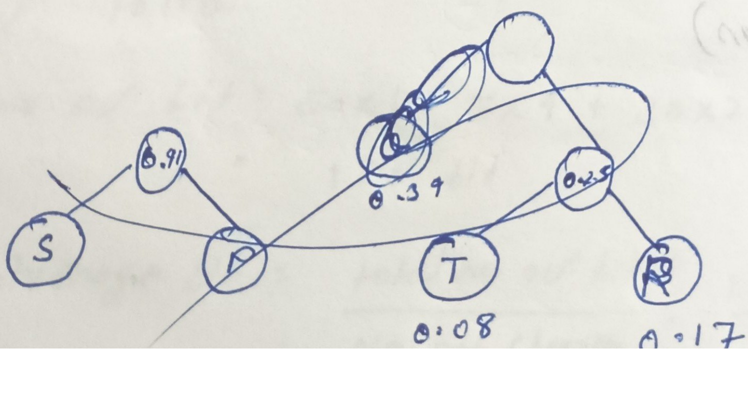
Q

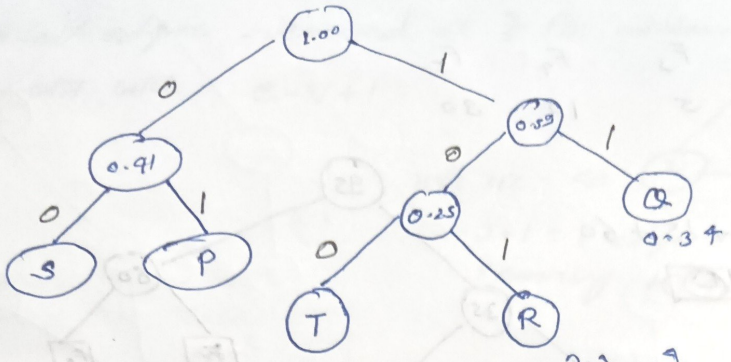
Character	Probability	
P	0.22	01
Q	0.34	11
R	0.17	101
S	0.19	00
T	0.08	100

If a message of 100 char over X is encoded using Huffman coding, then expected length of the message in bits ___



0.41, 0.25, 0.37





यहाँ पर total bit जब calculate करेंगे तो bit/character आएगा because probability दिया है

total bit/character = $0.22 \times 2 + 0.34 \times 2 + 0.17 \times 3 + 0.17 \times 2 + 0.08 \times 3$

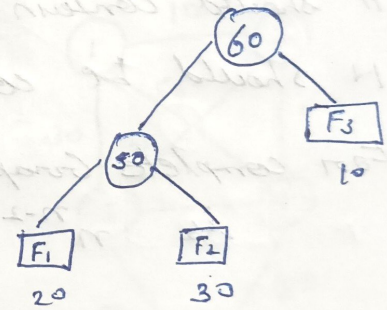
$= 2.25$ bit/Char

length of bit $= 2.25 \times 100 = 225$

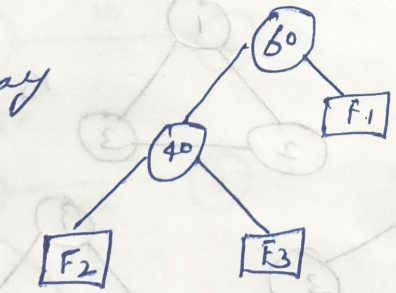
Optimal Merge Pattern

No. of record movement minimum करना है।

File	F_1	F_2	F_3
Records	20	30	10



इसका optimal way ~~एक~~ है F_3 जो minimum Record है उनको पहले move करायेंगे

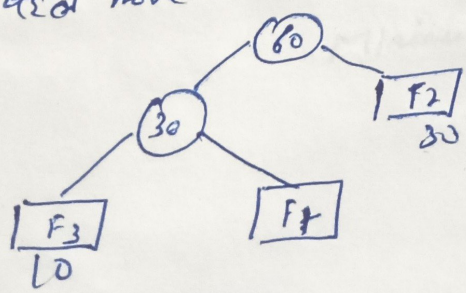


$60 + 40 = 100$

No. of Record movement $= 20 \times 2 + 30 \times 2 + 10 \times 1$

$= 40 + 60 + 10 = 110$

simply internal node का sum है $60 + 50 = 110$



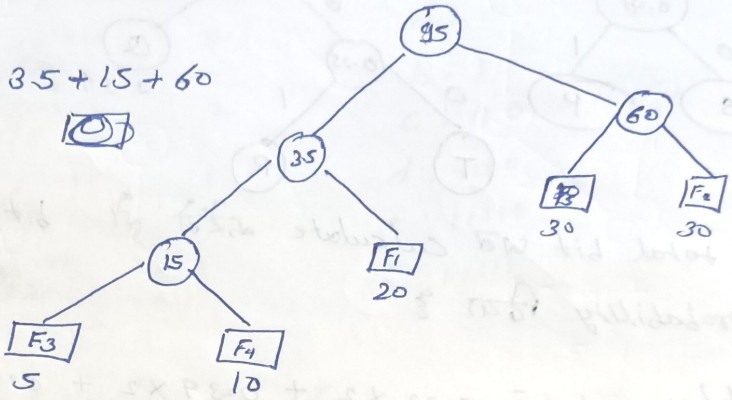
$60 + 30 = 90$

Huffman coding जैसा ही है।

File	F_1	F_2	F_3	F_4	F_5
Record	20	30	5	10	30

No. of file movement : $95+35+15+60$

$= 215$



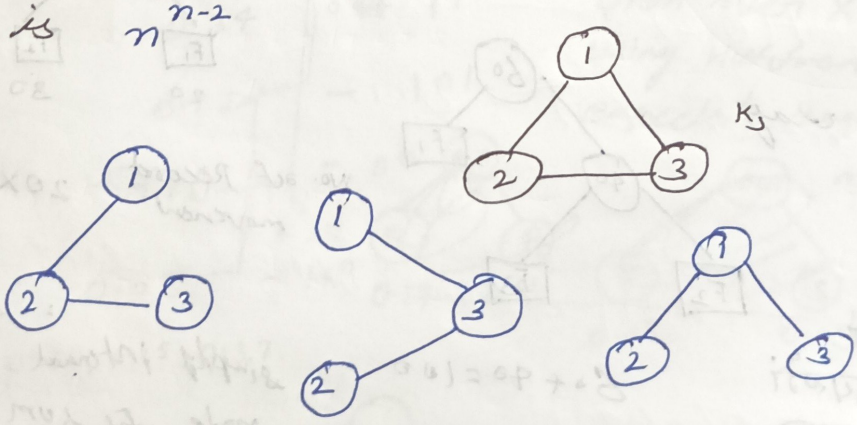
Time Complexity - $O(n \log n)$

→ Spanning Tree

A connected subgraph H of given Graph G is said to be spanning tree iff

1) H should contain all vertex of G
2) H should contain $(n-1)$ edge
3) H should be connected.

* For complete Graph (K_n) Number of spanning tree is n^{n-2}

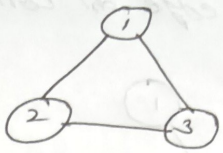


3 spanning tree

$K_3 = 3^{3-2} = 3^1 = 3$

* Given graph जिसमें V-vertex है उतने & e edges है

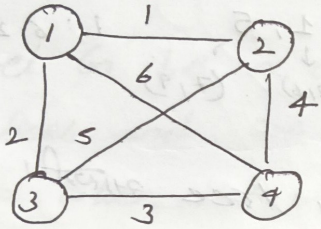
So. No. of edges removed कर दे तो minimum spanning tree बन जाए = e-V+1



यहाँ पर ⇐ 3-3+1 = 1 edge remove कर दे minimum spanning tree बन जाएगा

→ Minimum Cost Spanning Tree

↳ Minimum spanning tree जिसकी cost कम हो सबसे



⇒ इसका no. of spanning tree बनेगा : n^{n-2}

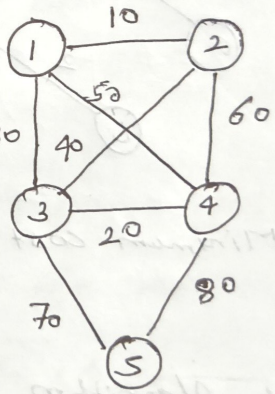
$= 4^{4-2}$

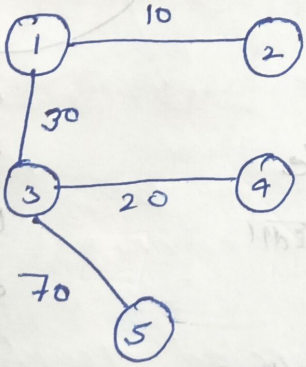
$= 16$

* अगर सारे edge distinct है तब Minimum spanning tree unique बनेगा।

→ Kruskal's Algorithm

सारे vertex लिख लो vertex को join जिस edge की cost min है वहाँ से such that cycle न बने

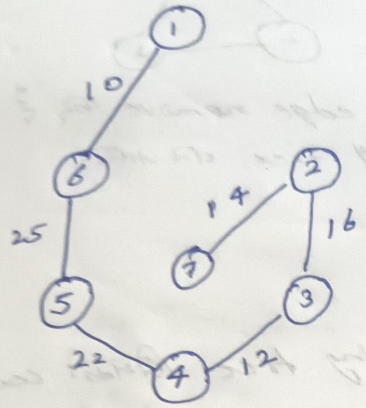


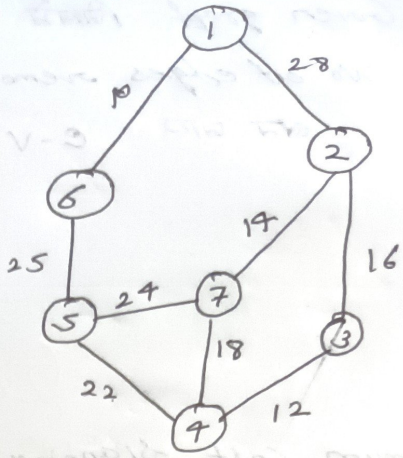


Minimum cost spanning tree

→ Prims Algorithm

हर step में लबका adjacent लिस्ट लेंगे और minimum edge को connect कर देंगे।





1 - ~~6~~, 2

~~6~~ - ~~5~~, 1

1, 5

↓

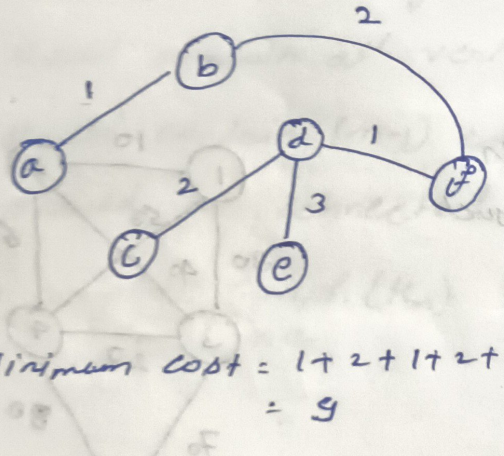
(2,6) (7,9)

4 : 5, 7, 3

1 : 6, 2

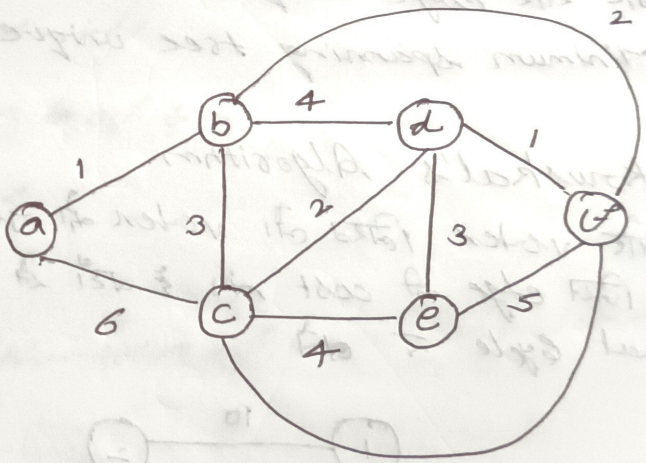
Prims Algorithm में graph हमेशा connected रहता है, Kruskal में disconnect हो सकता है।

दोनों से same minimum cost spanning tree आएगी।



Minimum cost = 1 + 2 + 1 + 2 + 3

= 9



a : c

b : c, d, f

f : e, c, d

d : c, e, b

Prims Algorithm में starting node given होगा। Kruskal में नहीं दिया रहता।

→ Prims Algorithm में Build_heap function use करते और each step में extract करते हैं minimum element heap से।

Time Complexity Prims Algorithm :- $O((V+E)\log V)$

or

$O(V^2 \log V)$

Time: $V + V + V\log V + E\log V$

↓ Minimum extract करने का।

↓ Decrease Heap function call करेंगे जब भी।

→ **Prims Algorithm में कभी भी A Graph disconnect नहीं होता इसपे question भी आता है।

Q Let G be a connected undirected Graph of 100 vertices and 300 edges. The weight of a MST of G is 500. when the weight of each edge of G is increased by 5, the weight of a MST become

Sol 100 - vertices
300 - edges

MST में (n-1) edge होगा
99 edge

$99 \times 5 = 495$ → इतना cost बढ़ जाएगा।

weight of MST become 995

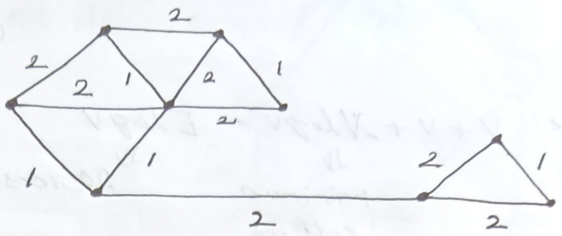
Q consider a graph $G = (V, E)$ where $V = \{V_1, V_2, \ldots V_{100}\}$, $E = \{(V_i, V_j)\} \mid 1 \le i < j \le 100\}$ and weight of the edge (V_i, V_j) is $|i-j|$ The weight of the MST of G is ——

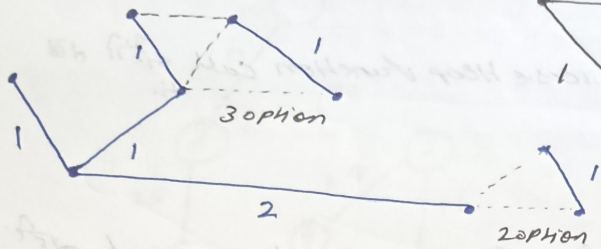
Sol $(2-1) + (3-2) + (4-3) + \ldots\ldots (100-99)$

$1 + 1 + 1 + \ldots\ldots\ldots + 1 = 99$

Q The no. of distinct min-spanning tree for the weighted graph below is

Sol.



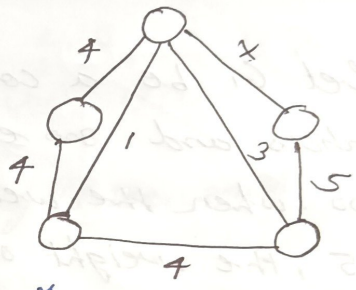


$3 \times 2 = 6$

Q consider the following undirected graph G:

Choose a value for X that will maximise the no. of min-weight spanning trees of G. The no. of MUST of G for this value of

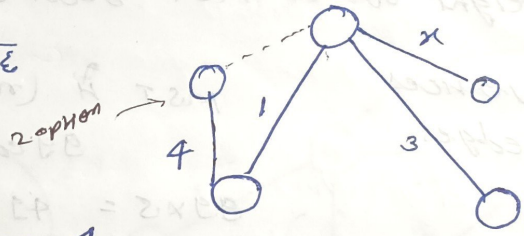
X ~~=~~ ≥ 4



यहाँ X की value जहाँ पूरे

रहा है no. of tree

पूरे रहा है

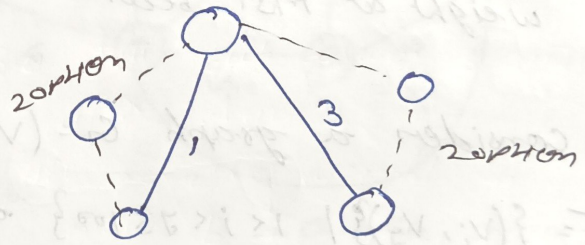


X = 4 ही possible value है

X = 5 के corresponding 4 graph आएंगे

X = 4 के corresponding 2 graph आएंगे

इस value के लिए ही maximum आएंगे।



X > 5 के लिए minimum no. of spanning tree = 2

4 option

Q Let G be an undirected connected graph with distinct weight Let e_{max} be the edge with max-weight and e_{min} be the edge with min-weight.

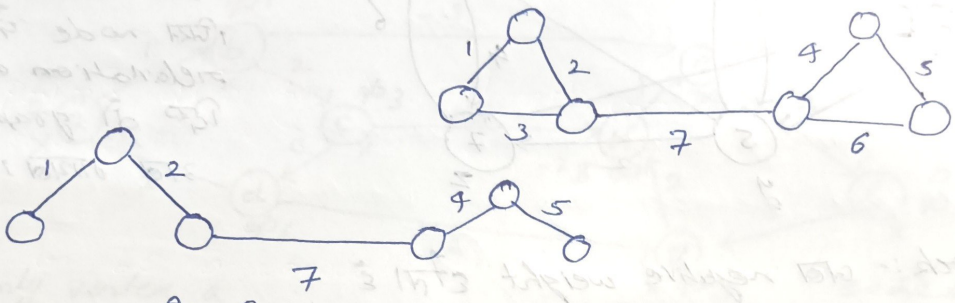
False

A) Every MST of G must contain e_{min} True

B) if e_{max} is in a MST, then its removal must disconnect G True

C) No MST contain e_{max} False

d) G has a unique MST True



Question में बोल देता repeated edge है या सिर्फ edge बोलता तब option d सही होता अतएव False

→ Dijkstra Algorithm

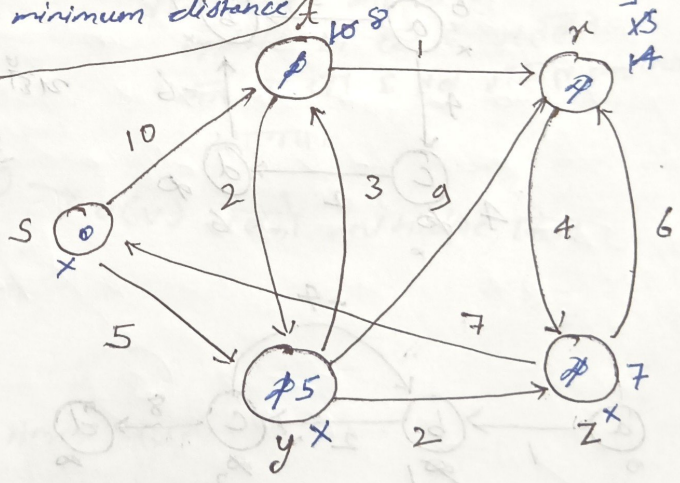
दोनों graph में applicable होता है directed & undirected graph

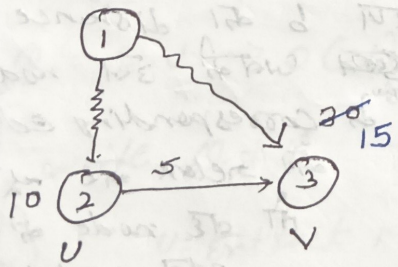
Relaxation

if $d(v) > d(u) + w(u,v)$

then $d(v) = d(u) + w(u,v)$

हर node पर जो distance लिखा रहता वो source से उस node तक का minimum distance होता है।





$30 > 10+5$

$20 > 15$

Step1: source को zero बाकी को ∞ assign कर देंगे

Step2: outgoing edges को relax करेंगे।

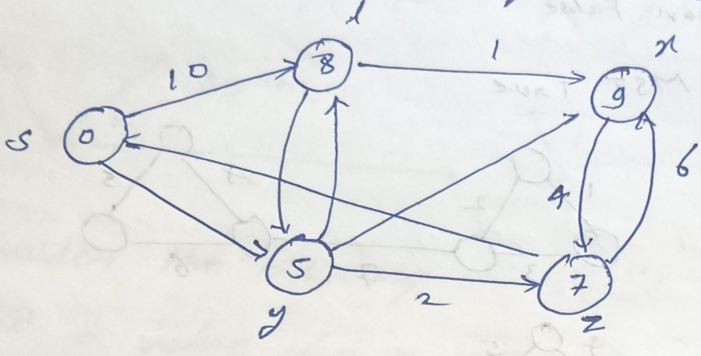
Step3: उस node से associated edge को relax करेंगे

Step4: Repeat step 2 जब तक सारे node relax न हो जाए

Step5: उस Min distance वाला node को select करेंगे

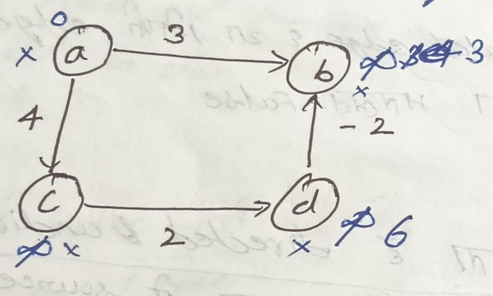
जो node relax हो गया एक बार दूसरी बार उसकी तरफ देखेंगे भी नहीं। ऐसा इसलिए करते है because उस node से अब कोई edge relax नहीं होगी। और distance भी change नहीं होता।

After applying Dijkstra Algorithm Graph become

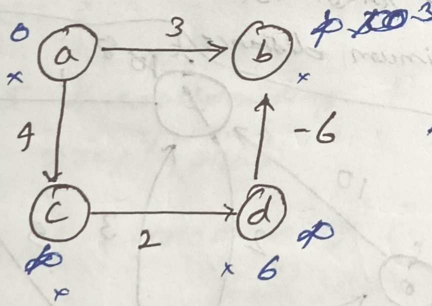


* सारे weight +ve है तो मान लो जिस node के relaxation लगा दिए वो graph से अब गायब।

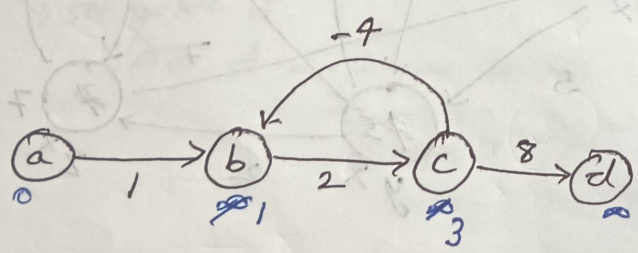
Drawback:- जब negative weight होता है



इस case में Dijkstra Algorithm सही answer दे रहा है।



यहाँ answer गलत दे रहा है
3 देगा but answer 0 आना चाहिए था
यहाँ negative weight के कारण b का distance update होना चाहिए जबकि उस node के corresponding edge को relax कर चुके है तो वह node की distance अब update नहीं होगी।



↓

negative weight cycle है

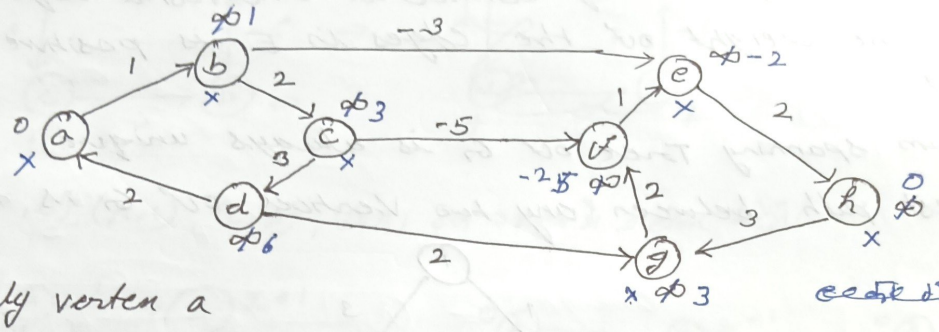
↓

मतलब $-4+2 = -2$ b का distance.

ऐसे करते करते b, c का distance $-\infty, -\infty$ आता but dijkstra b का distance 1 देगा c का distance 3 देगा।

→ Graph में -ve weight है तब answer सही भी आ सकता था गलत भी।
but अगर Graph में -ve weight Cycle present है तब answer गलत ही आएगा।

Q Compute the correct shortest path distance to



a) only vertex a
b) only vertices a, e, v, g, h
c) only vertices a, b, c, d
d) All the vertices

सबका answer सही आ रहा है.

** एक बार जिस node पे X लगा दिए further Dijkstra algorithm लगाते जाएं further node पर अगर X वाले node का distance update हो रहा है संतुलन बन आ रहा है तब गलत answer आएगा।

→ Dijkstra's Algorithm

Dijkstra (G, w, s)
Initialise single source (G, s) → O(V) सबको initialise किया
$S = \emptyset$ — O(1)
$Q = G.V$ → O(V)
(MinHeap)
while $Q \neq \emptyset$ — V times चलेगा।
u = Extract Min (Q) — O(log V)
$S = S \cup \{u\}$
for each vertex $v \in G.Adj[u]$ — O(V)
Relax (u, v, w) → O(log V)] O(V log V)
→ Decrease Key function

$O(V^2 \log V)$

Relaxation Function उतनी बार ही call होगा जितनी edges हैं।

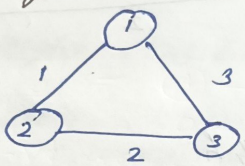
Time: $O(V) + O(1) + O(V) + O(V \log V) + O(E \log V)$

Time Complexity : $O((V+E)\log V)$ or $O(E \log V)$

Q Let $G(V, E)$ be any connected undirected edge-weight graph. The weight of the edges in E is positive & distinct.

1) ✓ Minimum spanning Tree of G is always unique

2) ✗ Shortest path between any two vertices of G is always unique



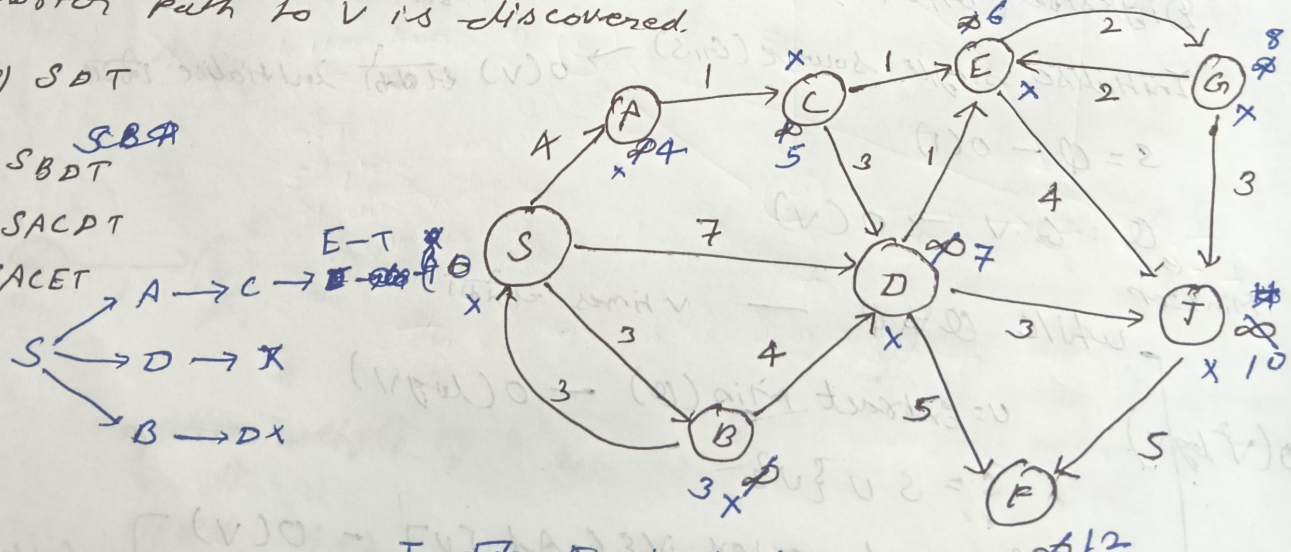
Q Consider the directed graph shown in the figure below. There are multiple shortest path between vertices S and T which one will be reported by Dijkstra's shortest path algo². Assume that, in any iteration, the shortest path to a vertex V is updated only when a strictly shorter path to V is discovered.

a) SDT

b) SBDT

c) SACDT

d) ✓ SACET



S → A → C → E → G → T

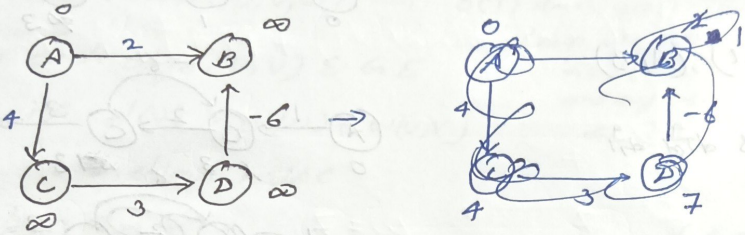
S → D → T

S → B → D → T

T तक का shortest path

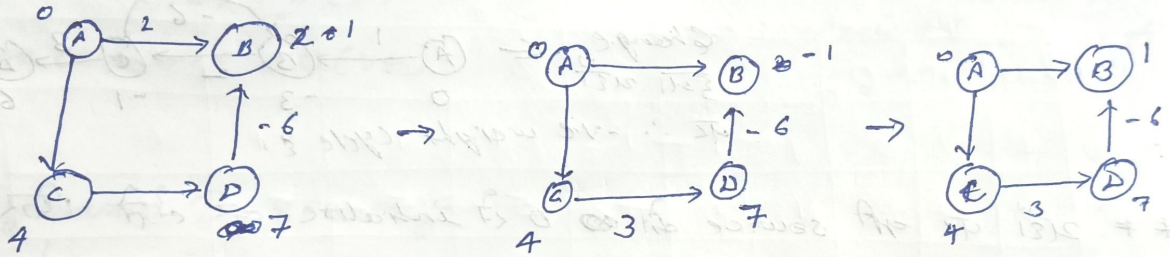
→ Bellman Ford Algorithm

Suppose V vertex है तो सारे edges (V-1) time relax करेंगे। जिस sequence में पहली बार relax किए थे उसी sequence में हर बार relax करेंगे।



किसी भी order में First time relax कर सकते हैं।

(A,B), (A,C), (C,D), (D,B) इस order में relax करेंगे।



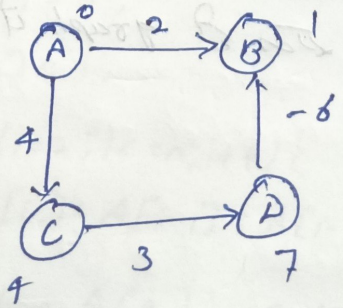
-ve weight cycle में Bellman Ford Algorithm भी fail जाएगा। गलत answer देगा।

Bellman Ford Algorithm -ve weight cycle detect कर लेता है।

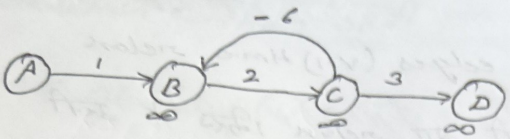
(V-1) time relax करने के बाद अगर 1 time और relax किए और edge relax हो रही तब -ve weight cycle है।

above case 1 बार और relax कर रहे हैं

weight	Dijkstra	Bellman
+ve	✓	✓
-ve	✓/✗	✓
-ve cycle	✗	✗



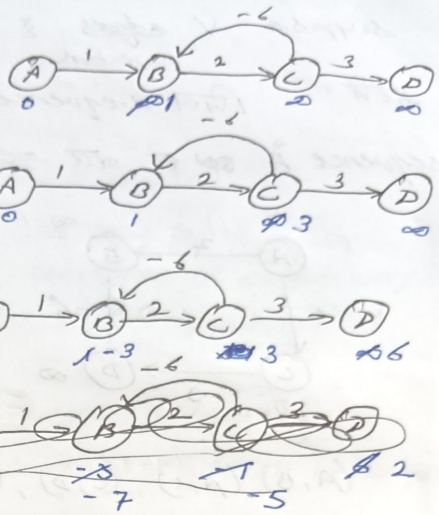
no updation मतलब -ve weight cycle नहीं है।



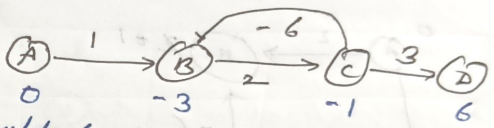
V = 4-1 = 3

(C, D), (C, B), (B, C), (A, B)

हर step पर previous वाले को input मान लेंगे।



Change हुआ यहाँ पर ∴ -ve weight cycle है।

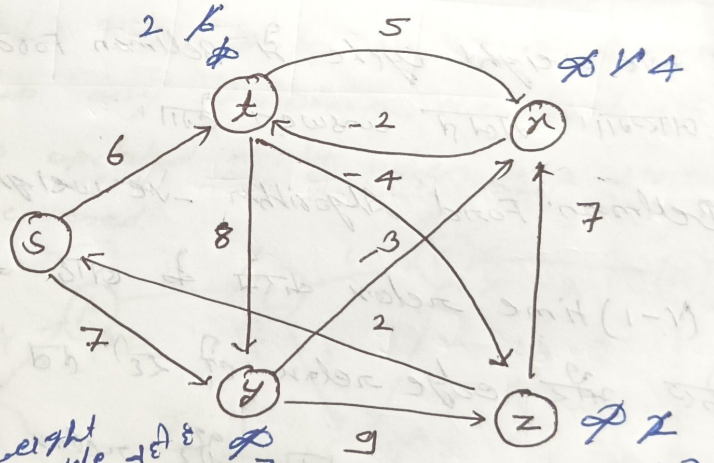


** यहाँ पर भी source को ∞ 0 से initialise कर लेते फिर बाकी node को ∞ से।

(t, x)
(t, y)
(t, z)
(x, t)
(y, x)
(y, z)
(z, x)
(z, s)
(s, t)
(s, y)

4 iteration

-ve weight cycle के लिए 1 iteration और check करेंगे। 5th iteration में कोई value update नहीं हुई इसका मतलब -ve weight cycle नहीं है



एक ही graph में कैसे भी कर सकते है।

Bellman Ford (G, w, s) → weight matrix

Initialise single source (G, s) → O(V)

for i = 1 to |G.V| - 1 → (V-1) times

  for each edge (u, v) ∈ G.E → E times

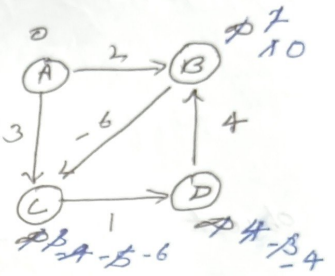
    Relax (u, v, w) → O(1) times यहाँ keep use नहीं कर रहे array use कर रहे हैं

for each edge (u, v) ∈ G.E

  if d(v) > d(u) + w(u, v)

    return False

return True.



(A, B), (A, C), (C, D), (D, B) (B, C)

यहाँ -ve weight cycle present है, D का weight -5 ही आ रहा है।

Time: O(VE) + O(E)

Time complexity : O(VE)

→ Dynamic Programming

→ Memorization (Top-down Dynamic programming)

जो भी function call करेंगे पहले check करेंगे की वो function पहले तो call नहीं किया है अगर किया है तो function call नहीं करेंगे जो value store है वही use कर लेंगे।

```
F(n) {
  if (n ≤ 1)
    return n;
  if [A[n] != -1)
    return A[n];
  else
    return A[n] = F(n-1)
           + F(n-2)
```

→ Bottom Up Dynamic Programming (Tabulation)

```
F(n) {
  A[0] = 0;
  A[1] = 1;
  for (i = 2; i <= n; i++) {
    A[i] = A[i-1] + A[i-2]
  }
  return A[n];
}
```

Bottom से start किया है

दोनों approach से

Time complexity - O(n)

Space complexity - O(n)

Fibonacci series के लिए।

→ 0/1 knapsack problem.

$$KS(n,w) = \begin{cases} 0 & n=0 \text{ or } w=0 \\ KS(n-1,w) & wt[n] > w \\ \max\{KS(n-1, w-wt[n]) + P[n],\ KS(n-1,w)\} \end{cases}$$

object	1	2	3	4
Profit	1	4	5	7
weight	1	3	4	5

$w = 7$

$(n+1)\ (w+1)$

	wt	P	w=0	w=1	w=2	w=3	w=4	w=5	w=6	w=7
n=0	0	0	0	0	0	0	0	0	0	0
n=1	1	1	0	1	1	1	1	1	1	1
n=2	3	4	0	1	1	4	5	5	5	5
n=3	4	5	0	1	1	4	5	6	6	9
n=4	5	7	0	1	1	4	45	7	8	9

$KS(1,1)$ — $KS(0,0)+1$ / $KS(0,1)$

weight ~~का~~ को ~~का~~ को consider किया है।

1	2	3	4
0	1	1	0

9 के above वाले row में 4 मिलता उसे include नहीं किया।

$9 - 5 = 4$

$4 - 4 = 0$

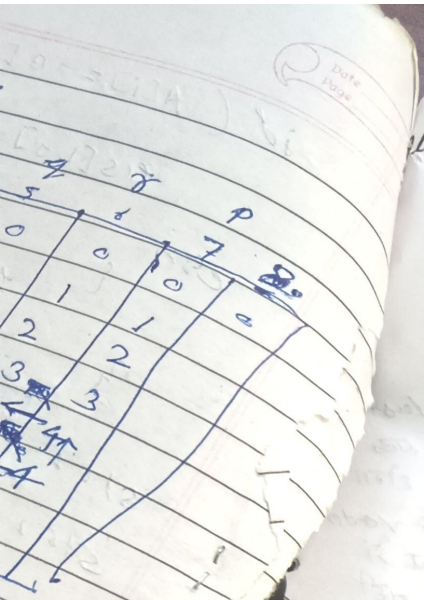
यहाँ 4 मिला उसके above row में 4 नहीं है तो उसे include नहीं करेंगे।

0 above row में present ∴ ~~0~~ को भी include नहीं किया

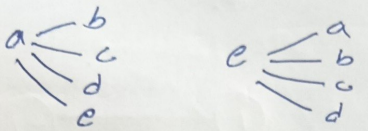
Time complexity :

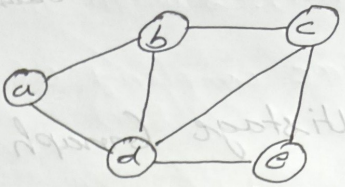
$O((n+1)(w+1))$

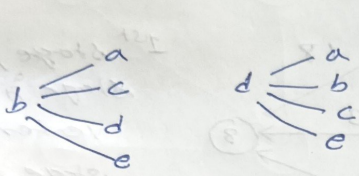
$O(nw)$

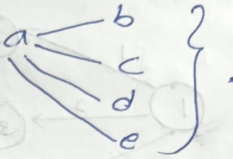


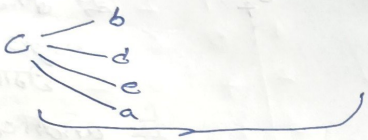
All pair shortest Path (Floyd warshall)











Floyd warshall में हर vertex से दूसरे vertex का shortest path मिल जाता है।

* अगर हम Dijkstra Algorithm लगाते हैं All pair shortest path निकालने के लिए तो हर vertex से Dijkstra Algorithm लगाएंगे। तब $V \cdot O(E \log V)$ $E = \frac{n(n-1)}{2}$

$$V \cdot O(V^2 \log V) = O(V^3 \log V)$$

Bellman लगाएंगे तब - $V \cdot O(VE)$

$$\therefore V \cdot O(V \cdot V^2) = O(V^4)$$

But Floyd warshall Algorithm की Time complexity - $O(V^3)$ होती है।

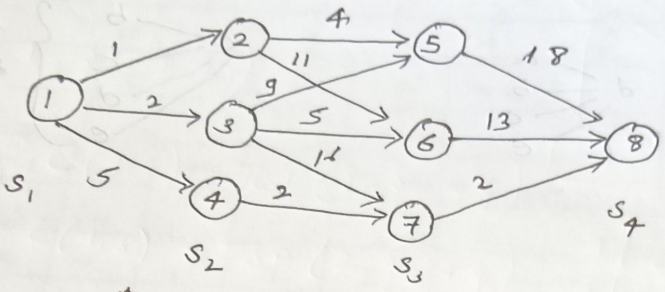
→ Directed, undirected graph दोनों के लिए applicable है।

* -ve weight, -ve weight cycle दोनों के लिए Floyd warshall algorithm गलत answer देगा।

Dijkstra, Bellmann, Floydwarshall से bellmann ही -ve weight cycle detect कर पाता है, -ve weight सिर्फ हो तो सही answer देता है।

For procedure see notes मत्रे से लिखता हूँ अपने पढ़ लेना bsdk करना Gaand mraoo!!

→ Multistage Graph



1st stage, last stage में एक vertex होगा। stage के vertex की बीच में edge नहीं होगा। undirection होगी सारी edge एक ही direction में।

stage ↑ vertex ↑

$$MS(1,1) = \begin{cases} c(1,2) + MS(2,2) \\ c(1,3) + MS(2,3) \\ c(1,4) + M(2,4) \end{cases}$$

$$MS(s_i, V_j) = \begin{cases} \min\{ c(V_j, K) + MS(s_{i+1}, K) \} & \forall K \in s_{i+1}, \\ & \text{and } (V_j, K) \in E \\ c(V_j, D) & \text{if } (s_i == s_F - 1) \end{cases}$$

min cost required from stage s_i and vertex V_j to destination.

Dynamic programming से solve करेंगे :- Approach :- Bottom up DP

1	2	3	4	5	6	7	8
9	22	18	4	18	13	2	0

$A[4] = c(4,7) + A[7]$

$\therefore 2 + 2 = 4$

$$A[2] = \begin{cases} c(2,5) + A[5] \\ c(2,6) + A[6] \end{cases} \quad \begin{matrix} 4 & 18 \\ 11 & 13 \end{matrix}$$

$A[1] = 1+22,\ 2+18,\ 5+4$

$$A[3] = \min \begin{cases} c(3,5) + A[5] \Rightarrow 18 \\ c(3,6) + A[6] \Rightarrow 13 \\ c(3,7) + A[7] \Rightarrow 2 \end{cases} \quad \begin{matrix} 9 \\ 5 \\ 16 \end{matrix}$$

vertex 1 से vertex 8 का min distance = 7

→ Longest common subsequence

X: A B C D H F

Y: A E D F K

ADF
Longest common subsequence

$$LCS(m,n) = \begin{cases} 0 & \text{if } m=0 \text{ or } n=0 \\ 1 + LCS(m-1,n-1) & \text{if } X[m] == Y[n] \\ \max \begin{cases} LCS(m,n-1) \\ LCS(m-1,n) \end{cases} & \text{if } X[m] \neq Y[n] \end{cases}$$

इस problem को brute force से करेंगे तो $O(2^{m+n})$

X: A B C D

Y: A E D F

		A	B	C	D
	0	0	0	0	0
A	0	1	1	1	1
E	0	1	1	1	1
D	0	1	1	1	2
F	0	1	1	1	2
		A			D

LCS : AD

Time complexity: $O(m \times n)$

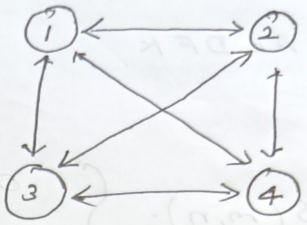
→ Travelling Salesman Problem (using Dynamic Programming)
starting vertex से सारे vertex को cover without repeating any vertex और ऐसे cover करना है कि cost minimum आए।

Hamiltonian cycle find करना है जिसकी cost minimum है।

n = 5 vertex है तो ~~उसके~~ according brute force के according
(n-1)! solution आएंगे।

1 2 3 4 5 1
4!

so, Time complexity O(n!) i.e.
O(n²)



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

$$TSP(1, \{2,3,4\}) = \min \begin{cases} C(1,2) + TSP(2, \{3,4\}) \\ C(1,3) + TSP(3, \{2,4\}) \\ C(1,4) + TSP(4, \{2,3\}) \end{cases}$$

~~Dynamic~~ से करेंगे तो $(n-1)2^{n-2}$ बार function call होगा

$$TSP(i,S) = \begin{cases} C(i,k) + TSP(k, S-\{k\}) & k \in S \quad S \neq \phi \\ C(i,1) & S = \phi \end{cases}$$

$T(1, \{2,3,4\})$

15

$(1,2) + T(2, \{3,4\})$ 6 9

$(1,3) + T(3, \{2,4\})$

$(1,4) + T(4, \{2,3\})$

13 9

$(2,3) + T(3, \{4\})$ 2 11

$(2,4) + T(4, \{3\})$ 1 8

$(3,4) + T(4, \{\phi\})$ 8 3

$(4,3) + T(3, \{\phi\})$ 7 1

$(4,1) = 3$

$(3,1) = 1$

ऐसे ही आगे सब कर लेंगे।

min(13, 9)

→ Sum of subset Problem

{3, 2, 1, 7} sum = 6

Brute force मतलब सारे के corresponding 2^n subset वाले।
तो Time complexity $O(2^n)$

$$SS(n,s) = \begin{cases} SS(n-1,s) & \text{if } s < V[n] \\ SS(n-1, s-V[n]) \text{ OR } SS(n-1,s) & \text{if } s \geq V[n] \end{cases}$$

{3, 2, 1, 7} sum = 6

False $n=0, s \neq 0$

True $n=0, s=0$

True $n \neq 0, s=0$

	0	1	2	3	4	5	6
n=0	T	F	F	F	F	F	F
n=1 (3)	T	F	F	T	F	F	F
n=2 (2)	T	F	T	T	F	T	F
n=3 (1)	T	T	T	T	T	T	T
n=4 (7)	T	T	T	T	T	T	T

SS(1,1) $s < V[n]$ $1 < 3$
↓
SS(0,1)

True मिला

SS(3,1,3) = SS(0,0) OR S(0,3)
T OR F

$(n+1)(s+1)$

Time complexity $O(ns)$
space complexity $O(ns)$

→ Matrix Chain Multiplication

$$M[i,j] = \min_{i \leq k < j} \{ M[i,k] + M[k+1, j] + P_{i-1} P_k P_j \}$$

0 if $i = j$

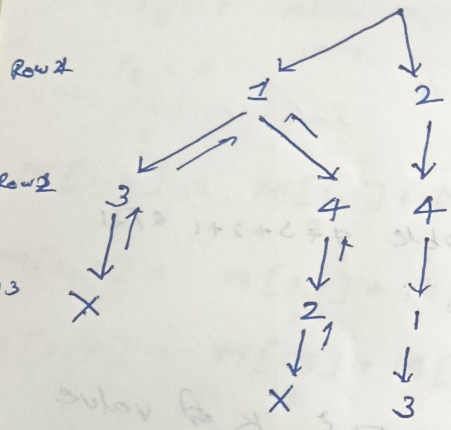
→ Backtracking (DFS follow करती है)

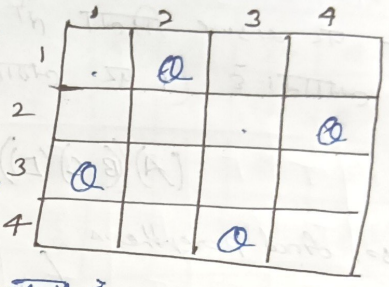
↳ जब problem को divide करके solve करता है तभी type time check भी करता है

for eg n number को sort करना है तो Greedy वाला देगा Brute force n! way निकाल लेगा और last में check करेगा कौन सा correct है but Backtracking हर step पर check करता है इससे answer आएगा हल नहीं, नहीं आएगा तो backtrack करके दूसरी possibility check करेंगे।

→ N queens problem (using backtracking)

Row के रहा है





Backtracking से multiple answer आ सकता है

2, 4, 1, 3 ऐसा इसका mirror image आएगा।

→ इस तरह के tree को state space Tree बोलते है।

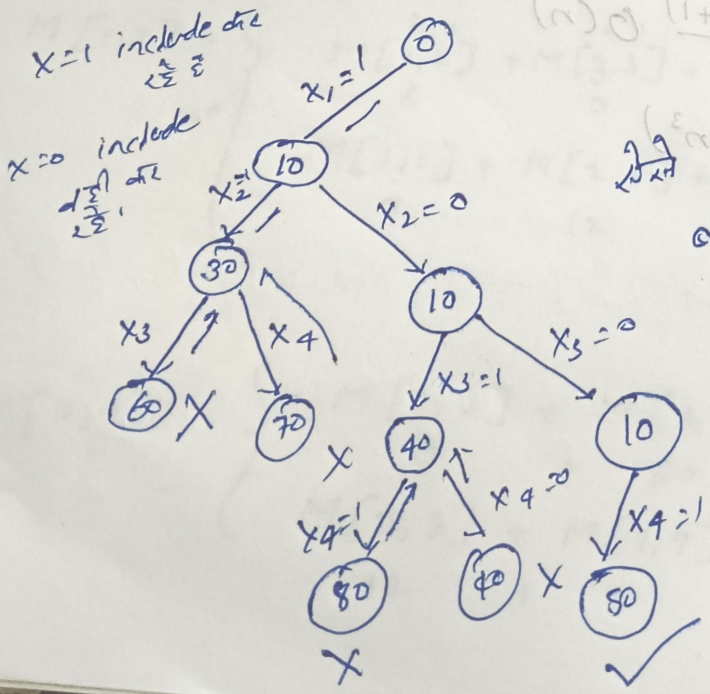
→ Sum of subset problem → Brute force $O(2^n)$

(Backtracking)

$\{10, 20, 30, 40\}$

$x_1 \quad x_2 \quad x_3 \quad x_4$

Sum = 50



ऐसे ही बाकी solution आएगें।

$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$

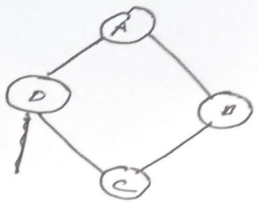
1	2	3	4
1	0	0	1

→ Graph Coloring Problem

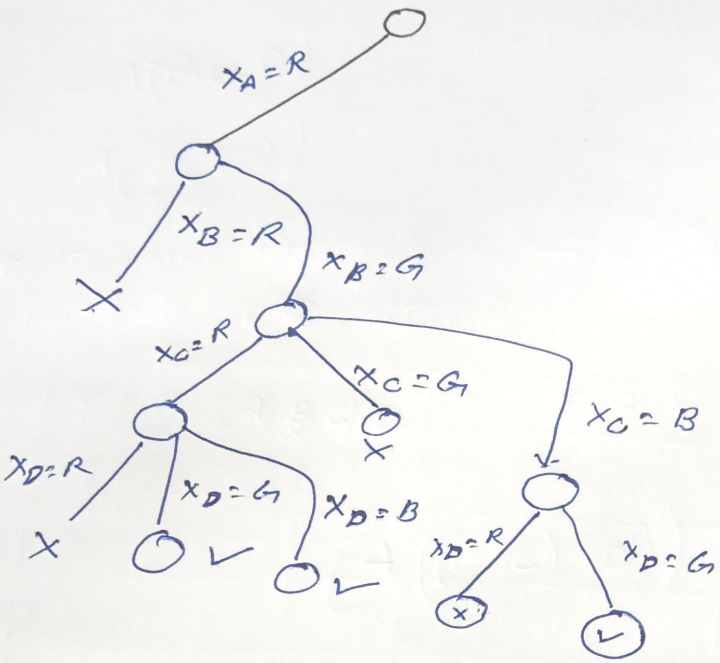
Brute force से $3 \times 3 \times 3 \times 3 = 3^4$

n vertices होंगे तब 3^n

Backtracking से करते हैं 3 रंग



colors {R, G, B}



R G R G

R G R B

R G B G

ऐसे ही बाकी कर लेंगे multiple answers आएगा।