time Complexity কমানোর জন্য algorithm যেভাবে process/resource কম লাগবে সেটা Algo. faster হবে।

Algo = Processor Operation = Sum of all individual operation

Assigment operation = 1 individual operation

Arithmatic " = +, −, *, / = 1 "

Comparism = >, <, >= = 1 "

return " = 1 "

```
→ int function 1 (int x)
{
  x=x+1;      ——— 1+1
  return x*5  ——→ 1+1
}                  4
```

```
int sum=0 —— 1
for(int i=1; i<=n; i++)   1+1
{
  sum=sum+i;  → 1+1+1
}
                Comparison ±1
return sum;  +1
```

* Total time: 8 processor operation

Comparing Algorithms Using processor Operation:

```
int linearsearch(int A[], int n, i) {
  for(i=0; i<n; i++)
    if(A[i]==value)
      return i;
  return -1;
}
```

1 operation
0(মনে 10 = 11 operation
10 times = 10 "
10 times = 10 "
1 times
মোট = 1+11+10+10+1
= 33 operations

input(n)	no. of operation
10 =	33
100 =	103
⋮	

Time Complexity -এর ধারণা :

input → $T(n)$ → output

input-size — processor operation

depends করে নির্ভর

- int sum(int a, int b)

```
{
  x = a*b;    // 2
  return x;   // 1
}
```

$T(n) = 3$ alway = 3 (Constant)

Constant amount of time : input এর উপর নির্ভরশীল না।

```
int sum = 0;
for (i=1; i<=n; i++)
    sum = sum + i;
return sum;
```

$1 + 1 + (n+1) + n + 2n + 1$

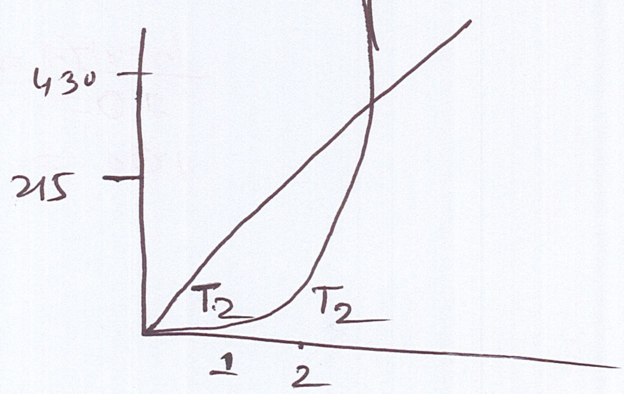
$T(n) = 4n + 4$

Best Case, Worost Case Time Complexity :

যদি অ্যালগরিদমের time complexity বাড়ানো যায় তখন বাড়াতে হবে Worost case time complexity.

Comparison Using time Complexity :

$T_1(n) = n^2$	$T_2(n) = 215\,n$
$n = 1 \quad = 1$	215
$n = 2 \quad = 4$	215×2
⋮	
$n = 215 \times 215$	$n = 215 \times 215$
$n = 216 \times 216$	$n = 215 \times 216$
	⋮



একটা বড় value এর জন্য কোন Complexity ভালো বোঝা যায়

- যদি n এর মান বড় দেয়া হয় তাহলে কি হবে

$T_1(n) = 67n^2 + 58n + 35$

$n = 10^7$,

$67 \times (10^7)^2 + 58 \times 10^7 + 35$

এখানে

Consider না করলেও হবে,

তাই $T_1(n) = O(n^2)$

(only Consider larger term)

$T_2(n) = 55n^2 + 10\sqrt{n} + 3$

No need

ক্রমবর্ধমান হার: $1 < \log n < \sqrt{n} < n < n\log n < n^2 < 2^n < n!$

$7n^2 + 2n + 37$
$n^2 + n$
$75n^2$
} all are same in perspective of $O(n^2)$

Same ক্রম $6n + 39$, $6n + \log n + 39$, $85n$ — all are same $O(n)$

Big-O Arithmetic

```
int n = 5          O(1)
int sum = 0        O(1)
for(int i=1; i<=n; i++)
{
    sum = sum + i;
}                  O(n)
for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
    {
        //---;
    }              O(n^2)
```

$O(n^2) + O(n) + O(1) + O(1)$

$= O(n^2)$

$O(f(n)) + O(g(n))$

$= O(f(n))$ if $f(n) > g(n)$

$= O(g(n))$ if $g(n) > f(n)$

$O(f(n)) * O(g(n))$

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

```
for (int i=1; i<=n; i++)          O(n)
{
    for (j=1; j<=n; j=j*2) O(log n)
    {
    }
}
```

= O(n) × O(log n)

= O(n log n)

Types of function:

O(1) → Constant

O(log n) → Logarithmic

O(n) → Linear

$O(n^2)$ ⇒ Quadratic

$O(n^3)$ ⇒ Cubic

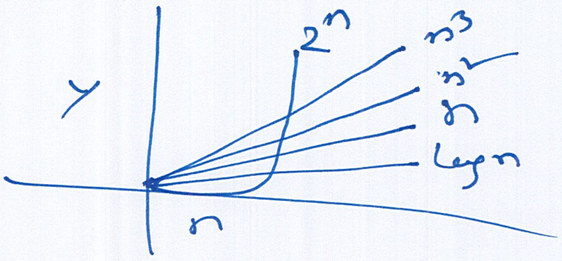
$O(2^n)$ ⇒ Exponential

$O(3^n)$ ⇒

f(n) = 2
f(n) = 5 → O(1)
f(n) = 5000

f(n) = 2n+3
f(n) = 5000n + 700 → O(n)
$f(n) = \frac{n}{3000} + 6$

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



O → upper bound

Ω — lower "

Θ – Theta (average bound)

Best, Worst, Average Case Analysis:

Linear Search:

7	9	10	12	13

Best case: যাকে খুঁজি তাকে যদি প্রথমেই পেয়ে যাই তাহলে হবে Best case. (present at 1st index)

↳ Best case time Constant = $O(1)$

Worst case: When the algorithm takes maximum amount of time. যখন যে value খুঁজতেছি সেটা সবার শেষে পাই বা নাই। → $O(n)$

Average case: $\dfrac{\text{all possible case time}}{\text{number of cases}}$

সব সময় সম্ভব হয় না, তাই worst case কেই average case ধরা হয়।

all possible case = যদি প্রথম index এ পাই তাহলে 1 Comparison, যদি 2য় index এ পাই তাহলে 2 comparison, যদি 3য় index এ পাই তাহলে 3 Comparison

$$\text{Average} = \frac{1+2+3+\cdots+n}{n \text{ cases}}$$

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

[Case গুলো শুধু Analysis] [Notation (O, Ω, Θ) হলো মাপকাঠি]

→ Best case, worse এর জন্য কোন notation ব্যবহার করা হয়?

Recurrence Relation:

Example1:

```
Void Test (int n)  ——— T(n)
{
  if(n>0){ ———————— 1
    Printf("%d", n); — 1
    Test (n-1);  ——— T(n-1)
  }
}
```

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

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

Solving Using Substitute Method,

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

$$T(n) = [T(n-2)+1] + 1$$

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

$$T(n-2) = T(n-3) + 1$$

$$T(n) = T(n-3) + 3$$

$$\vdots$$

Continue for K times;

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

Assume $n-K=0$;

$$n = K$$

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

$$T(n) = T(0) + n$$

$$= 1 + n$$

$$T(n) \geq O(n)$$

Example 2:

```
void Test(int n){
  if(n>0){ ———————— 1
  for(i=0; i<n; i++){ ———— n+1
    printf("%d", n); ———— n
  Test(n-1); ———————— T(n-1)
  }
}
```

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

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

$= T(n-1) + n$

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

Solvⁿ: $T(n) = T(n-1) + n$ — (I)

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

So, $T(n) = T(n-2) + (n-1) + n$ — (II)

$T(n-2) = T(n-3) + n - 2$

From (II) $T(n) = [T(n-3) + n-2] + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$ — (III)

$\vdots$

K^{th} time,

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

Assume, $n - K = 0$

$n = K$

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

$T(n) = T(0) + 1 + 2 + \cdots + (n-1) + n$

$= T(0) + \frac{n(n+1)}{2}$

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

$= O(n^2)$

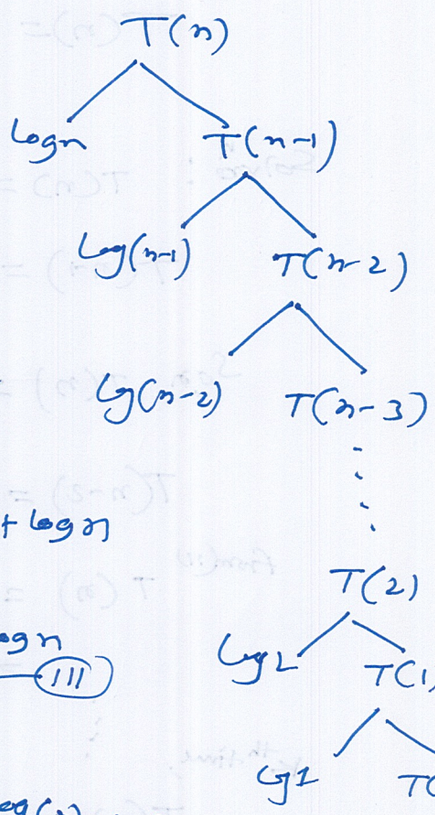
Ex 3

```
Void Test(int n){
  if (n>0)
  {
  for (i=1; i<n; i=i*2)
    {printf("%d", i);        — log n
    }
  Test(n-1);                 — T(n-1)
  }
}
```

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

$$T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-1)+\log n & n>0 \end{cases}$$

Tree method



Log(n) + Lg(n-1) + ---- + Log2 + Log1

Substitution:

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

$T(n) = [T(n-2) + \log(n-1)] + \log n$

$= T(n-2) + \log(n-1) + \log n$ — (II)

$T(n) = [T(n-3) + \log(n-2)] + \log(n-1) + \log n$

$= T(n-3) + \log(n-2) + \log(n-1) + \log n$ — (III)

⋮

k times,

$T(n) = T(n-k) + \log 1 + \log 2 + \cdots + \log(n-1) + \log n$ $= \log(n*(n-1)*\cdots 2\times 1)$

$= \log n!$

$\log(n \log n)$

$n-k=0$

$n=k$

$T(n) = T(0) + \log n!$

$= 1 + \log n!$

$O(n \log n)$

Binary Search:

```
Algorithm BinarySearch(l, h, key)
{ if (l == k) {
    if (A[l] == key)
        return l;
    else return 0;
  }
  else, mid = (l+h)/2;
  if (key == A[mid])
        return mid;
  if (key < A[mid]):
  { return BinarySearch(l, mid-1, key);
  } else, return BinarySearch(mid+1, h, key);
}
```

$T(n/2)$

worst case, worst case

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

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

Substitution method,

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$$

Put $T(n/2)$ is $T(n)$

$$\therefore T(n) = T\left(\frac{n}{4}\right) + 1 + 1$$

$$= T\left(\frac{n}{4}\right) + 2 \longrightarrow (11)$$

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$$

Put $T(n/4)$ in (11)

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

$$= T\left(\frac{n}{8}\right) + 3 \longrightarrow (111)$$

⋮

K term,

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

Let, $\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

$\therefore k = \log n$

$$\Rightarrow T(n) = T(1) + \log n$$

$$= 1 + \log n$$

$$= O(\log n)$$

Max-Min problem: (Divide & Conquer)

Algo:

```
Max : a[i]
Min : a[i]
for i=2 to n do
    if a[i] > max then — n-1
        max = a[i]
    if a[i] < min then — n-1
        min = a[i];
return (max, min);
```

2n-2

Method 2: Divide & Conquer করে দুই part এ ভাগ করে Max আর Min বের করলে, তাহলে Max, Max এবং Min, Min Comparison করতে or 1 টার বেশি লাগবে:

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

$$= 2T\left(\frac{n}{2}\right) + 2 \quad \text{——(I)}$$

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

from (I) $T(n) = 2\left\{2T\left(\frac{n}{4}\right) + 2\right\} + 2$

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

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2^2 + 2 \quad \text{——(II)}$$

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

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

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

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2 \quad \text{——(III)}$$

$$\vdots$$

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

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

$$n = 2 \cdot 2^k \Rightarrow n = 2^{k+1}$$

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

$$= 2^k \cdot 1 + 2^k + 2^{k-1} + \cdots\cdots + 2$$

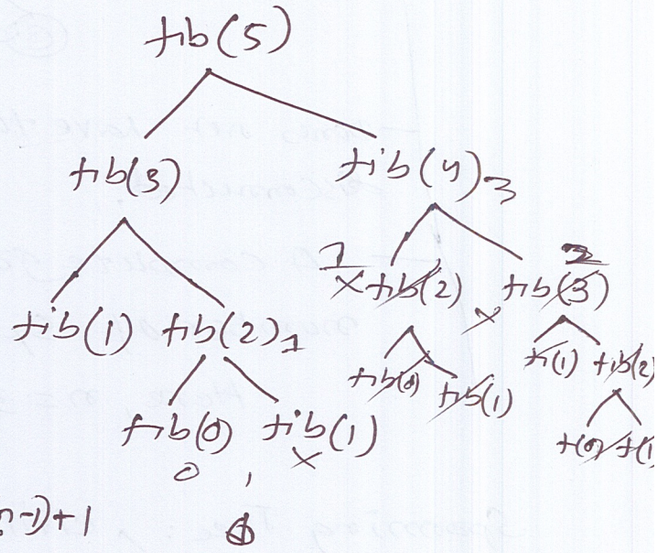
$$= 2^k + \frac{2(2^k - 1)}{2-1} \Rightarrow 2^k + 2^{k+1} - 2 \Rightarrow \frac{n}{2} + n - 2 = \frac{3n}{2} - 2$$

So, 8 element, method 1 = 2n-2 = 14

" 2 = $\frac{3n}{2} - 2 = 10$ ——

Dyramic programming:

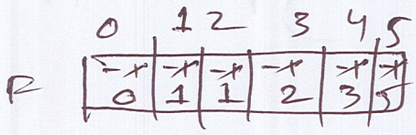
```
int fib(int n)
{
  if (n<=1)
    return n;
  else
    return fib(n-2)+fib(n-1);
}
```



$T(n-2)$ $T(n-1)$

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

$O(2^n)$



$fib(n) = n+1$ calles

$= O(n)$

Memoization.

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

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

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

$= 4T(n-2)+2+1$

$T(n-2) = 2T(n-3)+1$

$T(n) = 4\{2T(n-3)+1\}+2+1$

$= 8T(n-3)+4+2+1$

$\vdots$

$T(n) = 2^K T(0) + \frac{K(K+1)}{2}$

Or, $T(n) = 2T(n-1)+1$

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

$T(n) = 4T(n-2)+3$

$T(n) = 8T(n-3)+7$

$\therefore T(n) = 2^K T(0) + (2^K - 1)$

$= 2^K + 2^K - 1$

$= 2^n + 2^n - 1$

$= 2^n(1+1-1) = 2^n$

$= O(2^n)$

$n-K = 0$

$n = K$

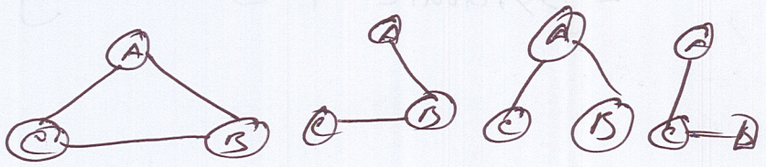
$\Rightarrow T(n) = 2^K + \frac{K^2+K}{2}$

$= 2^n + \frac{n^2+n}{2} = \frac{2 \cdot 2^n}{2}$

$= O(2^n)$ $= 2^n + \frac{n^2}{2} + \frac{n}{2}$

Greedy Algorithm

Spanning Tree:



- Does not have the cycle and it cannot be disconnected.
- A complete graph has maximum n^{n-2} number of spanning trees.

  Here, $n=3$ So $3^{3-2}=3$

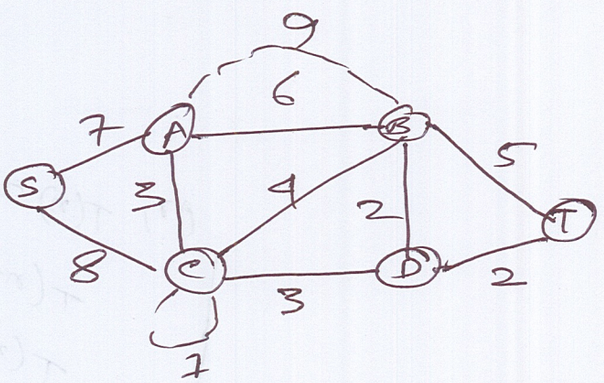
Spanning Tree: Civil network planning
Computer Network Routing protocol.
Cluster Analysis

Minimum Spanning Tree (MST): In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

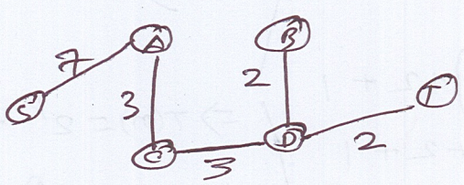
- Kruskal's Algo
- Prim's

} greedy Algorithm

Kruskal's:



B,D	D,T	A,C	C,D	C,B	B,T	A,B	S,A	S,C
2	2	3	3	4	5	6	7	8



Cost = SA + AC + CD + DB + BT

= 7 + 3 + 3 + 2 + 2 = 17

Huffman coding:

A B B C D B C C D A A B B E E E B E A B
= 20 character

A = 65 — 01000001 = 8 bits

How many bits will be used = 20 × 8 = 160 bits

Fixed-length coding

another method of fixed-length coding:

Character	Count / frequency	Code
A	4	000
B	7	001
C	3	010
D	2	011
E	4	100

Total Character = 20 = 20 × 3 bits = 60 bits

Table character = 5 × 8 bit = 40 bits

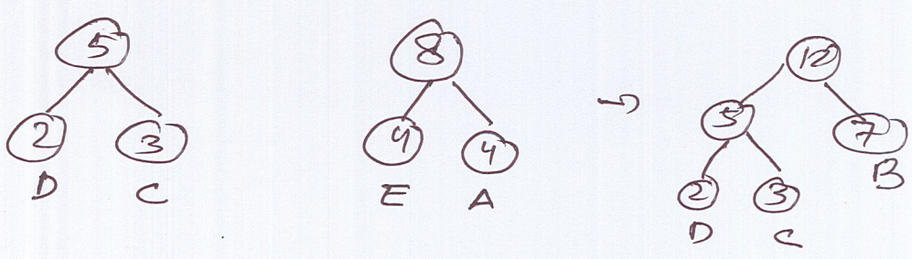
Code = 5 × 3 = 15 bits

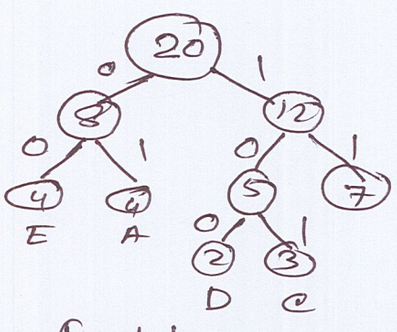
Total = 115 bits

Variable size coding:

Sorting:

D	C	E	A	B
↓	↓	↓	↓	↓
2	3	4	4	7





Final tree

A — 01 = 2 × 4 = 8

B — 11 = 2 × 7 = 14

C — 101 = 3 × 3 = 9

D — 100 = 3 × 2 = 6

E — 00 = 4 × 2 = 8

45 bits

Message Size = 45 bits

For table:

Charactor = 8bits Code = 12 bits

= 5×8 = 40bits

total = 52 bits

So, total size message + table = 45 + 52 bits.

= 97 bits

এমন থাকতে হবে:

Prefix Code: No code is prefix of another code

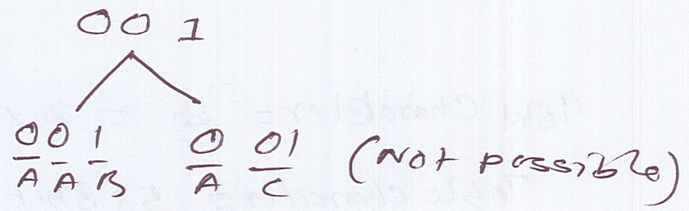
Like →

A = 0

B = 1

C = 01

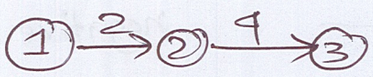
↳ prefix of A So, it not possible



Dijkstra's Algorithm:

Dijkstra's shortest path algorithm is similar to that of Prim's algorithm as they both rely on finding the shortest path locally to achieve the global solution. However, unlike prim's algorithm, the dijkstra's algorithm does not find the minimum spanning tree; it is designed to find the shortest path in the graph from one vertex to other remaining vertices in the graph. Dijkstra's algorithm can be performed on both directed and undirected graphs.

Since the shortest path can be calculated from single source vertex to all the other vertices in the graph, Dijkstra's algorithm is also called **single-source shortest path algorithm**. The output obtained is called **shortest path spanning tree**.



$1 \to 2 = 2 \quad 1 \to 3 = \infty$ (No direct path)

$1 \to 3 = \infty \Rightarrow 1 \to 2, 2 \to 3 = 2 + 4 = 6$

(Relaxation)

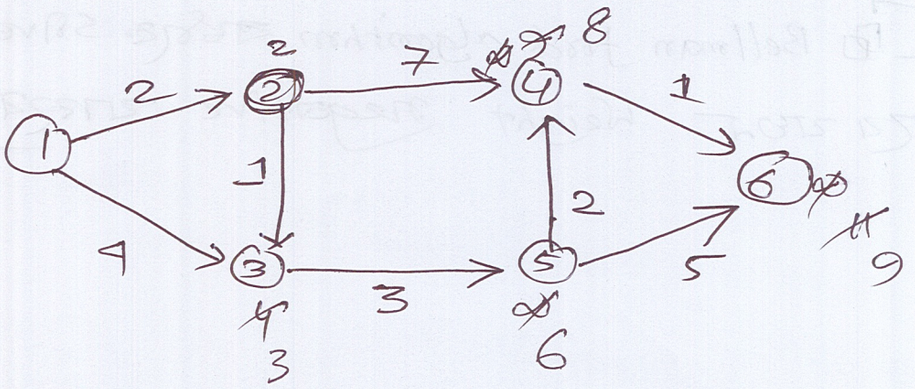
if $(d[u] + cost(u,v) < d[v])$

$d[v] = d[u] + cost(u,v)$

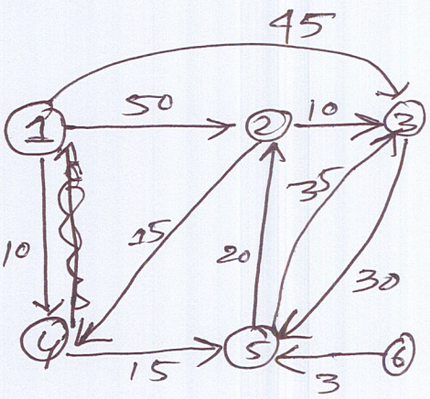
distance of $d[u] = 2$,

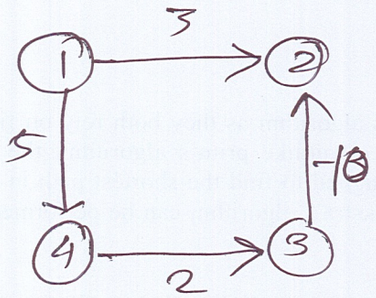
$cost(u,v) = 4$ So, $2 + 4 < \infty$

Solve:



Selected vertex	2	3	4	5	6
4	50	45	10	∞	∞
5	50	45	10	25	∞
2	45	45	10	25	∞
3	45	45	10	25	∞
6	45	45	10	25	28





[Note: Dijkstra Algorithm Cannot consider Negative Weight]

[may work or may NOT work with negative weight]

Start	2	3	4
2	3	∞	5
4	3	∞	5
3	3	7	5
	-7		

[Bellman ford algorithm কখন solve হবে পারবে যখন weight negative থাকবে]

Solve: $T(n) = 3T(\frac{n}{4}) + n$ where $T(1) = 2$

$$T(n) = 3T(\tfrac{n}{4}) + n$$

$$= 3^2 T(\tfrac{n}{4^2}) + n + \tfrac{n}{4}$$

$$= 3^3 T(\tfrac{n}{4^3}) + n + \tfrac{n}{4} + \tfrac{n}{4^2}$$

$$\vdots$$

$$= 3^k T(\tfrac{n}{4^k}) + n + \tfrac{n}{4} + \tfrac{n}{4^2} + \cdots + \tfrac{n}{4^{k-1}}$$

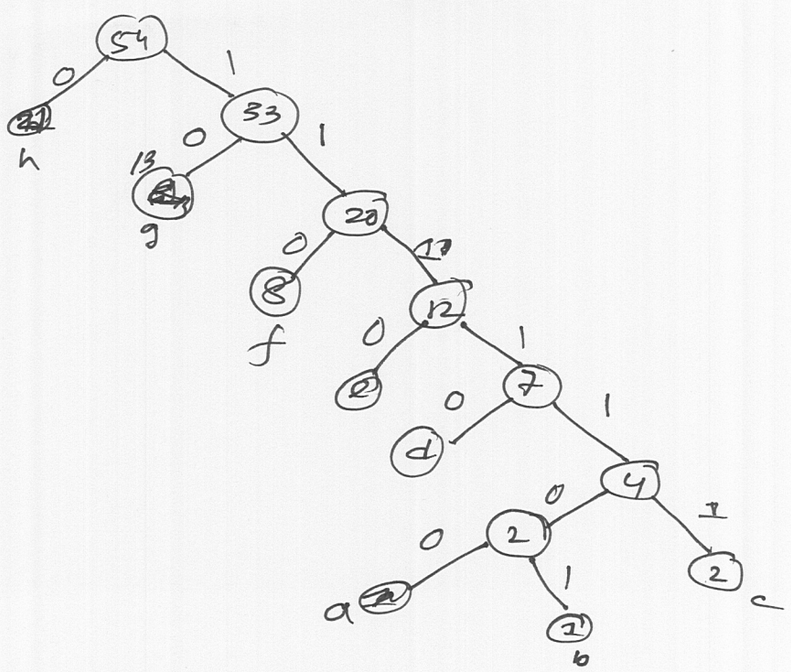
$$= 3^k T(1) + n\left[1 + \tfrac{1}{4} + \tfrac{1}{4^2} + \cdots + \tfrac{1}{4^{k-1}}\right]$$

$$= 3^k \cdot 2 + n\left[\frac{(\frac{1}{4})^k - 1}{\frac{1}{4} - 1}\right]$$

$$= O(n)$$

Solve: A Huffman frequency is a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13 h: 21

Sending Code: 11011111001110



$\underbrace{110}_{f}$ $\underbrace{11110}_{d}$ $\underbrace{0}_{h}$ $\underbrace{1110}_{e}$ $\underbrace{10}_{g}$