

## Algorithm Analysis

### Time Complexity analysis:

মেঘান একটি Algorithm কে তার input এর Mathematical function এর দ্বারা depend করে। ইচ্ছা করা হয়।

সুলভ Loop এর দ্বারা depend করা।

$O(n)$ ,  $W(n)$ ,  $\Theta(n)$  ইত্যাদি নিখুঁত একাক করা হয়।

Example:

\* `for(i=1; i<=n; i++)`

```
{   printf("*"); }
```

এখানে time complexity  $O(n)$

\* `for(i=1; i<=n; i++)`

```
{   for(j=1; j<=n; j++) }
```

```
{   printf("*"); }
```

```
}
```

```
}
```

এখন time complexity  $O(n^2)$

```
*for (i=1; i<=n; i++)
```

```
{
```

```
    for (j=1; j<=n; j++)
```

```
{
```

$O(n^2)$

```
}
```

```
for (i=1; i<=n; i++)
```

```
{
```

$O(n)$

```
}
```

ଅଧ୍ୟାତ୍ମ 1st segment ଅରୁ time complexity  $O(n^2)$

ଅଧ୍ୟାତ୍ମ 2nd segment ଅରୁ time complexity  $O(n)$

ଏହିକମ୍ ସାଙ୍ଗଜ୍ଞ ଯେଉଁ କମ୍plexity କଣିକା କରିବାରୁ count

ପାଇଁ ହେଲା ।

⑥

```

*for ( i=1; i<=m; i++)
{
    for ( j=1; j<=n; j++)
    {
        { }
    }
}

```

এই কোডের time complexity হবে  $m \times n$  অর্থাৎ  $O(mn)$

এই কোডের time complexity এবং সময়ের নির্গত depend

Time complexity কোন numbers এবং উপর নির্ভুল নাবাল নাই।

মেমরি:

```

for ( i=1; i<=m+10; i++)
{
    for ( j=1; j<=n/2; j++)
    {
        { }
    }
}

```

এখন,  $(m+10) \times n/2 = \frac{1}{2}(n^2 + 10n)$

এখন এই total time complexity এবং variable  
কিন্তু এইটো

ଶ୍ରେଷ୍ଠଗତ depend କାହେଁ ଅତିରିକ୍ତ ଏକ କ୍ଷେତ୍ରରେ  $n^2 + n$  କିମ୍ବା  
ଏଥାଣେ  $n^2 > n$  ଅତିରିକ୍ତ time complexity ହେବାରେ  
 $O(n^2)$

ସୁଧାରଣା:

$O(1)$

$O(\log n)$

$O(\sqrt{n})$

$O(n)$

$O(n \log n)$

$O(n\sqrt{n})$

$O(n^n)$

$O(n^n \log n)$

.

.

.

$O(2^n)$

$O(n!)$

$O(n)$

$O(n^2)$

$O(n^3)$

$O(n^4)$

Polynomial  
algorithm

$O(\log n)$

$O(n \log n)$

$O(n^n \log n)$

logarithm  
algorithm

N.B.:

```
for (i=1; i<=n; i=i/2)
{
    // some statements
}
```

এই একটি time complexity হবে  $O(\log n)$

⑥

## Linear Search

■ যদি data set sorted না থাকে।

■ একটা loop নিচে অঙ্গী element check করতে  
হবে যতক্ষণ না key element পাওয়া না পায়।

■ Time complexity  $O(n)$

simulation:

search 5 in this dataset :-

3 9 4 2 1 6 5 8

Hence,

3 9 4 2 1 6 5 8  
↓  
 $3 \neq 5$

= 9 4 2 1 6 5 8  
↓  
 $9 \neq 5$

= 4 2 1 6 5 8

= 4 2 1 6 5 8  
↓  
 $4 \neq 5$

= 2 1 6 5 8

↓  
 $2 \neq 5$

= 1 6 5 8

↓  
 $1 \neq 5$

= 6 5 8

↓  
6 ≠ 5

= 5 8

5 == 5

∴ 5 has been found.

Code:

```
# include <stdio.h>
```

```
int main()
```

```
{ int arr [8] = { 3, 9, 4, 2, 1, 6, 5, 8 };
```

```
int n=5, i, c=0;
```

```
for (i=0; i<8; i++)
```

```
{ if (arr[i] == 5)
```

```
{
```

```
c=1;
```

```
break;
```

```
}
```

```
}
```

```
if (c == 1) printf ("found");
```

```
else printf ("not found");
```

```
    return 0;
```

```
}
```

## Binary Search

- In data set increasing or decreasing sorted थाकू।
- data set के उपरोक्त कल्पना थाकू अवश्यक है।  
— ना mid point के key element के पास है।
- Time complexity  $O(\log n)$

Example:

Search 8] in the following dataset:

⑨

Hence,

$$\{ \begin{array}{ccccccc} 2 & 6 & 13 & 21 & 36 & 47 & 63 & 81 & 92 \end{array} \} \\ \text{36 } L \text{ 81}$$

$$= \begin{array}{cc} 47 & 63 \end{array} \} \begin{array}{cc} 81 & 92 \end{array} \\ \text{63 } L \text{ 81}$$

$$= \begin{array}{cc} 81 & 92 \end{array} \\ 92 > 81$$

$$= \underline{\begin{array}{cc} 81 \end{array}}$$

$$81 = 81$$

$\therefore 81$  has been found.

(22)

code:

```
#include <stdio.h>
int binary-search (int a[], int n, int key)
{
    int start = 0, end = n - 1, mid;
    while (start <= end)
    {
        mid = (start + end) / 2
        if (key > a[mid])
            start = mid + 1;
        else if (key < a[mid])
            end = mid - 1;
        else return mid;
    }
    return -1;
}
```

## Insertion sort

এখন loop এর কাজে অন্তিম element check রয়েছে

কাজে।

Time complexity  $O(n^2)$

Slowest sorting algorithm.

যদি কোন প্রক্ষেত্রে stability and less memory প্রয়োজন হয়ে থাকে তাহলে এটি use করা better.

Simulation:

$$1 \quad 6 \quad 2 \quad 5 \quad 7$$

$\boxed{1 < 6}$

$$= 1 \quad 6 \quad 2 \quad 5 \quad 7$$

$\boxed{1 < 2}$

$$= 1 \quad 6 \quad 2 \quad 5 \quad 7$$

$\boxed{1 < 5}$

$$= 1 \quad 6 \quad 2 \quad 5 \quad 7$$

$\boxed{1 < 7}$

$$= 1 \quad 6 \quad 2 \quad 5 \quad 7$$

$\boxed{6 > 2}$

$$= 1 \quad 2 \quad \boxed{6 > 2} \quad 5 \quad 7$$

= 1 2 5 6 7  
6 < 7

= 1 2 5 6 7 → sorted

= 1 2 5 6 7  
1 < 2 & 2 < 5

Code:

```
#include <stdio.h>
void insertion-sort(int arr[], int n)
{
    int i, key, j;
    for (i=1 ; i<n ; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j+1] = key;
    }
}
```

## merge sort

- Uses divide and conquer method
- stable sorting algorithm অর্থাৎ sorting করার position  
তিনিই বাস্তব।
- recursively mid টিকে থাকলে যাতেই তা পর্যন্ত every  
single element আলাদা তা হয়, তাহলে sort করা মেরা  
কুণ্ডলী
- extra memory লাগে
- Time complexity worst case & average case
- $O(n \log n)$
- STL function: stable\\_sort(array, array+size)
- simulation:

1 4 6 9 8 3 4 5 2  
 = 1 4 6 9 8    3 4 5 2  
 = 1 4 6    9 8    3 4    5 2  
 = 1 4    6    9    8    3 4    5 2  
 = 1 4 6 9 8 3 4 5 2  
 = 1 4 6 8 9 3 4 5 2  
 = 1 4 6 8 9 2 3 4 5  
 = 1 4 6 8 9 2 3 4 5  
 = 1 4 6 8 9    2 3 4 5

(15)

$$= \boxed{1 \ 2 \ 3 \ 4 \ 4 \ 5 \ 6 \ 8 \ 9} \rightarrow \text{sorted}$$

Code:

```
#include <stdio.h>
void mergeSort(int arr[], int l, int m, int r)
{
    int i, j, k, n1, n2;
    n1 = m - l + 1;
    n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
        arr[k] = L[i++];
    while (j < n2)
        arr[k] = R[j++];
}
```

{  
     $\text{arr}[k] = R[j];$   
     $j++;$

}

$k++;$

}

while ( $i < n_1$ )

{  
     $\text{arr}[k] = L[i];$   
     $i++;$   
     $k++;$

}

while ( $j < n_2$ )

{  
     $\text{arr}[k] = R[j];$   
     $j++;$   
     $k++;$

(17)

void mergesort (int arr[], int l, int r)

{

if ( $l < r$ )

{

int  $m = \lfloor (n-1)/2 \rfloor;$

mergesort (arr, l, m);

mergesort (arr, m+1, r);

merge (arr, l, m, r);

}

}

## Quick Sort

- Divide and conquer method କେ ସବୁ ।
- Last element କେ Pivot ସିରାମୋ ଏବଂ first element କେ marker ସିରାମୋ । marker & pivot ଏବଂ ମର୍କେ compare ହେ । ଯଦି marker, pivot ଏବଂ ଚତୁର୍ଥ ବାଟୁ ହୁଏ ତାହାରେ marker, pivot ଏବଂ position ଯାଏ ଏବଂ pivot ଏହାରେ ବାଟୁରେ element ଏବଂ ଆଗମ୍ବା ଯାଏ ଏବଂ pivot ଏହାରେ ବାଟୁରେ element ଏବଂ position ଯାଏ ।  
ବାଟୁରେ element କେ marker ଏବଂ position ଯାଏ ।  
ଆବୁ ଯଦି marker & pivot ତମାନ ହୁଏ ଅର୍ଥରେ  
marker, pivot ଏବଂ ଚତୁର୍ଥ ବାଟୁ ହୁଏ ତାହାରେ marker  
ଏବଂ ପାଇଁରେ element କେ marker ହୁଏ ଯାଏ । marker  
& pivot ଯାଏନ ଏବଂ ହୁଏ ଯାଏ ତଥା କୌଣସି ଯାଏ ଏବଂ  
କେ element ଯାଏ । ତଥା ଆଗମ୍ବା ୨୭୩ ବାଟୁ ଉପରେ  
ଥାବେ । left subarray ଓ same process କରାଯାଇ  
ଏବଂ recursively କଲାଏ ଯାଏ ।

- Time complexity average case -  $O(n \log n)$
- worst case -  $O(n^n)$  . କିନ୍ତୁ average case  
ଏବଂ worst case -  $O(n \log n)$  .
- ଅଳକାକାର fast - କାହାରେ ବାଟୁ ।

Code:

```
#include <stdio.h>

void swap (int * a, int * b)
{
    int t = * a ;
    * a = * b ;
    * b = t ;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++)
        if (arr[j] <= pivot)
    {
        i++;
        swap (&arr[i], &arr[j]);
    }
}
swap(&arr[i+1], &arr[high]);
return (i+1);
```

21

```
void quicksort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}
```



22

## Greedy

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice with the hope of finding a global optimum.

- Optimum.
- By problem greedy ফিল্ড solve করা যাবে। DP  
ফিল্ড solve করা যাবে। যা DP ফিল্ড solve করা যাবে।  
ফিল্ড solve করা যাবে। যা DP ফিল্ড solve করা যাবে।  
অতিরিক্ত পাই এবং নাও  
অতিরিক্ত পাই এবং নাও
- Greedy complexity DP' এ অনন্য অংশ।  
Greedy ক্ষেত্র অংশ এবং চলিং অংশ।  
Greedy ক্ষেত্র অংশ।

## Greedy Coin change Problem

minimum coins we require  
value राशी यादे तो हम करते हैं।

simulation:

Given coins value : 10, 5, 1

Required value = 71

C.V.      Coin      Value  
10 x 7 → 70

1 x 1 → 0

minimum coin required 8 and the coins  
are 10 10 10 10 10 10 10 1

Code:

```
#include <bits/stdc++.h>
```

```
int arr[] = {1, 5, 10};
```

```
int n = size of (arr) / size of (arr[0]);
```

```
void findMin(int v)
```

```
{     vector<int> ans;
```

```

for(int i = n-1; i >= 0; i--) {
    {
        while (v >= arr[i]) {
            {
                v = arr[i];
                ans.push_back(arr[i]);
            }
        }
    }
}

for (int i = 0; i < ans.size(); i++) {
    {
        printf("%d ", ans[i]);
    }
}
}

int main() {
    int n = 7;
    findMin(n);
    return 0;
}

```

## Greedy Bin packing Problem

মুক্তি First এ sort করতে হবে।

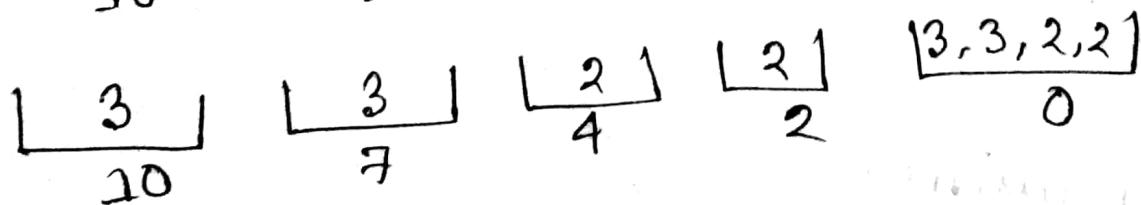
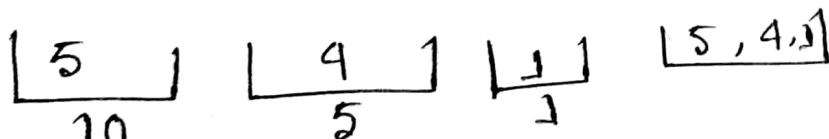
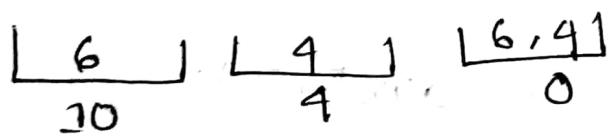
পুরুষ Then অবচেতনা করি value কে pack করতে হবে,  
গুরুত্বের পূর্বের value কে pack করতে হবে। এবং  
চলতে আবশ্য। first bin এর আজো full করতে  
হবে। check করতে কোন value গুলো fit হবে।

simulation:

5 3 2 4 6 2 1 4 3

After sort:

6 5 4 1 4 3 3 2 2 1



Code:

```
#include <bits/stdc++.h>
using namespace std;
int next fit (int weight[], int n, int c)
{
    int res=0, bin-rem=c;
    for (int i=0; i<n; i++)
    {
        if (weight[i] > bin-rem)
        {
            res++;
            bin-rem = c - weight[i];
        }
        else
            bin-rem = weight[i];
    }
    return res;
}
```

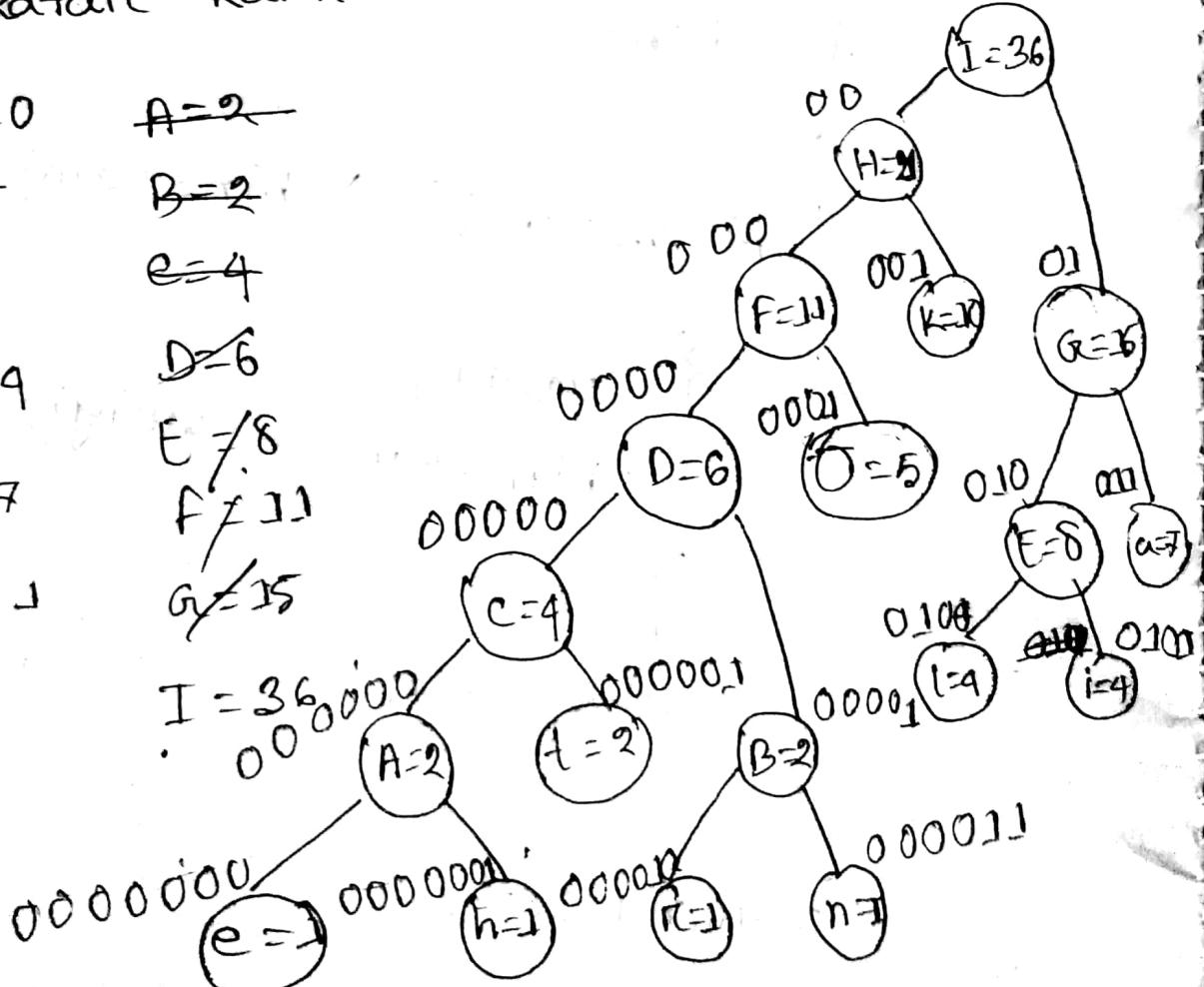
## Huffman Coding

- Min heap tree
- Data compression algorithm
- variable length coding
- ସିଟି ଚାରାଟି ଏବଂ ପ୍ରେଫିକ୍ସି କୋଡ୍ ଅଳ୍ପ ରହା  
ଚାରାଟି ଏବଂ ପ୍ରେଫିକ୍ସି କୋଡ୍ ଏବଂ ଜାମ୍ବେ ମିଳିଲା ।
- smallest size ରୁ ନିଯନ୍ତ୍ରଣାତ୍ମକ node ଏବଂ ଅଳ୍ପ

simulation:

Kolikatare kalikanto kaka kakoli ke kohilo

$k=10$	$A=2$
$O=5$	$B=2$
$L=4$	$E=4$
$i=9$	$D=6$
$a=7$	$F=8$
$R=1$	$J=11$
$n=3$	$G=15$
$t=2$	$I=36$
$c=3$	$0000$
$h=1$	$00000$



K=001

0 = 000 1

$$l = 0.100$$

i = 0.101

$$a=0.11$$

$$\pi = 000010$$

$$n = 000011$$

$$t = 000001$$

$$e = 0000000$$

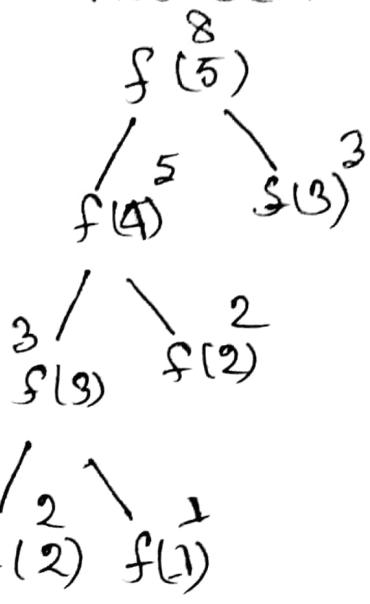
$$h = 0\ 000001$$

Algorithm 1st character node find 2nd  
 character node find min value sum  
 character node build tree build  
 character node find sum

হবে।  
 Root এর value ০ হালতেও হবে। প্রথমে যদি আসে  
 Root এর value ০ হালতেও হবে। প্রথমে যদি  
 Node এর মাঝে ০-সংখ্যা ০ বর্গমিতি হবে। তাহলে  
 Node এর মাঝে ০-সংখ্যা ১ সমান হবে। অতি parent-  
 child মাঝে ০ এর সময় একজন হবে।  
 node এরই process value হলুয়া child node

গুরুত্ব value decide করেছে এবং ।

### Fibonacci Tree without DP



এই দেশটি ~~মাত্র~~ value সূলা store করে রাখা হয়। অতিক্রম  
new করে যেৱে ইত্যাদি হয় না।

Code:

① int fib(int n)

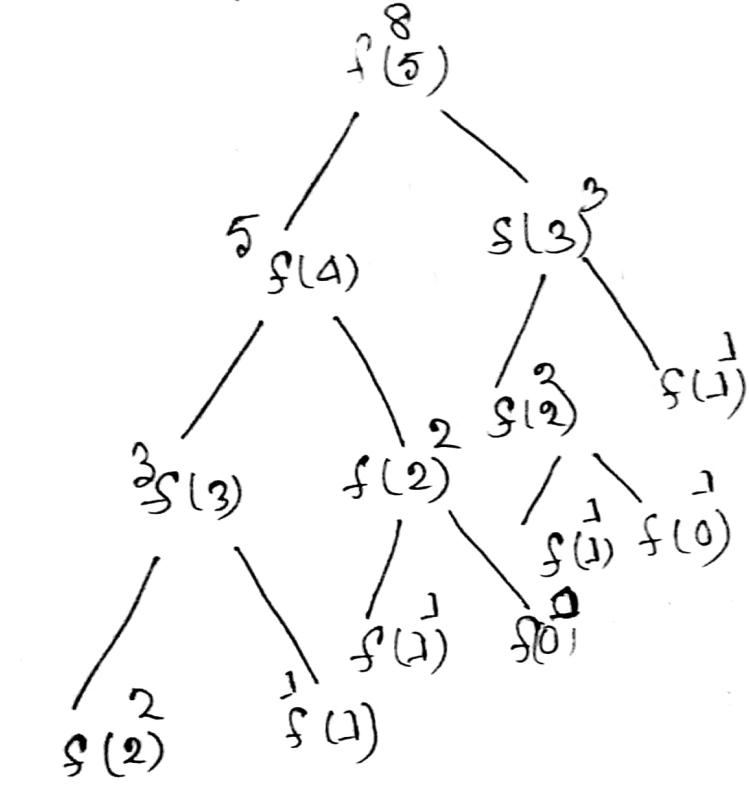
```

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

22  
27/7/8 3

(32)

Fibonacci tree without DP



ফলাফল অন্তিমের জন্য fib value হবে 1  
গুরুত্ব

কোড়া:

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

# Longest Increasing Subsequence

এটি sequence থেকে সর্বচেয়ে লম্বা subsequence  
কৈবল্য করে যেখানে increasing order'তে numbers (সংখ্যা)

থাকে।  
এতি number'তে value' বরাবর। তারপর অটোগুলোকে একে একে পৃথক করে আবেদন করে।  
গুরুত্বে একে একে পৃথক করে আবেদন করে।  
simulation: আবার কৈবল্য করতে হবে।

3	7	2	5	9	1	8	3	6
v	x	v	x	v	v	x	x	x
2v	2x	2v	2x	2v	3v	2v	2v	2v

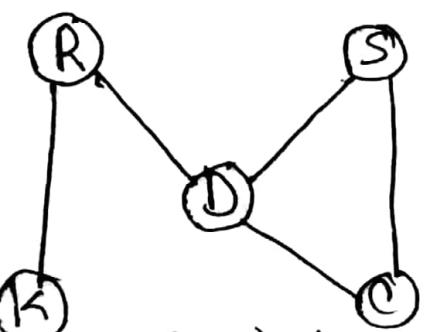
# Graph

Properties:

1. Node / vertex, অন্তর্ভুক্ত বিন্দু - must have node
2. Edge, অন্তর্ভুক্ত দুটি node connect করা লাইন
3. Direction
4. Weight

Representation:

1. Adjacency matrix / Adj. matrix
2. Adjacency list / Adj. list
3. Edges list



Adjacency Matrix:

D	C	S	R	K	
D	0	1	1	1	0
C	1	0	1	0	0
S	1	1	0	0	0
R	1	0	0	0	1
K	0	0	0	1	0

(46)

complexity:  $O(v^n)$

Adjacency list:

D: C R S

C: D S

R: K

S: D C

K: R

complete graph: যদি ক্লোন আছে  $v$  সংখ্যক vertex

এবং এল  $\frac{v(v-1)}{2}$  সংখ্যক Edge আছে তাহলে তাৰ

complete graph বলে।

Dense graph: যদি ক্লোন আছে  $v$  সংখ্যক vertex

এবং এল  $\frac{v(v-1)}{2}$  প্ৰায় Half অথবা তাৰ চেয়ে বেশি থাকে edge

তাকে Dense graph বলে।

Sparse graph: যদি ক্লোন আছে  $v$  সংখ্যক vertex

এবং এল  $\frac{v(v-1)}{2}$  প্ৰায় Half প্ৰায় কম edge থাকে

তাহলে তাকে sparse graph বলে।

Dense graph হলে Adjacency matrix use  
করুণ হবে।

Sparse graph হলে Adjacency list use করুণ  
হবে।

যখন বাস্তবাত্মক query বস্তু হয় তখন Adj. Matrix  
use করুণ হবে। এই ক্ষেত্রে ans  $O(1)$  এ পাওয়া যাবে।  
যদি Adj. list use করি তাহলে  $O(V)$  টা আমি আছি  
আজকে।

Directed graph / Unidirected graph:

একটি edge directed হলু পুরোটা directed graph  
~~হবে।~~

Undirected graph / Bidirected graph:

~~Undirected graph~~ না থাকে তাহলে  
edge গুলি directed না থাকে তাহলে

undirected graph হবে।

Edge গুলি weight থাকলে weighted graph হবে।

যদি কোন weighted graph এ কোন edge

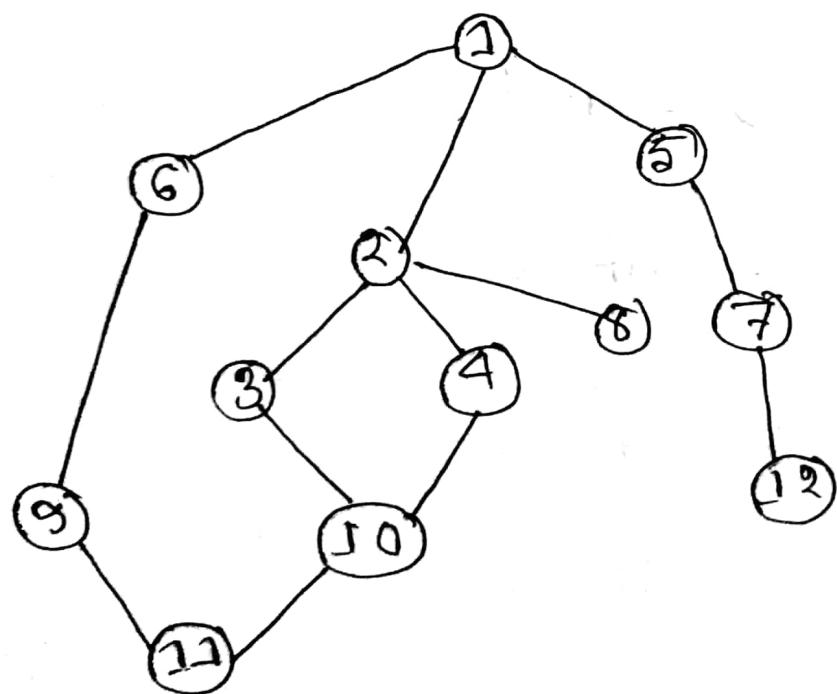
গুলি weight না দেওয়া থাকে তাহলে তা weight

নেওয়া হবে।

## Breadth first search

- shortest path এর কার্যান্বয় Algorithm
- অস্তি graph unweighted হয় তাহলে BFS ব্যবহার
- কার্যত হবে।
- অস্তি অতি node এর edge গুলোর weight 3000  
হয় তাহলে BFS কাজ করবে। Time complexity  $O(V+E)$

simulation:



Q

dist

1 - INF 0

2 - INF 1

3 - INF 2

4 - INF 2

5 - INF 1

6 - INF 1

7 - INF 2

8 - INF 2

9 - INF 2

10 - INF 3

11 - INF 3

12 INF 3

```

#include <queue>
using namespace std;
queue<int>Q;
Q.push(s);
dist[s]=0;
while (!Q.empty())
{
    x = Q.front();
    Q.pop();
    for (all y adj. to x)
    {
        if (dist[y] == INF)
        {
            dist[y] = dist[x]+1;
            Q.push(y);
        }
    }
}

```

functions of queue

- Q.empty();
- Q.push(y);
- Q.front();
- Q.pop();

এখন, প্রথম source push করা হইয়েছে। এবং  
source এর সাথে connected node push  
করা হইয়ে আবার source node pop করা।  
জোবে অঙ্গীর্ণ Node visit করা হইল।

Code:

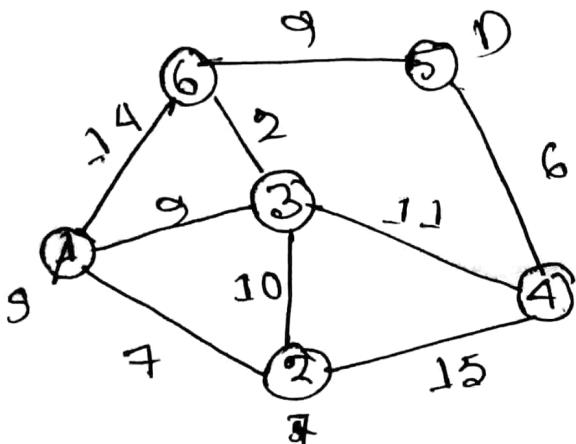
```
graph TD; A((A)) --> B((B)); A --> C((C)); A --> D((D)); B --> E((E)); B --> F((F)); C --> G((G)); C --> H((H)); D --> I((I)); D --> J((J)); E --> K((K)); E --> L((L)); F --> M((M)); F --> N((N)); G --> O((O)); G --> P((P)); H --> Q((Q)); H --> R((R)); I --> S((S)); I --> T((T)); J --> U((U)); J --> V((V)); K --> W((W)); K --> X((X)); L --> Y((Y)); L --> Z((Z)); M --> AA((AA)); M --> BB((BB)); N --> CC((CC)); N --> DD((DD)); O --> EE((EE)); O --> FF((FF)); P --> GG((GG)); P --> HH((HH)); Q --> II((II)); Q --> JJ((JJ)); R --> KK((KK)); R --> LL((LL)); S --> MM((MM)); S --> NN((NN)); T --> OO((OO)); T --> PP((PP)); U --> QQ((QQ)); U --> RR((RR)); V --> YY((YY)); V --> ZZ((ZZ)); W --> AA((AA)); W --> BB((BB)); X --> CC((CC)); X --> DD((DD)); Y --> EE((EE)); Y --> FF((FF)); Z --> GG((GG)); Z --> HH((HH));
```

53

## Dijkstra

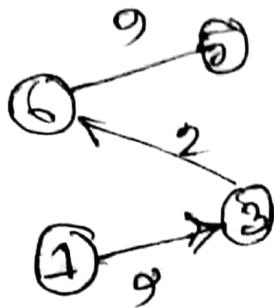
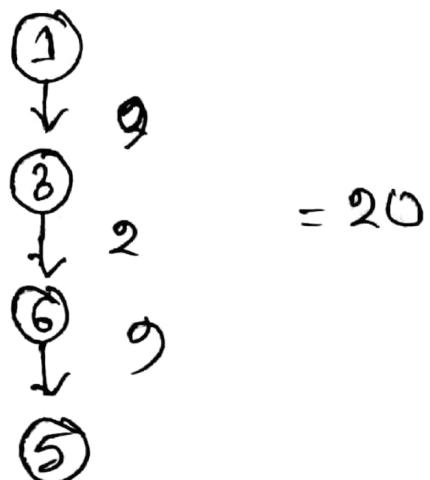
- single source shortest path algorithm
- Greedy Algorithm.
- অবস্থার কাট করে,
- Time complexity  $O(V^2)$

simulation:



	dist	parent
1	INF 0	
2	INF 7	
3	INF 9	1
4	INF 22 20	2 3
5	INF 20	6
6	INF 14 11	* 3

shortest path:

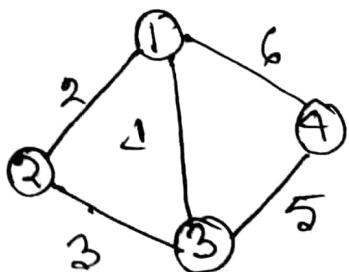


৫) শর্যানুসৰি মাত্র IN  
ভিয়োচন source থেকে start করতে হবে। শ্রয়চালনার মাত্র  
বিধৃত হবে। এবুপুর source min. dist. এর node visit  
করবে, যদি better result হয় তাহলে result change  
করবে। প্রেতে পর্যন্ত node টাঙ্গা check করবে,  
করবে। প্রেতে পর্যন্ত node টাঙ্গা check করবে,  
এবং এই node এর value  
এবং এই node এর visiting node এর value  
হ্যাজ করবে। এবং better result টা রাখবে।  
যোগ করবে। এবং destination  
shortest path এর করার চেষ্টা Destination  
থেকে start করবে। তার parent কে আ দেখা,  
এবং parent এর parent কে আ দেখা।  
এবং parent এর parent কে আ দেখা।

## Floyd Warshall Algorithm

- All pair shortest path algorithm
- Time complexity  $O(V^3)$
- Adj. matrix use করতে হবে এবং মেমোরি  
থাকলে তা দিয়ে replace করতে হবে।

simulation:



	1	2	3	4
1	$\infty$	2	1	6
2	2	$\infty$	3	$\infty$
3	1	3	$\infty$	5
4	6	$\infty$	5	$\infty$

```
for( k=1; k<=n; k++)
```

{

```
    for( i=1; i<=n; i++)
```

{

```
        for( j=1; j<=n; j++)
```

{

$$\text{dist}[i][j] = \min(\text{dist}[i][j], \\ \text{dist}[i][k] + \text{dist}[k][j]);$$

}

{

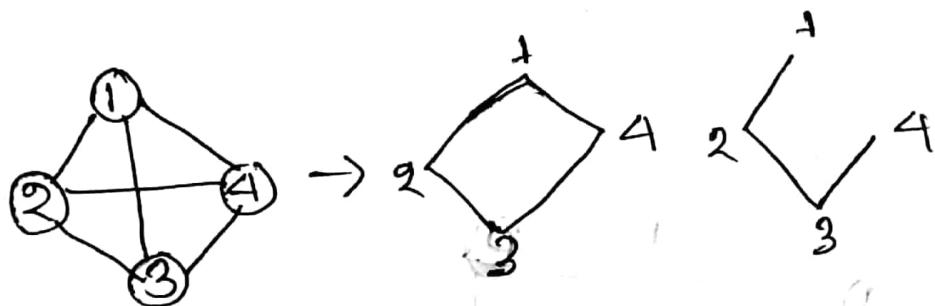
}

## minimum spanning Tree

\* Tree: এক বিঃগেরূ graph যার মধ্যে জগন Loop  
নই কিন্তু connected.  $\rightarrow$

জগন Tree পুরু node থাকব।  
থাকল  $(V-1)$  পুরু edge থাকব।

\* Spanning: ~~মুক্ত~~ Spanning Tree বা graph  
হত্তে এল main graph টেরে এব Node  
Touch কৰতে হব।



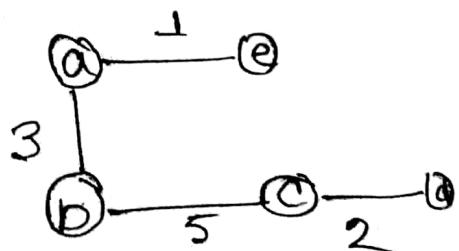
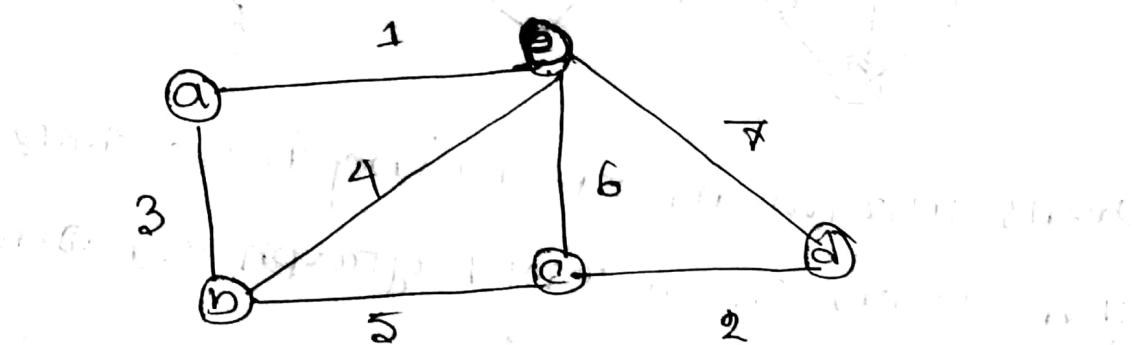
অর্থাৎ Minimum spanning tree বলতু আমুক  
সুপরি গ্ৰাফটো Tree অথবা graph তো তিমল subtree  
সুপরি গ্ৰাফটো Tree অথবা graph তো তিমল subtree  
যা অন্তিম node কো কুৰীয়াত অফেয়ায় touch কৰতা

64

## Kruskal's Algorithm

- Minimum spanning tree algorithm
- Greedy algorithm
- Time complexity  $O(V \log E)$  - যদি min heap ব্যবহার করা হয় তবে  $O(E \log E)$ ,  
→ যদি min heap ব্যবহার করা হয়,

simulation:



$$a - e = 1$$

$$a - d = 2$$

$$a - b = 3$$

$$b - e = 4$$

$$b - c = 5$$

$$e - c = 6$$

$$e - d = 7$$

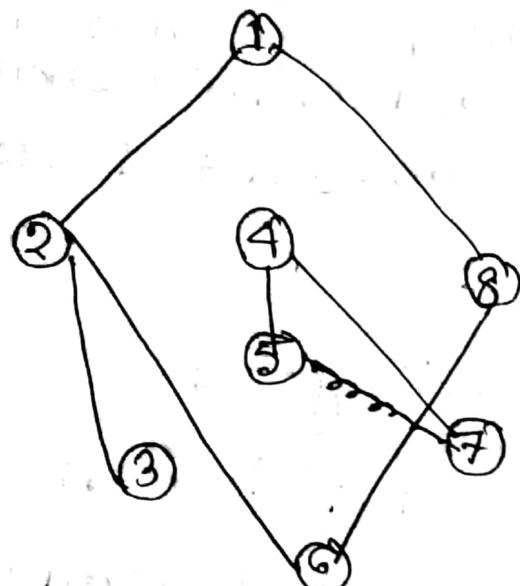
Edge (weight)	comp	cost
5	0	
4	1	
1	3	3
2	2	6
3	1	11
5		

অন্যান min weight অর্থে edge খুলো নিত যদি  
 অন্যান min weight অর্থে edge খুলো নিত যদি loop  
 সহি edge কি loop create না কর্য, যদি loop  
 create করে গোলে কোই edge পাওয়া দিয়ে পার্বত weight  
 create করে গোলে কোই edge পাওয়া দিয়ে পার্বত ।

## Depth First Search

- একটি unweighted graph-এর minimum spanning tree and All pair shortest path
- একটি graph-এ cycle detect করা।
- Path find করা।
- Topological sorting করা।
- কোন graph bipartite বিন্দু। Test করা।
- কোন graph strongly connected components of a graph পাই করা যাব।
- কোন একটি solution পাই puzzle solve করা।
- Time complexity  $O(V+E)$

## simulation:



To visit all the node :

```
void dss (int x)
{
    color [x] = 1;
    for (all y next to x)
    {
        if (color [y] == 0)
            dss (y);
    }
}
```

3

color[i] = -1; // Initially all nodes are uncolored

1  $\emptyset \dashv$

2  $\emptyset \dashv$

3  $\emptyset \dashv$

4  $\emptyset \dashv$

5  $\emptyset \dashv$

6  $\emptyset \dashv$

7  $\emptyset \dashv$

8  $\emptyset \dashv$

To find out whether the graph is disconnected:

```
comp = 0;
```

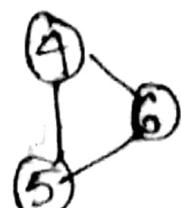
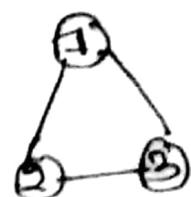
```
for (i=0; i < N; i++)
```

```
{ if (color[i] == 0)
```

```
{ comp++;
```

```
    dfs(i);
```

```
}
```



comp = 2  $\Rightarrow$  जड़े

जड़ीकरण द्वारा disconnected  
graph

To find the cycle in a graph:

cycle = 0;

void dfs(int x, int root)

{

color[x] = 1;

for(all y next to x)

{ if(color[y] == 1 && y != root)

{

cycle = 1;

}

else dfs(y);

}

## Edge classification:

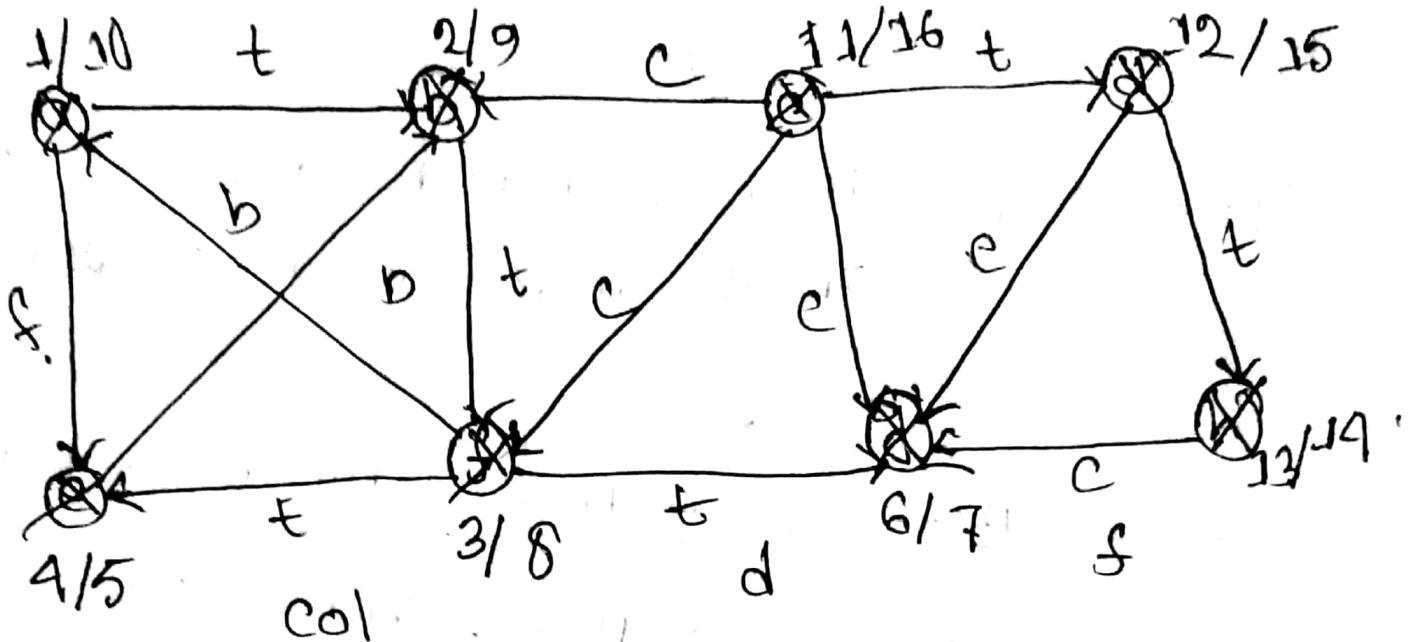
Tree: T  $\rightarrow$  New কোন node discover হ'কুল  
অর্ধে node white আকান পর্টি Tree  
edge.

Back: B  $\rightarrow$  Grey Node লালে Back edge.

Forward: F  $\hookrightarrow$  Black node হলে Parent  
node এবং discover এবং finish  
ing time এবং তার child  
node এবং discover and finish-  
ing time থাকে তাহলে forward  
edge.

Cross: C  $\rightarrow$  Black node হলে স্বতন্ত্র forward  
edge এবং condition fullfil কা-  
রণে তাহলে cross edge.

একটি graph-এ মনে কুল Back edge থাকে তাহলে  
cycle আবশ্য।



a	<del>go G/B</del>	1	10
b	<del>go G/B</del>	2	30
c	<del>go G/B</del>	31	16
d	<del>go G/B</del>	12	15
e	<del>go G/B</del>	4	5
f	<del>go G/B</del>	3	8
g	<del>go G/B</del>	6	7
h	<del>go G/B</del>	13	24

ଅଲୋଗ୍ରାମ ~~visiting~~ Alphabetically visit କରୁଥିଲୁ ,  
 first node ହେଉ adj. node to visit କରୁଥିଲୁ ,  
 ଏଥି ତାଣେ white ଅଛେ grey କେ କରୁଥିଲୁ ପରିବର୍ତ୍ତ୍ୟ  
 ଏହି ନେଉ white node କେ grey  
 ଏହି ନେଉ white node କେ black କରୁଥିଲୁ  
 white node କେ black କରୁଥିଲୁ Node ଏହି  
 parent node କେ visited କରୁଥିଲୁ ,  
 visit କରୁଥିଲୁ ।

void dfs(int x)

```
{
  color[x] = 'G';
  d[x] = ++time;
  for (all y adj. to x)
    if (color[y] == 'W')
```

```
{
  define-edge(x,y);
  {
    dfs(y);
  }
}
```

color[x] = 'B';

s[x] = ++time;

}

(Königstein)