30/01/19

Unit-II

• DIVIDE AND CONQUER

→ Finding sum of 'n' numbers

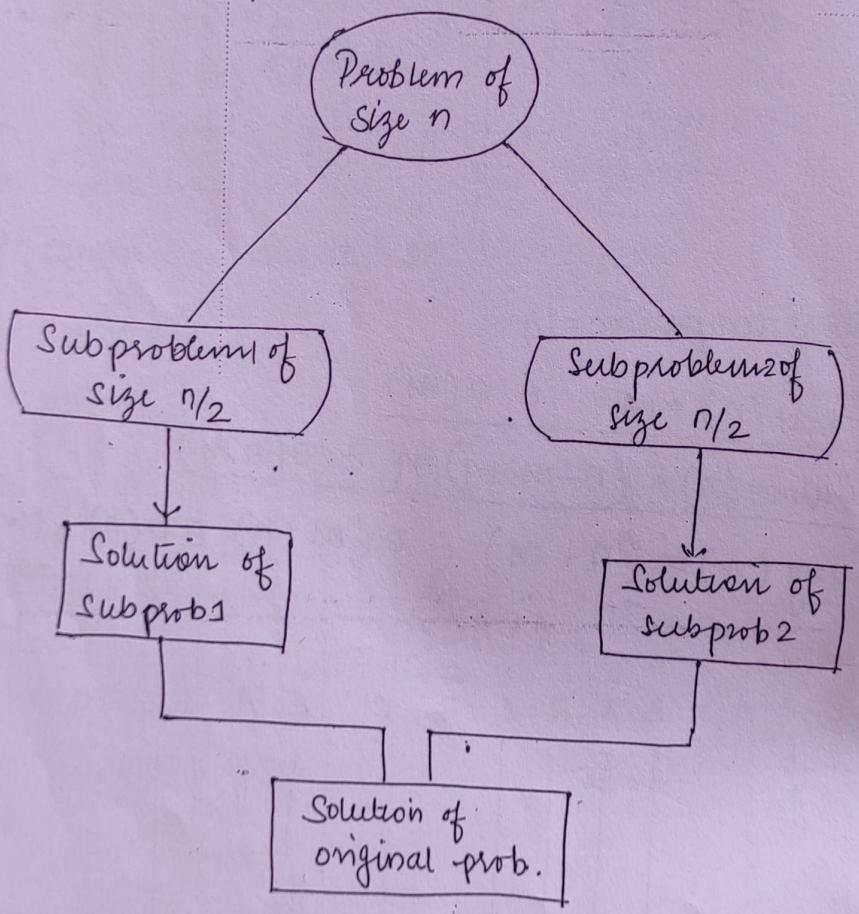
$a_0+a_1+a_2+\ldots\ldots+a_{n-1}$

$= (a_0+a_1+\ldots\ldots+a_{n/2-1})+(a_{n/2}+a_{n/2+1}+\ldots\ldots+a_{n-1})$

$= (a_0+a_1+\ldots\ldots+a_{11})+(a_{12}+a_{13}+\ldots\ldots+a_{24})$

$=(a_0+a_1+\ldots\ldots+a_5)+(a_6+\ldots\ldots+a_{11})+(a_{12}+a_{13}+\ldots\ldots+a_{17})+(a_{18}+a_{19}+\ldots\ldots+a_{24})$

If $n=25$ i.e $\frac{n}{2}=12$.



A problem instance of size 'n' is divided into two instances of size $n/2$. generally an instance of size 'n' can be divided into 'b' instances of size n/b, with 'a' of them need to be solved. ('a' and 'b' are constants with $a \geq 1$ and $b > 1$).

Assuming that size 'n' is a power of 'b', to simplify the analysis we get following recurrence relation for the running time $T(n)$.

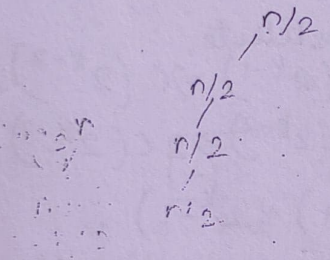
$$\boxed{T(n) = a \cdot T(n/b) + f(n)} \quad \text{— (1)}$$

general divide and conquer recurrence.

where $f(n)$ refers to the time spent on dividing the problem into smaller ones' and combining their solutions.

* Master Theorem: If $f(n) \in \theta(n^d)$ with $d \geq 0$ in recurrence relation (1), then

$$T(n) \in \begin{cases} \theta(n^d), & \text{if } a < b^d. \\ \theta(n^d \log_2 n), & \text{if } a = b^d. \\ \theta(n^{\log_b a}), & \text{if } a > b^d. \end{cases}$$



Eg: Computing sum of 'n' numbers.

Sol: Let us consider $n = 2^k$.

$A(n) = 2 \cdot A(n/2) + 1 \quad \forall \; n > 1$ and $A(1) = 0$

$T(n) = a \cdot T(n/b) + f(n)$

Here, $a = 2, b = 2, f(n) = 1$.

$b^d = 2^0 = 1$

$a > b^d$

i.e $2 > 1$

$A(n) \in \theta(n^{\log_b a})$

$\in \theta(n^{\log_2 2})$

$A(n) \in \theta(n)$

$f(n) \in \theta(n^d)$

$n^d = 1 \Rightarrow d = 0.$

$n^0 = 1.$

$\Rightarrow A(2^k) = 2$

$\boxed{\therefore A(n) \in \theta(n)}$

Eg: $C(n) = 2C(n/2) + (n-1) \quad \forall n > 1$

$C(1) = 0$.

Sol: $a = 2, b = 2, f(n) = n-1$

$b^d = 2^1 = 2 \quad n^d = n-1$

i.e, $n^d \approx n \Rightarrow d = 1$

$a = b^d$.

$\therefore C(n) \in \Theta(n^d \log n)$

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

$\therefore \boxed{C(n) \in \Theta(n \log n)}$

Backward substitution:

Given: $C(n) = 2C(n/2) + (n-1)$

Put $n = 2^k$.

$C(2^k) = 2C(2^{k-1}) + (2^k - 1)$, Substitute

$= 2C(2^{k-2}) + (2^{k-1} - 1) + (2^{k-1} - 1)$ $\quad C(2^{k-1}) = 2C(2^{k-2}) + (2^{k-1} - 1)$

$= 2C(2^{k-3}) + (2^{k-2} - 1) + (2^{k-1} - 1) + (2^k - 1)$ $\quad C(2^{k-2}) \Rightarrow 2C(2^{k-3}) + (2^{k-2} - 1)$

$\vdots$

$= 2C(2^{k-i}) + (2^{k-(i-1)} - 1) + (2^{k-(i-2)} - 1) + \dots$

Put $k - i = 0$

$i = k$

$= 2C(1) + (2^1 - 1) + (2^2 - 1) + (2^3 - 1) + \dots + (2^k - 1)$

$= 2 \times 0 + [2^1 + 2^2 + 2^3 + \dots + 2^k] - [1 + 1 + 1 + \dots k]$.

$= [2^1 + 2^2 + 2^3 + \dots + 2^k] - k$.

$= 2^{k+1} - 1 - 1 - k$.

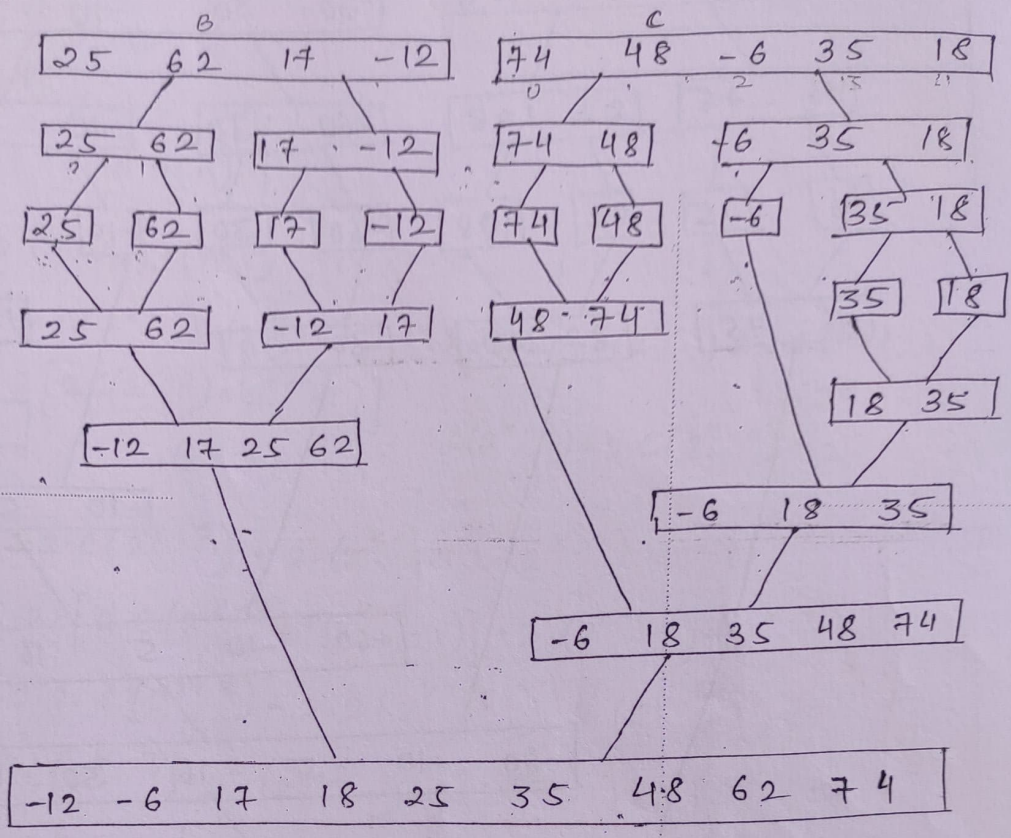
$= 2^{k+1} - 2 - k$.

$C(n) = 2^k \cdot 2 - 2 - k. = 2n - 2 - \log_2 n$.

$= 2(n-1) - \log_2 n$

• Merge sort - A = 25, 62, 12, -12, 74, 48, -6, 35, 18

Eg:
n=9



Algorithm Mergesort (A[0 ... n-1])

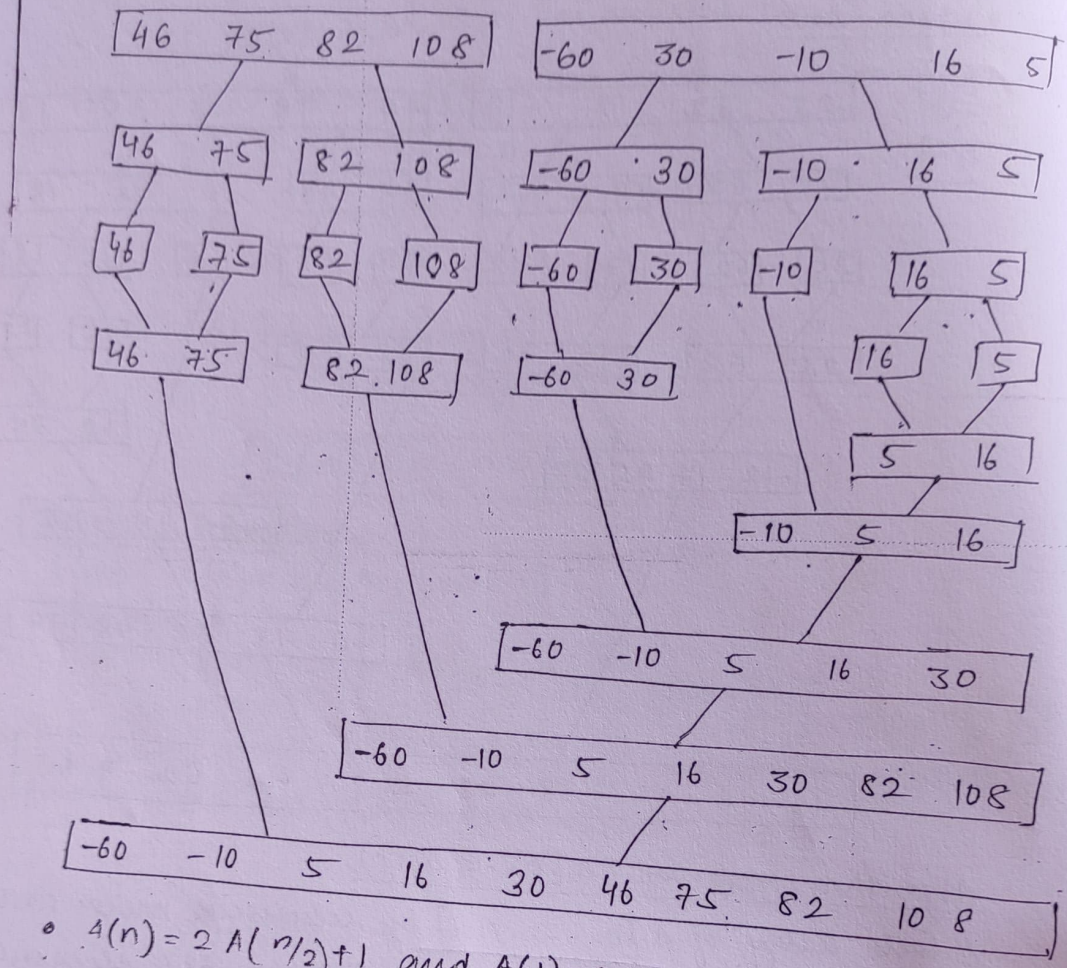
// Sorts array of A[0 ..... n-1] by recursive merge sort.

// Input: An array of A[0 ..... n-1] of orderable elements.

// Output: Array of A[0 ..... n-1] sorted in non-decreasing order.

if (n > 1)
    copy A[0 ..... (n/2)-1] to B[0 ..... [n/2]-1] ;
    copy A[(n/2) ..... (n-1)] to C[0 ..... [n/2]-1] ;
    Mergesort (B[0 ..... [n/2]-1]) ;
    Mergesort (C[0 ..... [n/2]-1]) ;
    Merge (B, C, A) ;

$C(n) \Rightarrow 2(n/2) + 2$



- $A(n) = 2A(n/2) + 1$ and $A(1) = 0$.

Put $n = 2^k$.

$A(2^k) = 2A(2^{k-1}) + 1$, Substitute, $A(2^{k-1}) = 2A(2^{k-2}) + 1$

$= 2[2A(2^{k-2}) + 1] + 1$ $\quad\quad$ $A(2^{k-2}) = 2A(2^{k-3}) + 1$

$= 4A(2^{k-2}) + 2 + 1 = 2 \cdot 2 [2A(2^{k-3}) + 1] + 2 + 1$

$= 2 \cdot 2 \cdot 2 A(2^{k-3}) + 4 + 2 + 1$

$= 2^3 A(2^{k-3}) + 2^2 + 2^1 + 2^0$

$= 2^3 [2A(2^{k-4}) + 1] + 2^2 + 2^1 + 2^0$

$= 2^4 A(2^{k-4}) + 2^3 + 2^2 + 2^1 + 2^0 + 2^4$

$\vdots$

$= 2^i A(2^{k-i}) + [2^1 + 2^0 + 2^3 + 2^4 + \ldots\ldots + 2^k]$

Put $k - i = 0$.

$k = i$

$= 2^k A(2^0) + [2^{k+1} - 1]$

$= 2^k \cdot 0 + (2^{k+1} - 1)$

$\therefore 2^k \cdot 2 - 1 = 2n - 1 \approx 2n$

$\therefore \boxed{A(n) \approx 2n \in \Theta(n)}$

• $C(n) = 2C(n/2) + (n-1) \quad \therefore C(1) = 0 \quad [n-1 = n]$.

$C(2^k) = 2C(2^{k-1}) + (2^k - 1)$, Substitute, $C(2^{k-1}) = 2C(2^{k-2}) + (2^{k-1} - 1)$

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

$C(2^{k-2}) = 2C(2^{k-3}) + (2^{k-2} - 1)$

$= 2 \cdot 2\, C(2^{k-2}) + 2 \cdot (2^{k-1} - 1) + (2^{k-1})$

$= 2 \cdot 2 \cdot \left[2C(2^{k-3}) + (2^{k-2} - 1)\right] + 2 \cdot (2^{k-1} - 1) + (2^{k-1})$.

$= 2 \cdot 2 \cdot 2 \cdot C(2^{k-3}) + 2 \cdot 2 (2^{k-2} - 1) + 2 \cdot (2^{k-1} - 1) + (2^{k-1})$

$= 2^3 C(2^{k-3}) + 2^2 (2^{k-2} - 1) + 2^1 (2^{k-1} - 1) + 2^0 (2^k - 1)$

$\vdots$

$= 2^i C(2^{k-i}) + 2^{i-1} (2^{k-(i-1)} - 1) + 2^{i-2} (2^{k-(i-2)} - 1) \ldots$

Put $k - i = 0$

$i = k$.

$= 2^k C(2^0) + 2^{k-1}(2 - 1) + 2^{k-2}(2^2 - 1) + \ldots$

$= 2^k C(1) + (2^k - 2^{k-1}) + (2^k - 2^{k-2}) + \ldots$

$[k^k + 2^k + 2^k + \ldots \ldots n \cdot 2^k] - [2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots + 2^0]$

$\rightarrow C(2^k) = 2C(2^{k-1}) + 2^k$

$= 2\{2C(2^{k-2}) + 2^{k-1}\} + 2^k$

$= 2^2 \cdot C(2^{k-2}) + 2^k + 2^k$

$\vdots$

$2^i C(2^{k-i}) + i \cdot 2^k$.

$2^k C(1) + k \cdot 2^k$.

$2^k \cdot k$.

$= n \log n$. $\quad \boxed{\therefore C(n) = n \log_2 n}$

1|02|19

- Algorithm Merge(B[0,.....p-1], C[0....q-1], A[0....p+q-1])

// Merges two sorted array into one sorted array.

// Input: An array B[0....p-1] and C[0....q-1] both sorted.

// Output: Sorted array A[0.... p+q-1] of the elements of B and C.

```
i ← 0; j ← 0; k ← 0
while(i < p and j < q) do
    if B[i] ≤ C[j]
        A[k] ← B[i]; i ← i+1
    else
        A[k] ← C[j]; j ← j+1
    k ← k+1
if i = p
    copy C[j...q-1] to A[k...p+q-1]
else
    copy B[i.....p-1] to A[k.....p+q-1]
```

→ Analysis

Input ← n

Basic operation ← comparison

$C_{worst}(n) = 2 \cdot C(n/2) + (n-1)$ for $n > 1$ and $C(1) = 0$.

$a = 2, b = 2, f(n) = n-1$ Complexity less than n^2

$b^d = 2^1 = 2 \quad n^d = n^1$

i.e. $n^d \approx n \Rightarrow d = 1$.

$a = b^d$

i.e.

$C(n) \in \Theta(n^d \log n)$

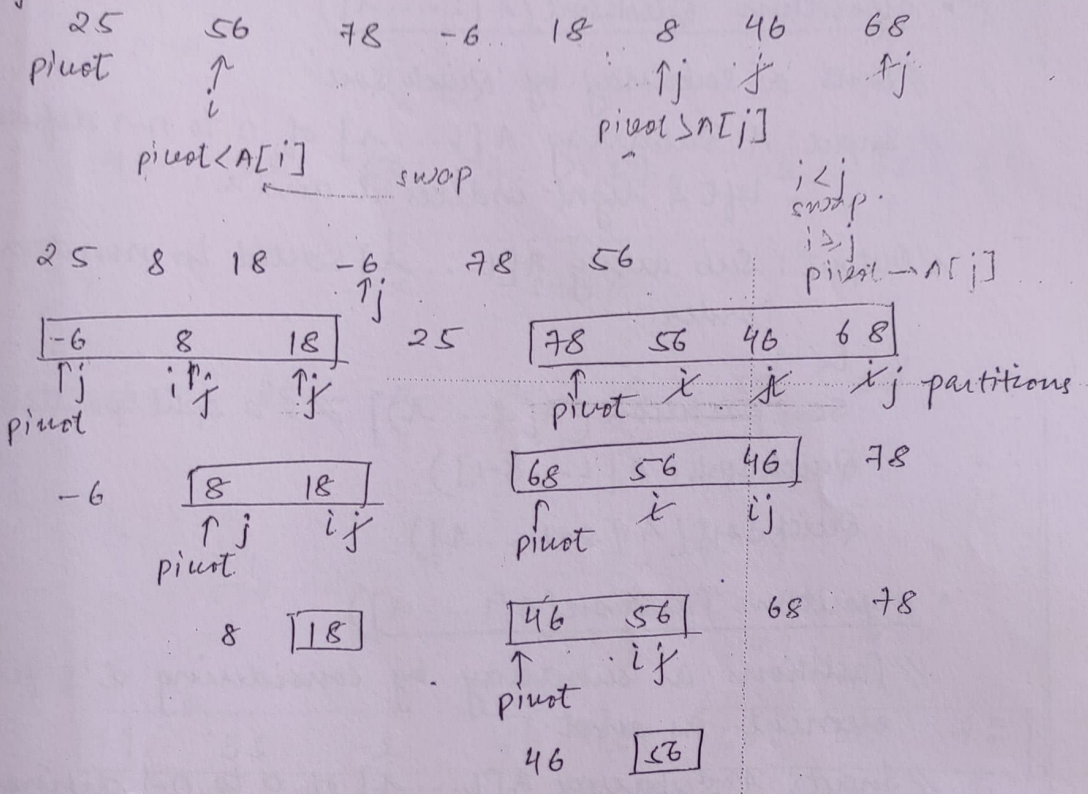
$\in (n \log n)$

$\therefore C(n) \in \Theta(n \log n)$

3/02/19

Quick Sort:

Eg: n=8



Sorted array: -6 8 18 25 46 56 68 78.

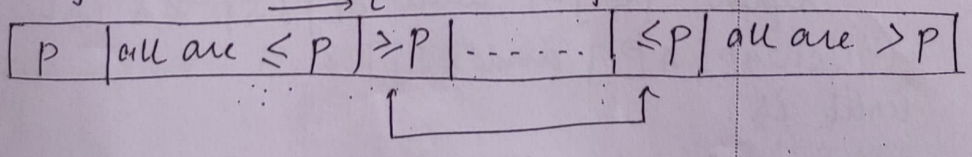
General form of Quick sort:

A[0] A[1] ..... A[n-2] A[n-1].

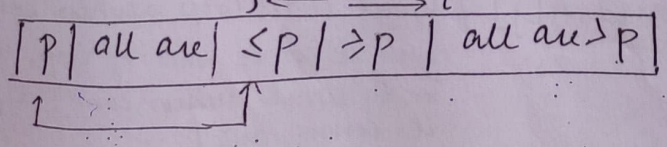
A[0] ..... A[s-1] | A[s] | A[s+1] ..... A[n-1]

< A[s] > A[s]

i) i<j

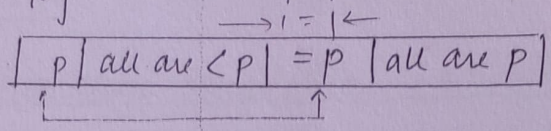


ii) i≥j



time for partitioning

iii) $i = j$



time for partitioning

- **Algorithm QuickSort(A[l....r])**

// Sorts a subarray by Quick sort.

// Input: A subarray A[l...r] of 0 to n-1 defined by left & right indices l and r.

// Output: Sub array A[l... r] sorted in nondecreasing order.

```
if l < r
    s ← partition(A[l...r)] // s is split position
    QuickSort(A[l...s-1])
    QuickSort(A[s+1....r])
```

- **Algorithm Partition(A[l...r])**

// Partitions a subarray by considering it's first element as pivot.

// Input: A subarray A[l...r] of 0 to n-1 define by its left and right indices l and r.

// Output: A partition A[l..r] with the split position n returned as this function's value.

```
p ← A[l]
i ← l; j ← r+1
repeat
    repeat i ← i+1 until A[i] ≥ p && i ≤ r
    repeat j ← j-1 until A[j] ≤ p && j ≥ L
    swap A[i] and A[j]
until i ≥ j
swap A[i] and A[j] //undo last swap when i ≥ j
swap A[l] and A[j]
return j
```

[repeat until loop → when cond° becomes true it comes out of loop, whereas in do while when cond° is true then it comes out].

04/02/19

n=8

0	1	2	3	4	5	6	7
45	72	56	7	19	-2	65	35

pivot i — swap

45 35 56 7 19 -2 65 72

A[i] ≥ p stops: 56 > 45 — swap

45 35 -2 7 19 56 65 72

56 > 45 — A[i] > p — i = 5, j = 4

i > j : 5 > 4

swap 19 undo 56

A[i] A[j]

19	35	-2	7	45	56	65	72

pivot — i = 0 — A[i] ≤ p : 7 < 19 — swap

19 7 -2 35 45 56 65 72

-2	7	19	35	65	72

pivot — A[i] ≥ p

-2 7 19 35 45 56 65 72

Sorted array: -2 7 19 35 45 56 65 72

→ Analysis

Input size ← n

Basic operation ← Comparison

$C_{best}(n) = 2\,C_{best}(n/2) + n$ for all $n \geq 1$

$C_{best}(1) = 0$.

WKT, $T(n) = aT(n/b) + f(n)$.

Here, $a=2$, $b=2$ $f(n)=n \in \theta(n^d)$

$d=1$.

$b^d = 2^1 = 2$

$C_{best}(n) \in \theta(n^d \log n)$

$C_{best}(n) \in \theta(n \log n)$

$C_{worst}(n) = (n+1) + n + (n-1) + \ldots\ldots\ldots + 3$.

$= (1+2+\ldots\ldots+(n-1)) - (1+2)$

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

$\approx \frac{n^2}{2} \in \theta(n^2)$.

$C_{avg}(n) = \frac{1}{n} \sum_{s=0}^{n-1} [(n+1) + C_{avg}(s) + C_{avg}(n-1-s)]$

(1st partition; pivot checking; $\forall\ n \geq 1$)

$C_{avg}(1) = 0$

$\therefore C_{avg}(n) = 2n \log_e n$

$\approx 1.38 n \log_2 n$

05/02/19

- Binary Search

Algorithm BinarySearch(A[0....n-1], k) [Iterative]

// Searches for a given key element k for an array A, non-recursively.

// Input: An array A[0....n-1] sorted in ascending order and a search key k.

// Output: An index of the element that is equal to k or -1.

```
l ← 0 ; h ← n-1
while (l ≤ h) do
    mid ← [(l+h)/2]
    if A[mid] = k  return mid
    if A[mid] > k  h ← mid-1
    else  l ← mid+1
return -1.
```

0	1	2	3	4	5	6	7	8	9
10	15	22	35	41	58	64	73	85	

l = 5
mid = 7.
A[7] = 64 ≠ k
l = mid + 1 = 8.
h = 9.

l. A[8] = k
return 8.

Algorithm BinarySearch(K, A, l, h) {Recursive}

```
if l > h   return -1
mid ← [(l+h)/2]
if A[mid] = k   return mid
if A[mid] > k
    return BinarySearch(K, A, l, mid-1)
else
    return BinarySearch(K, A, mid+1, h)
```

C(n) = C(n/2) + 1.

C(1) = 1

→ Analysis

Input size ← n

Basic operation ← comparison

Complexity : $C(n/2)$ because we check half part not $C(n/2)$.

$$C(n) = C(n/2) + 1$$

& $C(1) = 1$ ← bcoz we do 1 verification element.

⇒ $C(n) = C(n/2) + 1$ and $C(1) = 1$, if $n > 1$.

Put $n = 2^k$. [By backward subs°].

Sol°: Here, $a = 1$, $b = 2$, $f(n) = 1 \in \Theta(n^d)$

$d = 0$.

$b^d = 2^0 = 1$

$a = 1 = b^d$

$C(n) = \Theta(n^d \log_2 n)$

$\in \Theta(\log_2 n)$

$\therefore C(n) \in \Theta(\log_2 n)$

By Backward :

Put $n = 2^k$.

$C(2^k) = C(2^{k-1}) + 1$, Substitute $C(2^{k-1}) = C(2^{k-2}) + 1$

$= C(2^{k-2}) + 1 + 1$, Substitute $C(2^{k-2}) = C(2^{k-3}) + 1$

$= C(2^{k-3}) + 1 + 1 + 1$

$\vdots$

$= C(2^{k-i}) + i$

Put $k - i = 0$

$k = i$

$= C(2^{k-k}) + k$.

$= C(2^0) + k. = C(1) + k$

$C(2^k) = 1 + k$

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

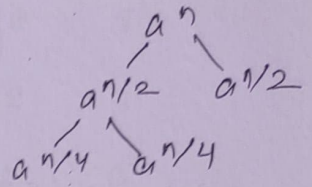
$\in \Theta(\log_2 n)$

DECREASE & CONQUER

1. Decrease by a constant value — Decrease by one
2. " —do— " constant factor — Divide and conquer
3. Variable size decreases

a^n

Bruteforce: $\underbrace{a \times a \times a \times \cdots \times a}_{n \text{ times}}$



→ Divide and conquer
$$\begin{cases} a^{n/2} \times a^{n/2} & \text{if } n > 1 \\ a & \text{if } n = 1 \end{cases}$$

→ Decrease by one
$$\begin{cases} a^{n-1} * a & \text{if } n > 1 \\ a & \text{if } n < 1 \end{cases}$$

→ Decrease by a constant factor
$$\begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even} \\ (a^{n-1/2})^2 \times a & \text{if } n \text{ is odd} \\ a & \text{if } n = 1 \end{cases}$$

Eg: $a^8 = (a^4)^2$

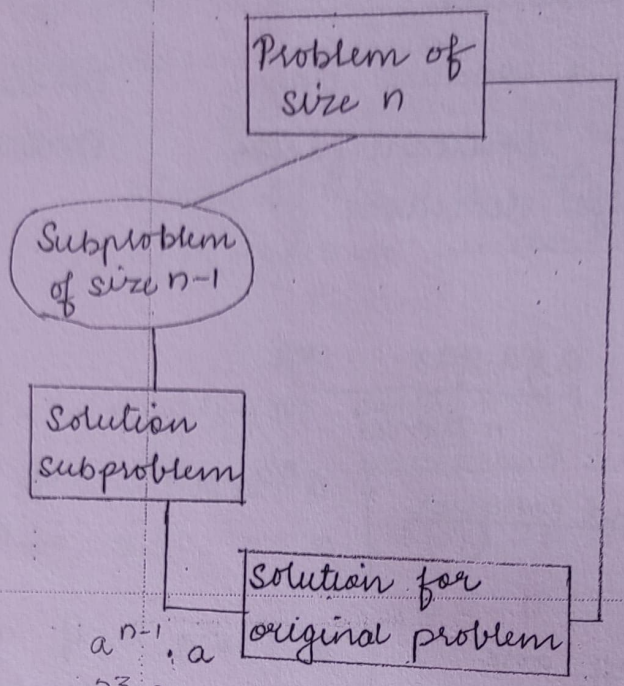
$a^9 = (a^4)^2 \cdot a$

→ variable size decrease:

Eg: gcd using Euclid's algorithm

$gcd(m, n) = gcd(n, m \bmod n)$

06/02/19



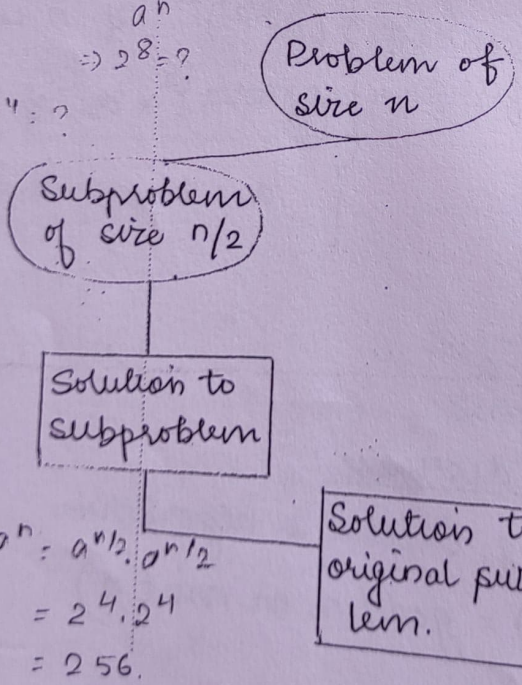
$n=1$
$a=2$

$a^{n-1} \cdot a$
$2^3 \cdot 2$
$= 2^4 = 16$

a^n
$\Rightarrow 2^8 = ?$

$2^{n/2} : 2^4 : ?$

$2^4 = 16$



$a^n = a^{n/2} \cdot a^{n/2}$
$= 2^4 \cdot 2^4$
$= 256.$

Decrease by one

Insertion Sort

n = 8 Passes = n-1

if (a[i] > a[i+1])

	56	45	-10	62	35	43	18	5
v = 45	45	56	-10	62	35	43	18	5
	-10	45	56	62	35	43	18	5
	-10	45	56	62	35	43	18	5
	-10	35	45	56	62	43	18	5
	-10	35	43	45	56	62	18	5
	-10	18	35	43	45	56	62	5
	-10	5	18	35	43	45	56	62

- Algorithm InsertionSort(A[0....n-1])

// Sorts a given array by Insertion Sort.

// Input: An array of A[0.....n-1] of 10 elements.

// Output: An array of A[0.....n-1] sorted in non decreasing order.

```
for i ← 1 to n-1 do
    v ← A[i]
    j ← i-1
    while j ≥ 0 and A[j] > v
        A[j+1] ← A[j]
        j ← j-1
    A[j+1] ← v
```

Tracing:

	0	1	2	3	4	5	6	7
	78	82	-5	23	16	35	28	8
i=1, v=82	78	82	-5	23	16	35	28	8
i=2, v=-5	-5	78	82	23	16	35	28	8
i=3, v=23	-5	23	78	82	16	35	28	8
i=4, v=16	-5	16	23	78	82	35	28	8
i=5, v=35	-5	16	23	35	78	82	28	8
i=6, v=28	-5	16	23	28	35	78	82	8
i=7, v=8	-5	8	16	23	28	35	78	82

→ Analysis:

Input size ← n

Basic operation ← comparison.

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n-1-1+1 = n-1 \in \Theta(n)$$

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} (i-1-0+1) = \sum_{i=1}^{n-1} i$$

$$= 1+2+\ldots n-1$$

$$= \frac{(n-1)(n)}{2} \approx \frac{n^2}{2} \in \Theta(n^2).$$

Average case Analysis:

$$C_{avg}(n) = \sum_{i=1}^{n-1} \left(\frac{i+1}{2}\right)$$

$$= \frac{1}{2} \sum_{i=1}^{n-1} (i+1) = \frac{1}{2}(2+3+4+\ldots\ldots+n).$$

$$\therefore C_{avg}(n) = \frac{1}{2}\left(\frac{n(n+1)}{2} - 1\right) = \frac{1}{2}\left(\frac{n^2}{2}\right) = \frac{n^2}{4}.$$

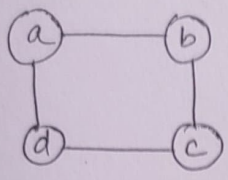
We use this when the order is almost sorted.

08/02/19

GRAPH

$G = (V, E)$

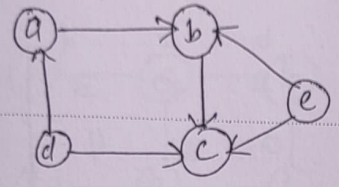
G_1:



$V = \{a, b, c, d\}$

$E = \{(a,b), (a,d), (b,c), (c,d)\}$

G_2:



$V = \{a, b, c, d, e\}$

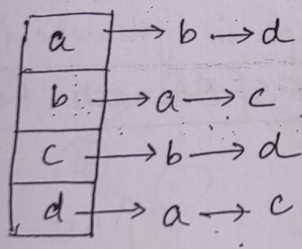
$E = \{(a,b), (b,c), (c,a), (d,a), (d,c), (c,b), (e,c)\}$.

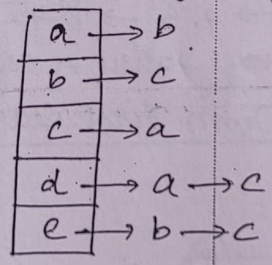
Adjacency matrix of G_1 & G_2:

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

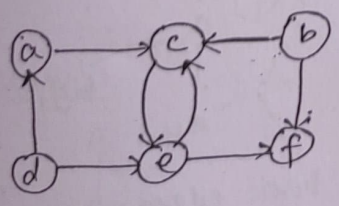
	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	0
c	1	0	0	0	0
d	1	0	1	0	0
e	0	1	1	0	0

Linked list representation of G_1 & G_2 (Adjacency list):





[G_1 matrix representation is symmetric on its principal diagonal & is undirected].



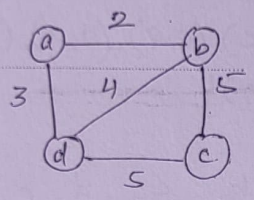
$V = \{a, b, c, d, e, f\}$

$E = \{(a,c), (b,c), (b,f), (c,e), (e,c), (e,f), (d,e), (d,a)\}$.

Adjacency matrix:

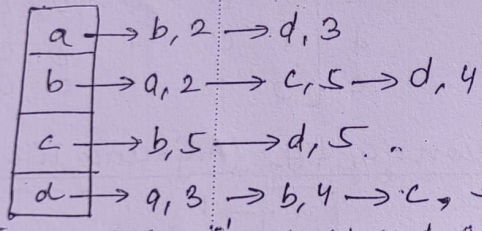
	a	b	c	d	e	f
a	0	0	1	0	0	0
b	0	0	1	0	0	1
c	0	0	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	0	0	1
f	0	0	0	0	0	0

⇒



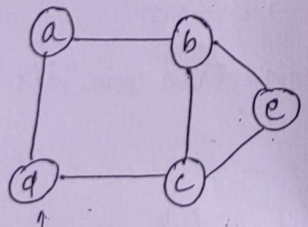
	a	b	c	d
a	0	2	∞	3
b	2	0	5	4
c	∞	5	0	5
d	3	4	5	0

linked list representation



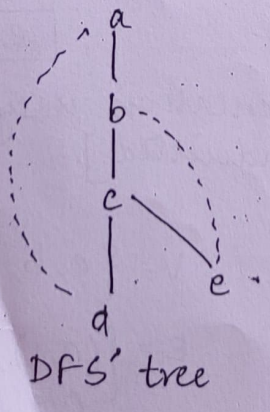
[To traverse vertices or edges of a graph in systematic fashion.

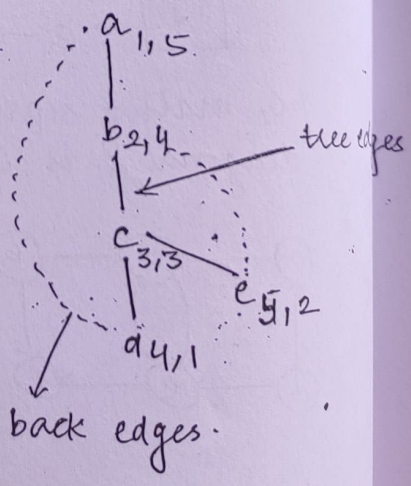
DFS [Depth First Search] & BFS [Breadth First Search]



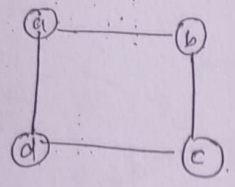
a 1,5
b 2,4
c 3,3
d 4,1
e 5,2

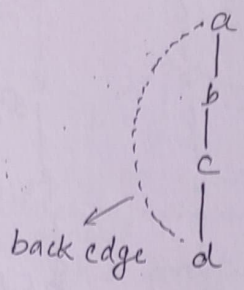
Stack.





[To solve DF's, stack data structure is used]

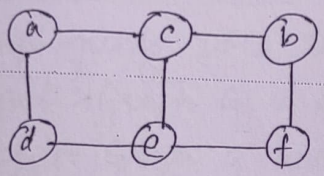




Push, pop
a 1,4
b 2,1
c 3,2
d 4,1

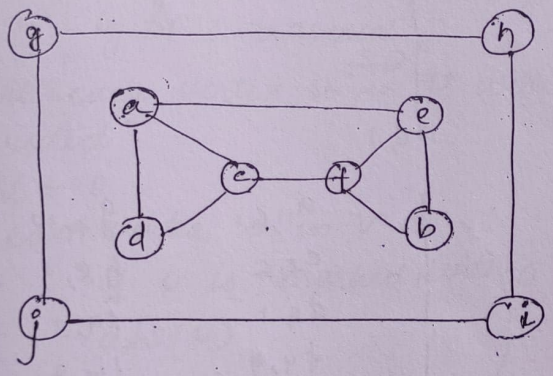
[Back edge signifies cycle]

1).

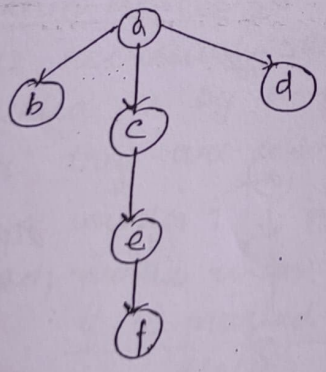


① Starting vertex 'a'
② Starting vertex 'e'.

2).

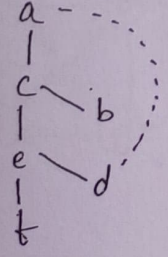


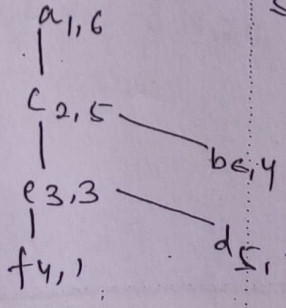
3).



Answers:

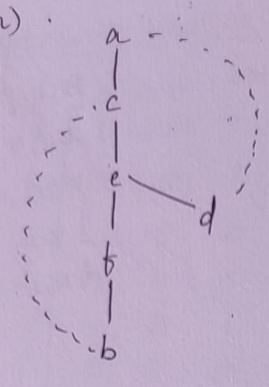
1. i).

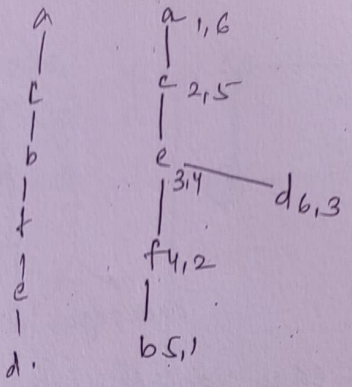




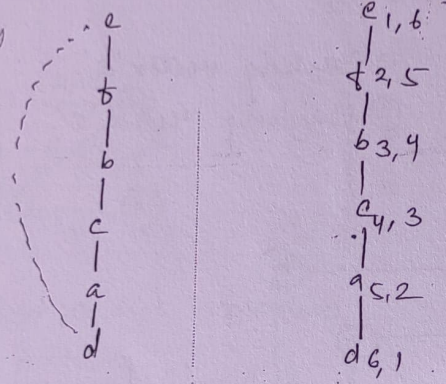
Starting vertex 'e'.

(or)



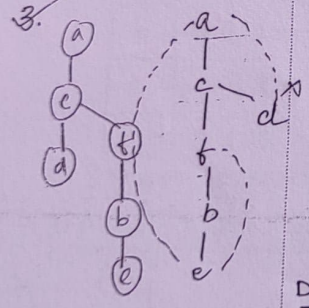


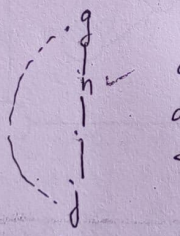
ii)



starting vertex 'e'

3)

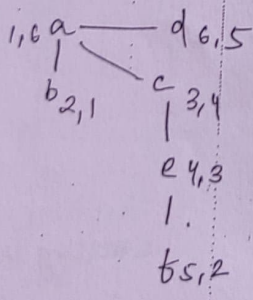


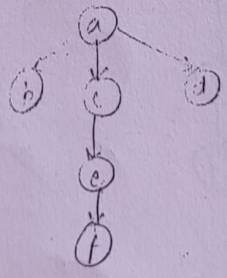


a 1,6
c 2,5
d 3,1
b 5,4
g 7,10
h 8,9
i 9,8
j 10,7

DF'S forest

4.





alphabetically low order

11/02/19

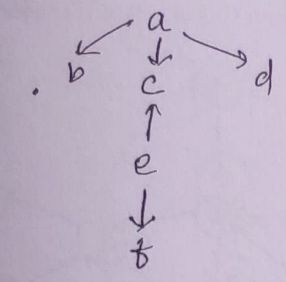
* Algori
// Impl
// Input
// Output

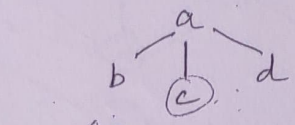
mark
univis
count
for ea

• Algori
// visit
to u
order
count
for e

Tracing:

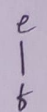
11/02/19





c is not an ancestor for e.
∴ back edge should not be drawn.
DF's forest.

a 1,4
b 2,1
c 3,2
d 4,3
e 5,6
f 6,5



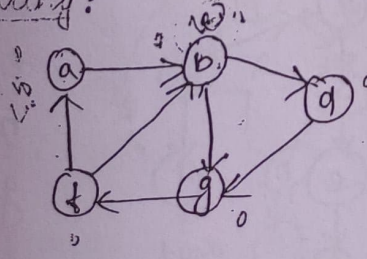
* Algorithm DFS(G)

// Implements DF's traversal of a given graph.
// Input: Graph G is equal to (V, E).
// Output: Graph G with its vertices marked with consecutive integers in the order they have been encountered by DF's traversal.

mark each vertex v in V with '0' as a mark of being "unvisited".
count ← 0
for each vertex 'v' in V do.
    if v is marked with 0
    dfs(v)

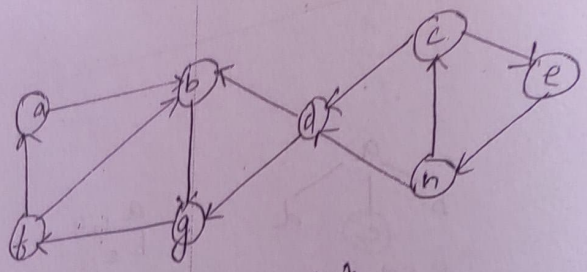
• Algorithm dfs(v)
// visits recursively all the unvisited vertices connected to vertex v by a path and number them in order they are encounter, via a global variable count.
count ← count + 1 ; mark v with count
for each vertex w in V adjacent to v do
    if w marked with 0
    dfs(w).

Tracing:



V = {a, b, d, f, g}
count ← 0, 1, 2, 3, 4, 5
T dfs(v) = a, b, g, f, d
v = 'a'
w = b.
dfs(w=b) v = 'b' → w = g.
w = f
dfs(w=f) v = 'f'

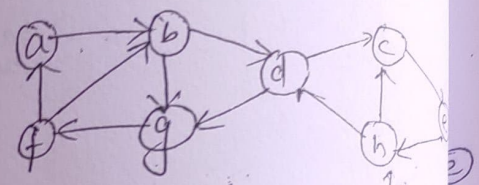
v = g
dfs(w=f) v = g
w = b
dfs(w=f) v = f
w = a.
w = b.



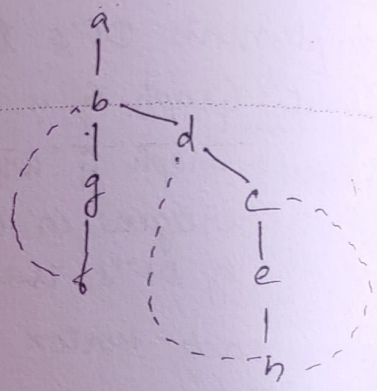
$V = \{a, b, c, d, e, f, g, h\}$.

dfs(v: a)
v: 'a' — w: b
count → 0,1
dfs(w='b') v=b
w: g
dfs(w='g') v=g
w: f
dfs(w='f') v=f
w: b
dfs(-1).
w: a
w: b
dfs(c)
v: c
w: d.
dfs(w='d') v=d.
(w=b
w=g)

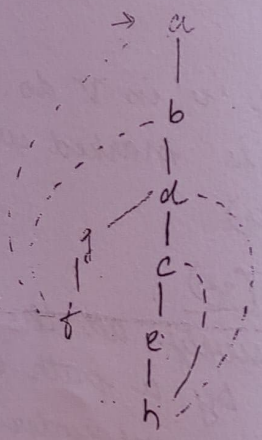
2)

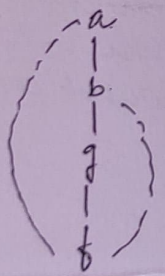


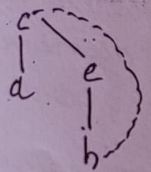
DFS' forest.



a 1, 8
b 2, 7
d 3, 6
c 4, 5
e 5, 2
h 6, 1
g 7, 5
f 8, 4.



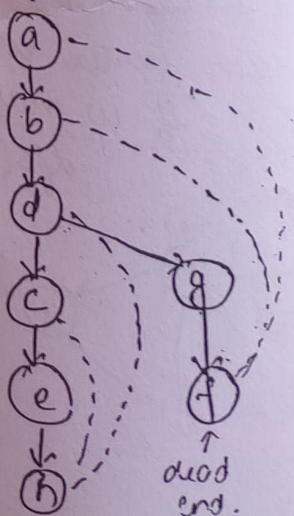




a 1, 4
b 2, 3
g 3, 2
f 4, 1
c 5, 8
d 6, 5
e 7, 7
h 8, c

whether they are predecessor or parent

Starting vertex 'a'.



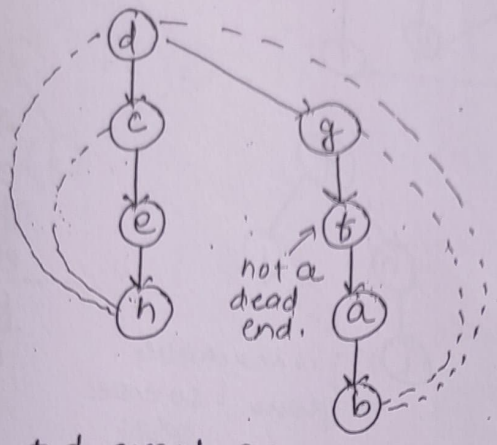
dead end.
predecessor

a 1, 8
b 2, 7
d 3, 6
c 4, 3
e 5, 2
h 6, 1
g 7, 5
f 8, 4

BFS

12|02|19

2) Starting vertex 'd'.



d 1, 8
c 2, 3
e 3, 2
h 4, 1
g 5, 7
f 6, 6
a 7, 5
b 8, 4

DFS' forest.

[Connected graph as we can reach all the vertices from 'd'].

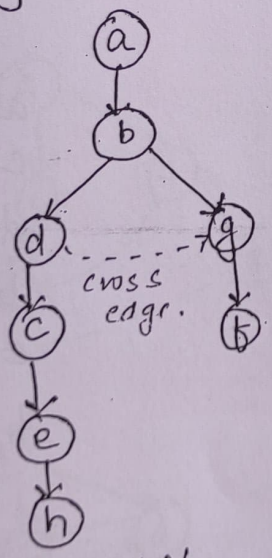
Here $|V| = 8$.

- Adjacency matrix: $\Theta(|V|^2)$
- Adjacency list: $\Theta(|E|+|V|)$

} both BFS & DFS.

BFS: [Question 2].

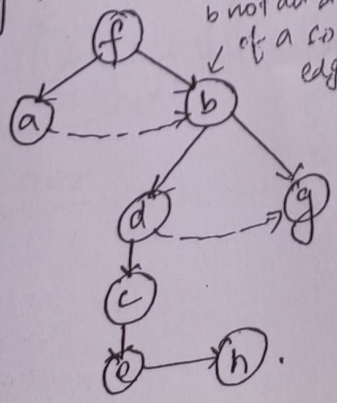
2) Starting vertex 'a'.



a 1, ① ← same order.
b 2
d 3
g 4
c 5
~~f 6~~ e 6
e 7 h 7
~~h 8~~ f 8

a	b	d	g	c	e		

Starting vertex 'f'.



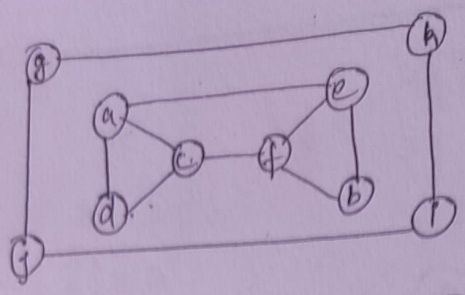
BFS trees.

f 1
a 2
b 3
d 4
g 5
c 6
e 7
h 8

If some nodes not visited then forest. All the vertices are visited from f. So its a tree.

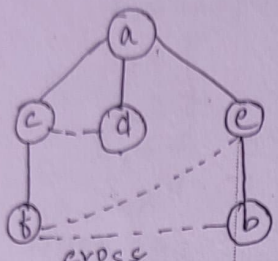
[Queue data structure is used for BFS]

Q.

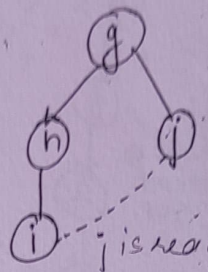


a 1
c 2
d 3
e 4
f 5
b 6

g 7
h 8
d 9
i 10



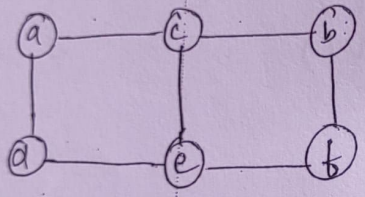
cross edge.



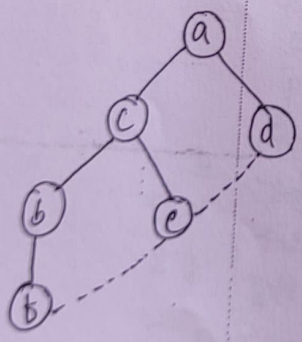
j is reachable from i so cross edge.

BFS forest.

Q1.

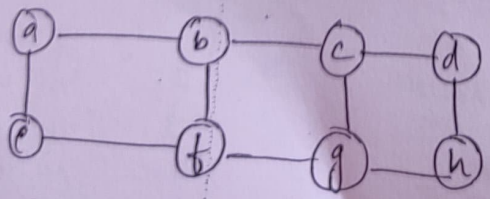


Starting vertex 'a'.

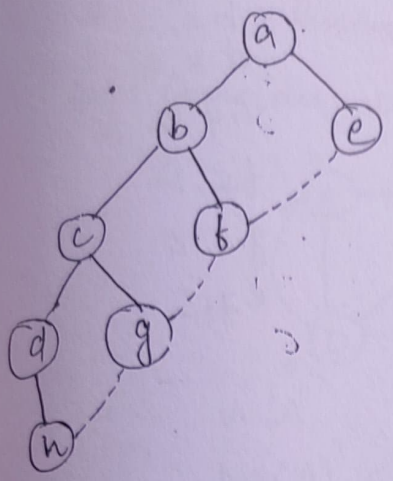


a 1
c 2
d 3
b 4
e 5
f 6

Q2.

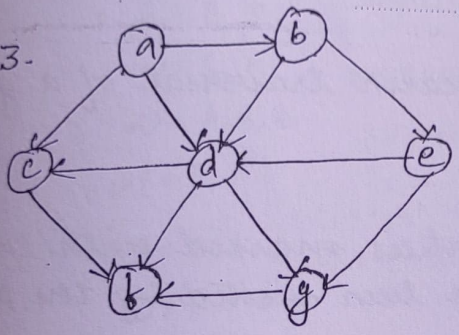


Starting vertex 'a'.

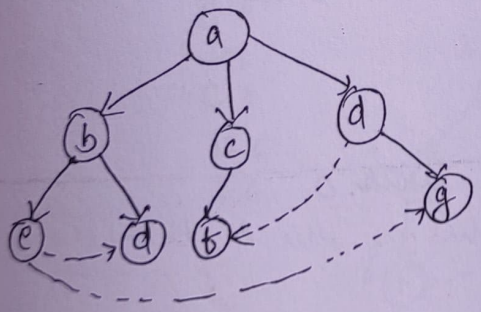


a 1
b 2
e 3
c 4
f 5
d 6
g 7

Q3.



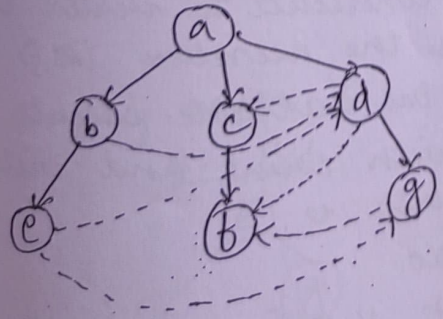
Starting vertex 'a'
Starting vertex 'd'.



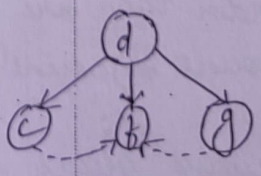
a 1
b 2
c 3
d 4
e 5
d 6
f 7
g 8

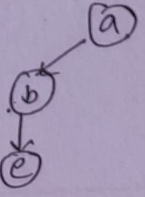
a 1
b 2
c 3
d 4.
e 5
f 6.
g 7

d 1
c 2
f 3
g 4
a 5
b 6
e 7



BFS Tree.



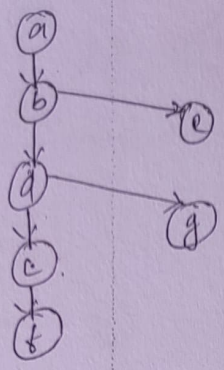


BFS forest.

13/02/19

DFS

Q 3.



a 1, 7
b 2, 3
d 3, 4
c 4, 2
f 5, 1
g 6, 2
e 7, 5

- Algorithm BFS(G)

// Implements breadth first search traversal of a given graph.

// Input: Graph G = (V, E)

// Output: Graph G with its vertices marked with consecutive order, they have been visited by the DFS traversal.

mark each vertex v or V with 0 as a mark of being "unvisited".

count ← 0

for each vertex v in V do

if v is marked with 0

bfs(v) ← comes here when # then for loop

- Algorithm bfs(v)

// visits all the unvisited vertex connected to vertex v by a path and assigns them the number in order they are visited by global variable count.

count ← count+1, mark v with count and initialize a queue with v.

while queue is not empty do.

for each vertex w in V adjacent to the front vertex do

if w is marked with 0

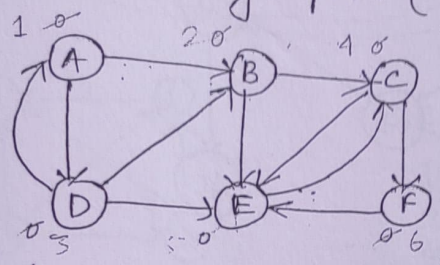
count ← count+1, mark w with count.

add w to the queue.

Remove front vertex from the queue.

Eg: Consider this graph: (Tracing)

G:



$V = \{A, B, C, D, E, F\}$

V = A.

bfs(A)

W = B, D — adjacent nodes of A

W = C, E — adjacent to B.

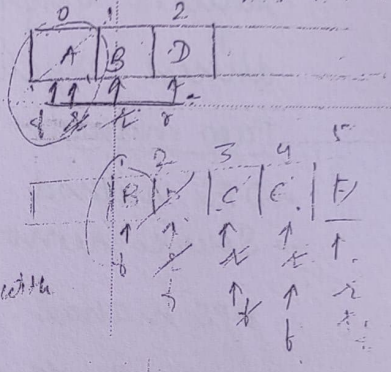
W = A, B, E — adjacent to D

W = E, F — adjacent to C.

W = C — adjacent to E not marked with 0, 5 ... remove.

W = E

count ← 0, 1, 2, 3, 4, 5, 6

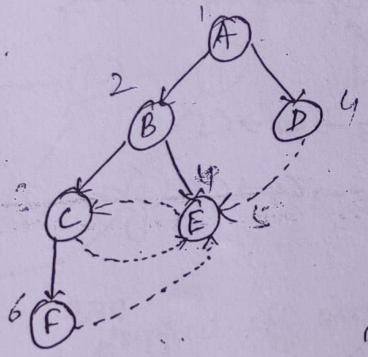


f = r = -1

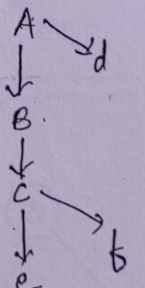
queue becomes empty

BFS tree

A 1
B 2
D 3
C 4
E 5
F 6



DFS tree



a 1, 6
b 2, 4
c 3, 3
e 4, 1
f 5, 2
d 6, 5

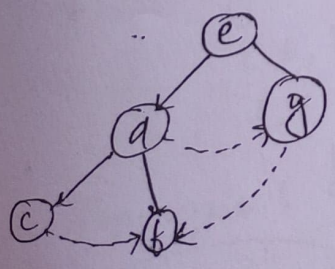
B not a dead end.

no back edge.

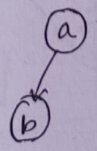
Q3:

Starting vertex 'e'.

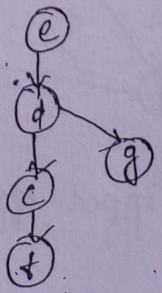
BFS tree



e 1
d 2
g 3
c 4
f 5
a 6
b 7



DFS:

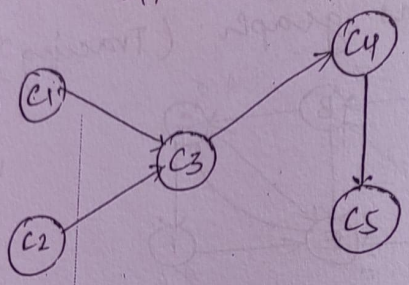


e 1, 5
d 2, 4
c 3, 2
f 4, 1
g 5, 3
a 6, 7
b 7, 6

14/02/18

• Topological Sorting (Ordering)

C_1, C_2, C_3, C_4



directed graph (digraph)
directed acyclic graph (DAG) dag.

Two methods:

→ DFS method
→ Source removal method.

DFS method:

popped off order: C_5, C_4, C_3, C_1, C_2

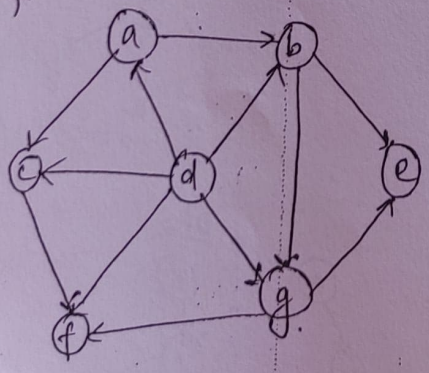
Queue C_2: $C_4 \rightarrow C_3 \rightarrow C_4 \rightarrow C_5$

$C_2 \rightarrow C_3 \rightarrow C_4 \rightarrow C_5$

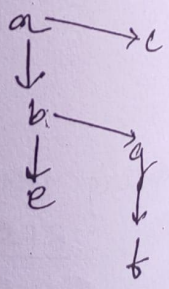
$C_4 \rightarrow C_3 \rightarrow C_4 \rightarrow C_5$

C_1,
$C_2, 5$
C_3,
$C_4, 2$
$C_5, 1$

G:

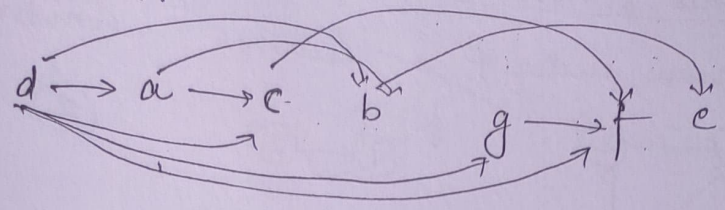


DFS
$a_{1,6}$.
$b_{2,4}$
$e_{3,1}$
$g_{4,3}$
$f_{5,2}$
$c_{6,5}$
$d_{7,7}$

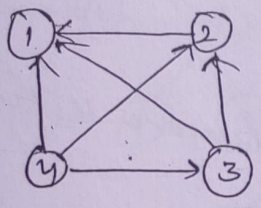


Popped off order: e, f, g, b, c, a, d

Topological order:



G:

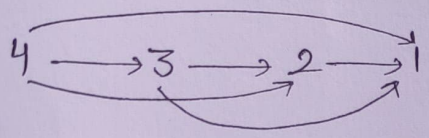


DFS.

1, 1
2, 2.
3, 3
4, 4

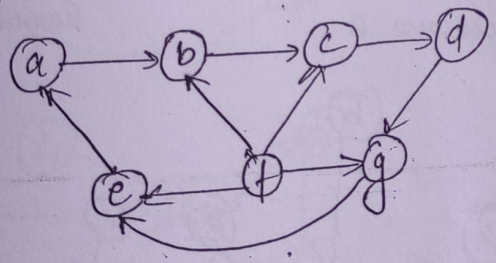
popped off order : 1 2 3 4.

Topological order:



4 → 3 → 2 → 1
4 → 2 → 1
4 → 3 → 1
4 → 1

G:



[Since cycle appears
No topological order].

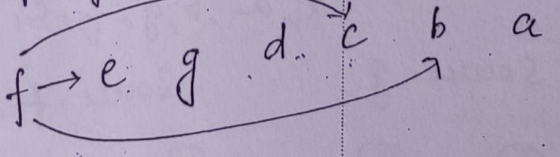
DFS

a, 1, 6
b 2, 5
c 3, 4
d 4, 3
g 5, 2
e, 6, 1
f, 7, 7.

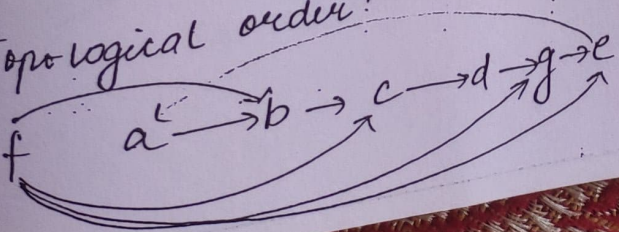
popped off order:

e g d c b a f

Topological order: ✗.

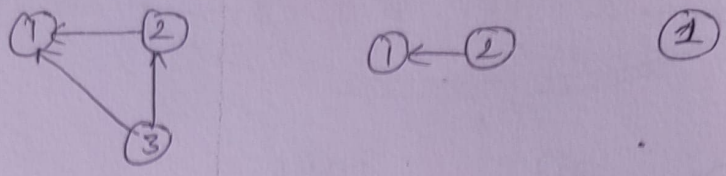


Topological order:



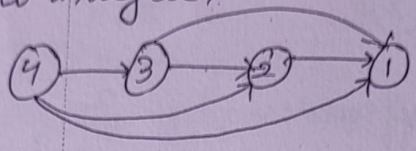
- **Source Removal method:**

Remove vertex 4 Remove 3 Remove 2 Remove 1



4, 3, 2, 1.

{ 4 has no indegree, hence remove 4 first. }.



Q: a → 1, 2 Indegree. Outdegree.

b → 1, 2

c → 2, 1

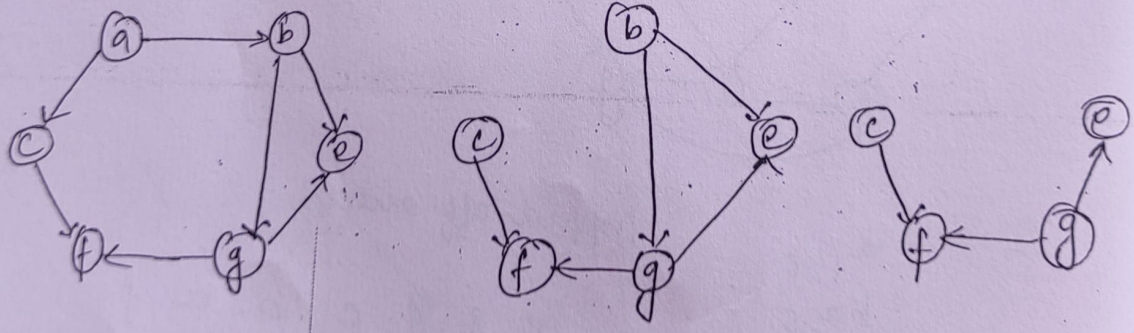
d → 0, 5

e → 2, 0.

f → 2, 0.

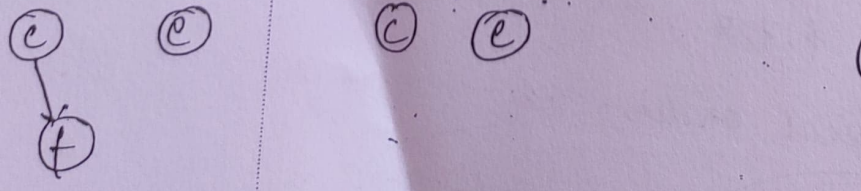
g → 2, 2.

Remove d. Remove a. Remove b.

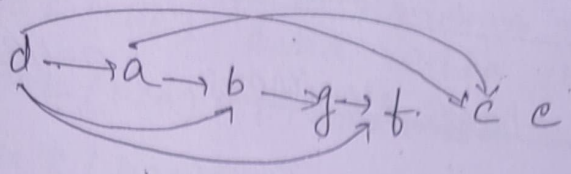


d, a, b, g, f, c, e.

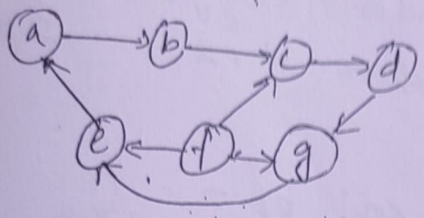
Source g Source f Source c Source e.



Topological order.

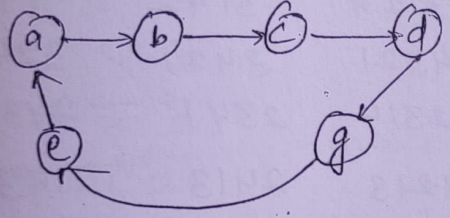


G:



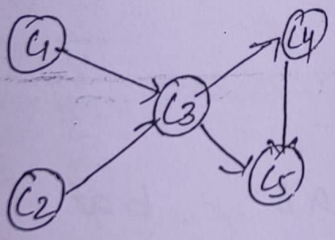
a → 1, 1
b → 1, 1
c → 1, 1
d → 1, 1
g → 1, 1
f → 0, 3.
e → 2, 1.

Remove f.

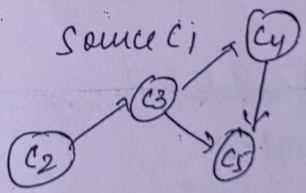


after this there is no vertex with indegree 0.
∴ no topological order exists.

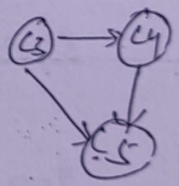
G:



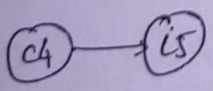
Source c_1



Source c_2



Source c_3

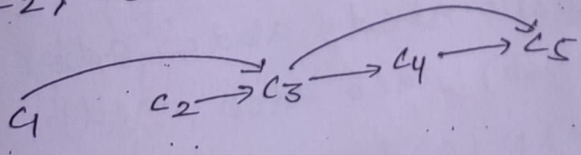


Source c_4

Source c_5.

proped order.

c_1, c_2, c_3, c_4, c_5.



15/02/19

Generating combinatorial objects -

Minimal change algorithm:

n = 1, 2, 3.

Initial : 1

Insert 2 to previous solution : 12 (RL) 21 (LR)
from right to left

Insert 3 to the previous sol' : 123 (RL) 132 (LR) 312 (RL)
from right to left (12) : 213 (RL)
from left to right (21) : 321 (LR) 231 (RL)

Insert 4 to the previous sol' : 1234 1243 1423
from right to left (123) 4123.
from left to right (132) : 4132 1432 1342 132
from right to left (312) : 3124 3142 3412 431
from left to right (321) : 4321 3421 3241 321
from right to left (231) : 2314 2341 2431 423
from left to right (213) : 4213 2413 2143 2134

Elements: a b c d

Initial: a :

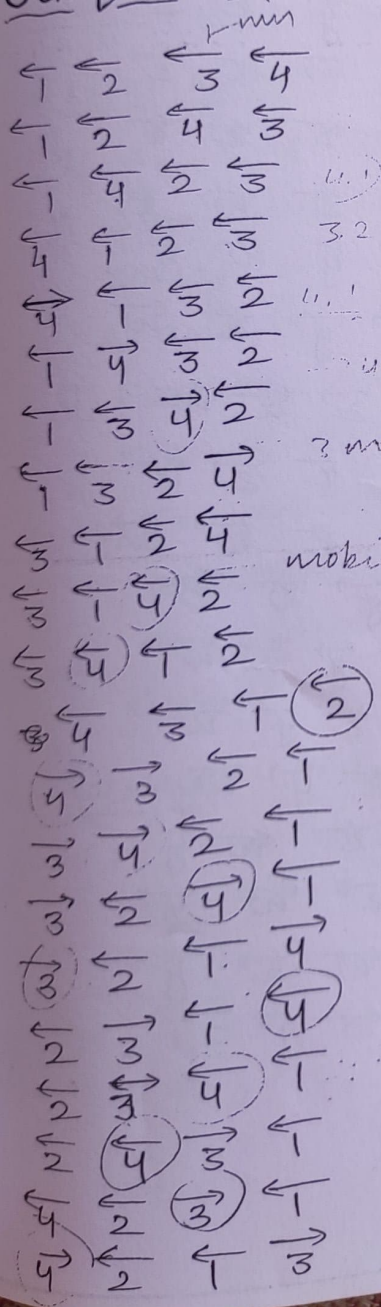
Insert b to previous sol' : a b b a
from right to left

Insert c to previous sol'
from right to left (ab) : a b c a c b c a b
from left to right (ba) : c b a b c a b a c

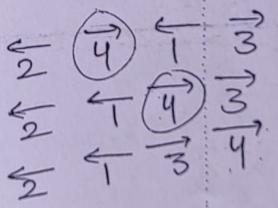
Insert d in previous
from R to L (abc) : a b c d a b d c a d b c d a b c
from L to R (in acb) : d a c b a d c b a c d b a c b d
from R to L (cab) : c a b d c a d b c d a b d c a b

from L to R (cba) : dcba cdba cbda cbad
from R to L (bca) : bcad bcda bdca dbca.
from L to R (bac) : dbac bdac badc bacd.

Johnson - Trotter Algorithm :



mobile element
4, 3, 2
4, 2.
4, 3.
3
3, 4



$a < b < d$

$w < x < y < z$.

$z > y > x$

mobile

17/02/19

Algorithm Johnson Trotter(n).
// Implements Johnson-Trotter algorithm for generating permutation.
// Input: A positive integer n
// Output: A list of all permutation of {1, ..., n}
Initialize the first permutation with $\overleftarrow{1}\ \overleftarrow{2} \ldots \overleftarrow{n}$
while there is a mobile integer k do.
    find the largest mobile integer k
    swap k and the adjacent integer it's arrow points to
    reverse the direction of all integers that all larger than k.

- **Lexicographic Ordering :**

If $a_{n-1} < a_n$ swap a_{n-1} and a_n, otherwise,

i) Scan current permutation from right to left looking for 1st pair of consecutive element, a_i and a_{i+1} such that $a_i < a_{i+1}$ $\forall$ $i < n$.

ii) Find the smallest element in the pair larger than a_i and put it in position i.

iii) The positions from $i+1$, through n are filled with the elements $a_i, a_{i+1} \ldots a_n$ from which the element written in ith position has been eliminated, in the increasing order.

Example: 123.

123 — 2<3
132 ✓
312
321 ✗
213 ✓ — 1<3.
231 ✓
312 ✓ — 1<2
321 ✓

1 2 3 4. 4>3. 3<4.

1 2 4 3.

1 3 2 4. 2<4.

1 3 4 2

1 4 2 3. 2<3.

1 4 3 2

1<4 we need to find ... largest than 1 i.e. 2.

2 1 3 4 3<4.

2 1 4 3

2 3 1 4 1<4

2 3 4 1

2 4 1 3 3<1

2 4 3 1 ——→ 4<3

3 1 2 4 ✓ 2<4

3 1 4 2 ✓

~~3 2 4 1 ✓ 1<4.~~

~~3 2 4 1~~ ~~3 4 2 1 ✓ 1<2~~

~~3 4 2 1~~ ~~3~~

3 2 1 4

3 2 4 1 [just larger ...]

3 4 2 1 2 1 3 4 2 4>3

3 4 2 1 1 4 2 3. ymmm... order.

4 1 2 3

4 1 3 2 1<3 ✓ swap by just large.

4 2 1 3

4 2 3 1 2<3 ✓ true.

4 3 1 2

4 3 2 1

• 4 6 2 5 3 1.

4 6 2 5 3 1

~~4 6 5 1 2 3.~~

~~4 6 5 1 3 2~~

4 6 3 1 2 5

4 6 3 1 5 2.

3 < 1.

5 P 2 → 120.

• a < b < c < d.

a b c d.

a b d c.

c < d.

a c b d.

a c d b.

~~a c b d.~~

a d c b

a d b c.

b a c d

b a d c

b c a d

b c d a

b d a c

b d c a.

c a b d

c a d c

c b a d

c b d a

c d a b

c d b a

d a b c

d a c b

d b a c

d b c a

d c a b

d c b a.

w < x < y < z.

w x y z

w x z y

w y x z

w y z x.

w z x y

w z y x

x w y z

x w z y

x y w z

x y z w

x z w y

x z y w

y w x z

y w z x

y x w z

y x z w

y z w x

y z x w

z w x y

z w y x

z x w y

z x y w

z y w x

z y x w.

• Generating Subset:

$P = \{a_1, a_2, a_3\}$

n	
0	ϕ
1	ϕ $\{a_1\}$
2	ϕ $\{a_1\}$ $\{a_2\}$ $\{a_1, a_2\}$
3	ϕ $\{a_1\}$ $\{a_2\}$ $\{a_1, a_2\}$ $\{a_3\}$ $\{a_1, a_3\}$ $\{a_2, a_3\}$ $\{a_1, a_2, a_3\}$
4	ϕ $\{a_1\}$ $\{a_2\}$ $\{a_1, a_2\}$ $\{a_3\}$ $\{a_1, a_3\}$ $\{a_2, a_3\}$ $\{a_1, a_2, a_3\}$ $\{a_4\}$ $\{a_1, a_4\}$ $\{a_2, a_4\}$ $\{a_3, a_4\}$ $\{a_1, a_2, a_4\}$ $\{a_1, a_3, a_4\}$ $\{a_2, a_3, a_4\}$ $\{a_1, a_2, a_3, a_4\}$

Binary Reflected grey code-

minimal change algorithm

000 001 011 010 110 100 101 111