14/05/2023

Asymptotic Notation and Basic efficiency classes

1) The efficiency analysis framework concentrates on the orders of the growth of an algorithm. basic operations count as the principle indication of the Algorithm efficiency

- O (Big oh)
- Ω (Big omega)
- Θ (big theta)

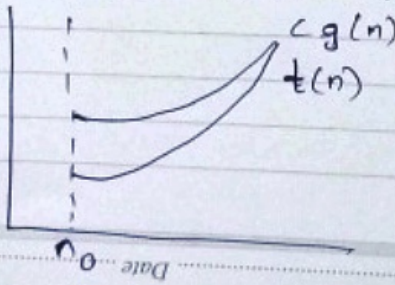
consider $t(n)$ & $g(n)$, can be any non-negative func^n defined on the set of natural numbers. where $t(n)$, an algorithm running time and $g(n)$ will be some simple function to compare the count of basic operations.

ex:- $n \in O(n^2)$
$100n + 5 \in O(n^2)$
$\frac{1}{2} n(n-1) \in O(n^2)$

$g(n)$ = Threshold value
$t(n)$ = no. of times algo runs

Big oh (O) Notations:-

Defn1:- A func^n $t(n)$ is said to be in (big oh) $O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ $\forall$ large n. i,e. if there exist some positive constant C and some non negative integer n_0 such that $t(n) \leq C\,g(n)$ $\forall$ $n \geq n_0$



Times of Experiment : ..... Date ....
Experiment No. ..... Experiment Result : .....
Page No.

Theorem from asymptotic notation write an ex each. 10m

As an example let us formally prove one of the assertion made above.

consider $100n+5 \in O(n^2)$

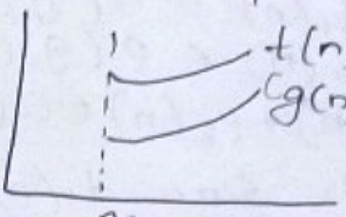
$100n+5 \leq 100n+n \quad \forall\ n \geq 5$

$101n \leq 101n^2$

Thus as values of the constants $C=100$ $n_0=5$

* Ω (omepa big)

Defn 2:- A func^n $t(n)$ is said to be in $\Omega(g(n))$, denoted by $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n. i,e if these exist a some +ve constant C and some non-negative integers n_0 such that $t(n) \geq Cg(n)$



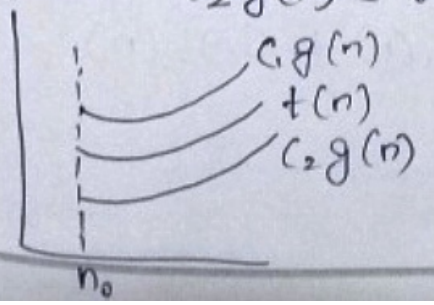
ex:- $n^3 \in \Omega(n^2)$

$n^3 \geq n^2 \quad \forall\ n \geq 0$

That is the constants $C=1$ $n_0=0$.

* Big theta (Θ) Notation.

Defn 3:- A func^n $t(n)$ is said to be in $\Theta(g(n))$, denoted by $t(n) \in \Theta(g(n))$. if $t(n)$ is bounded both Above and Below by some +ve constant multiple of $g(n)$ for all large n, i.e if there exist some +ve constant C_1 and C_2 and non-ve integers n_0 such that $C_2 g(n) \leq t(n) \leq C_1 g(n) \quad \forall\ n \geq n_0$



eg:- let us prove that $\frac{1}{2}n(n-1) \in \Theta(n^2)$

first we will prove the right inequality.

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

$$\frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad \forall n \geq 0$$

~~text~~ now $\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n \cdot \frac{1}{2}n$$

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{4}n^2 \quad \forall n \geq 2.$$

hence we can select constants $C_2 = 1/4$ $C_1 = 1/2$ $n_0 = 2$.

6m or 8m Theorem:-

if $t_1(n) \in O(g_1(n))$.
$t_2(n) \in O(g_2(n))$ then
$t_1(n) + t_2(n) \in O(max\{g_1(n), g_2(n)\})$

proof:- Since $t_1(n) \in O(g_1(n))$ there exist some +ve constant c_1 and some non-ve integer n_1

$$t_1(n) \leq c_1(g_1(n)) \quad \forall n \geq n_1 \rightarrow (1)$$

since $t_2(n) \in O(g_2(n))$

$$t_2(n) \leq c_2 g_2(n) \quad \forall n \geq n_2 \rightarrow (2)$$

let us denote $c_3 = max[c_1, c_2]$
and consider $n \geq max[n_1, n_2]$
so that we can use both inequalities that is
Adding (1) and (2) equation

$$t_1(n) + t_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$$

replace c_1 & c_2 with c_3

Time of Experiment : ..... Date .....
Experiment No. ..... Experiment Result : ..... Page No.

$t_1(n) + t_2(n) \leq c_3 g_1(n) + c_3 g_2(n)$

$\leq c_3 [g_1(n) + g_2(n)]$

$\leq c_3 2 \max\{g_1(n), g_2(n)\}$

Hence $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Mathematical Analysis of non-recursive Algorith

Algorithm Max element (A[0 ---- n-1])

// Determines the value of largest element in array

// I/P :- An array A[0 ---- n-1] of real number

// O/P :- The value of the largest element in A

```
max ← A[0]
for i ← 1 to n-1 do
    if A[i] ≥ max
        max ← A[i]
return max
```

Let us denote C(n) the no of times the basic operation will execute. In the above example the basic operation are likely two statements one is if A[i] ≥ max and another one max ← A[i] The second instruction will not execute all the time but the 1st instruction that is condition statement will execute for all the iterations of loop ∴ The basic function is first stat i.e if A[i] ≥ max.

formula (upper limit − lower limit + 1)

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

$= n-1-1+1$

$\therefore n-1 \in \Theta(n)$

formulas:-

1) $\sum_{i=l}^{u} (a_i = c \sum_{i=l}^{u} a_i$

2) $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$

3) $\sum_{i=l}^{u} 1 = u-l+1$

4) $\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1+2+\cdots+n = \frac{n(n+1)}{2} \approx \frac{1}{2} n^2 \in O(n^2)$

10M

Write a Algorithm to check whether the array is having unique elements ~~of~~ or not. ~~then check the~~ then derive the efficiency of Algo by mathematical Analysis.

ALGORITHM Unique elements(A[0.....n-1])
// Determines whether all the elements in a given
// array are distinct.
// I/P:- An array A[0....n-1])
// O/P:- Returns true if all the elements in A are
// distinct & false otherwise.
for i ← 0 to n-2 do
for j ← i+1 to n-1 do

Times of Experiment :
Experiment No. Experiment Result : Date
Page No.

10M

Mathem

The

will be

method

condition

eg:- Find

ALGORIT

// Compute

// I/P: A

// O/P: Th

if A[i] = A[j]
return false.
return True.

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$\sum_{i=0}^{n-2} n-1-(i+1)+1 \quad , \quad \sum_{i=0}^{n-2} = n-1+i-1+1$$

$$\sum_{i=0}^{n-2} n-1+i \quad , \quad \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$(n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

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

$$\approx \frac{1}{2} n^2 \in O(n^2)$$

ie motical Analysis of Recursive Algorithm.

" " " " "

be always derived by Backward substitu
od until you will Reach the stopping
tional statement
inding the factorial of a given number.
)RITHM F(n)
ute n! recursively
A non-ve integer n
The value of n!

if $n \leq 0$ return 1
else return $f(n-1) * n$

$M(n) = M(n-1) + 1$
$= [M(n-2) + 1] + 1$
$= [[M(n-3) + 1] + 2$
⋮ ⋮ ⋮
$= M(n-i) + i$
⋮ ⋮ ⋮
$= M(n-n) + n.$
$= M(0) + n \approx n$

Brute force ~~Algorithm~~ Method. comparison of one by one no like
Divide & Conquer Me "
Decrease & " "
Transform & " "
Dynamic Programming.
Greedy method.
Space and time trade off.

Brute force Methods:-
Bubble sort, selection sort.
Sequential pattern search.

ALGORITHM
// Sorts a given array by bubble sort
// I/p: An array ~~A(0...n-1)~~ A[0...n-1] of n orderable elements.

Times of Experiment : ..... Page No. .....
Experiment No. ..... Experiment Result : ..... Date .....

// O/p: Array A[0...n-1] sorted in Ascending order.
for i ← 0 to n-2 do
  for j ← 0 to n-2-i do
    if A[j+1] < A[j]
      swap (A[j], A[j+1])

0	8	1	1	1	1	1	
1	1	8	4	4	4	4	
2	4	4	8	2	2	2	
3	2	2	2	8	8	8	
4	9	9	9	9	9	3	
5	3	3	3	3	3	9	poss 1

0	~~Array~~ 8	1	1	1	1	1	
1	4	4	4	2	2	2	
2	2	2	2	4	4	3	
3	8	8	8	8	8	4	
4	3	3	3	3	3	8	pos 2
5	9	9	9	9	9	9	

0	1	1	1	~~1~~	~~1~~	~~1~~	
1	2	2	2	~~2~~	~~2~~	~~2~~	
2	4	4	4	~~3~~	~~3~~	~~3~~	
3	3	3	3	~~4~~	~~4~~	~~4~~	posi 3
4	8	8	8	~~8~~	~~8~~	~~8~~	
5	9	9	9	~~9~~	~~9~~	~~9~~	

pos: 5

0	1	1
1	2	2
2	3	3
3	4	4
4	5	5
5	6	6

$C_{worst} = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1$

$= \sum_{i=0}^{n-2} (n-2-i) - 0 + 1$

$= \sum_{i=0}^{n-2-i} (n-1-i)$

$(n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$ (calculate) $+$

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

BMSITM IT until first letter matches there is no need to check the second letter.

IT

II

III

IT STOPS.

ALGORITHM Sequential Matching (T[0...n-1], P[0...m-1])
// Implements Brute force string matching
// I/P:- An array T[0...n-1] of n characters representing Text.
// An array P[0...m-1] of m characters representing pattern. $m \le n$.
// O/P:- Pattern matches in Text or not.

```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j] do        ← Basic operation
        j ← j+1
    if j = m return i
return -1
```

Time of Experiment : ..... Experiment No. ..... Experiment Result ..... Date .....
Page No.

$n-m=5$ $n=7$ $m=2$

BMSITM

i	n-m	n	m	j	
0	5	7	2	0	I ≠ B
1				0	I ≠ M
2				0	0<2 T ≠ S
3				0	0<2 I = I
				1	1<2 T = T
				2 = 2	

i = 3 True.

IT G

i	n-m	n	m	j	
0	6	9	3	0	0≤6 0<3 I ≠ A
1				0	1≤6 0<3 I ≠ L
2				0	2≤6 0<3 I ≠ G
3				0	3≤6 0<3 I ≠ O
4				0	4≤6 0<3 I ≠ R
5				0	5≤6 0<3 I = I
				1	1<3 T = T
				2	2<3 G ≠ H
6				0	6≤6 2 ≠ 3
				0<3	I ≠ T
				0 ≠ 3	7≤6 false.

return -1 Pattern not found.

ALGORITHM

I T G false.
I T G false.
I T G false.
I T G false
I T G false
I T G True
I T G True
I T G false
I T G. false

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

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

$= (m+1)(n-m-0+1)$

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

n^2 author

Module-5 15/05/2023

Quick Sort.
comparison to Bubble Sort it is better

ALGORITHM QuickSort(A[l....r])
// Sorts a subarray by quicksort
// I/P:- A subarray A[l....r] of A[0....n-1]
// O/P:- Subarray A[l....r] sorted in
ascending order
if l < r
  s ← Partition(A[l....r]) // Basic operation
  Quicksort(A[l.....s-1])
  Quicksort(A[s+1.....r]

ALGORITHM partition(A[l....r])
// Partition a subarray
// I/P A subarray A[l....r] of A[0....n-1]
where l ≤ r.
// O/P A Partition of A[l....r] with
split S. partition.

P = Pivot

p ← A[l]
i ← l ; j ← r+1
repeat
  repeat i ← i+1 until A[i] ≥ p
  repeat j ← j-1 until A[j] ≤ p
  swap(A[i], A[j])
until i ≥ j
swap(A[i], A[j])
swap(A[l], A[j])
return j

If it false it will repeat. True means come out repeat until.

Time of Experiment : ..... Date .....
Experiment No. ..... Experiment Result : ..... Page No

	l	0	i	1	2	3	4	5	6	r	j+1	
example:	[5]	3	1	9	8	2	4	7				3 ≥ 5 F
	5	3	1	9	8	2	4	7				1 ≥ 5 F
	5	3	1	9	8	2	4	7				9 ≥ 5 True.
	5	3	1	9	8	2	4	7				7 ≤ 5 False
	5	3	1	9	8	2	4	7				4 ≤ 5 True
	5	3	1	4	8	2	9	7				swap.
	5	3	1	4	8	2	9	7				3 ≥ 6 false.
	5	3	1	4	8	2	9	7				8 ≥ 5 True.
	5	3	1	4	8	2	9	7				2 ≤ 5 True.
	5	3	1	4	2	8	9	7				4 ≥ 5 False. repeat
	5	3	1	4	2	8	9	7				8 ≥ 5 True.
	5	3	1	4	2	8	9	7				2 ≤ 5 True.
	5	3	1	4	8	2	9	7				6 ≥ 5 True.
	5	3	1	4	2	8	9	7				i , j
	[2	3	1	4]	[5]	[8	9	7]				l , j
	l			s-1		s+1		r				5 = 5

$C_{best}(n) = 2\,C_{best}(n/2) + n \quad \forall\, n > 1$

$C_{best}(1) = 0$

$C_{best} = n \log_2 n$ placement

Quicksort

$C_{worst}(n) = (n+1) + n + \dots + 3$

$= \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2)$ placement

Binary Search.

ALGORITHM BinarySearch(A[0.....n-1])
// implements non-recursive Binary search
// I/P:- An array of A[0....n-1] sorted in
Binary search and a search key K.
// O/P:- An ~~our~~ index of array elements that
// is equal to K or -1 if there is no such
// element

```
l ← 0; r ← n-1
while l ≤ r do
    m ← [(l+r)/2]
    if K = A[m] return m
    else if K < A[m]
        r ← m-1
    else
        l ← m+1
return -1
```

example:

	0	1	2	3	4
	1	2	[3]	[4]	5
	l		m		lM

K=4

l	r	m	K
0	4	2	4
3	4	3	

$C_{best}(n) \in O(1)$ $C_{avg}(n) \in O(\log_2 n)$ $C_{worst}(n) \in O(\log_2 n+2)$

inf

least order of the growth.

placement

efficient algorithm.

Merge Sort

Dividing array into single element.

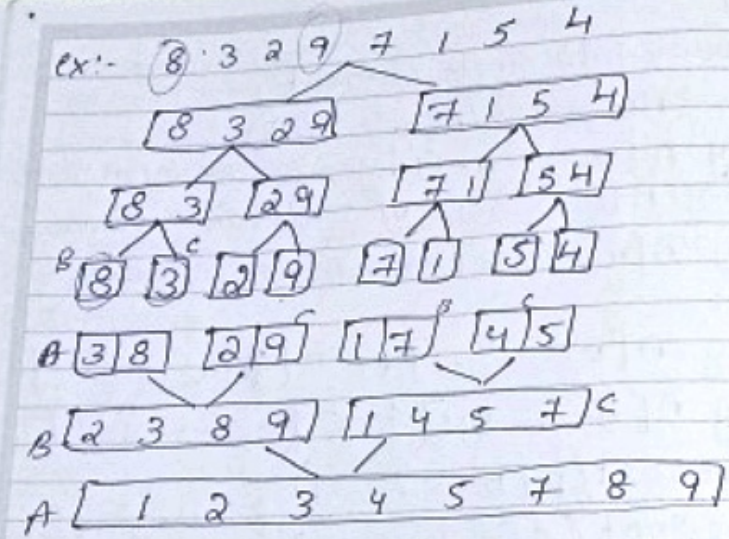
ALGORITHM MergeSort(A[0....n-1])
// sorts array A[0.....n-1] by recursive merge sort
// I/P: An Array A[0...n-1] of orderable numbers
// O/P: Array A[0...n-1] in ascending order

```
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)
```

ALGORITHM Merge(B[0...p-1], C[0...q-1],
A[0.....p+q-1]) sorted
// Merges two sorted array into one array
// I/P: Arrays B[0...p-1] & C[0...q-1] both sorted
// O/P: Sorted array A[0...p+q-1] both B & 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]
```

ex:- 8 3 2 9 7 1 5 4



Cworst(n) ∈ O(n log n)

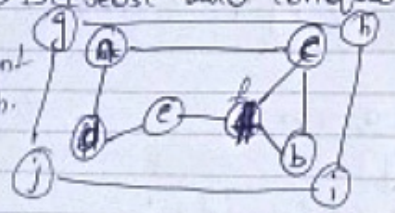
other d storage B & C are used. extra space space efficiency is not there hence we will use ~~too~~ quick sort

17/05/2023

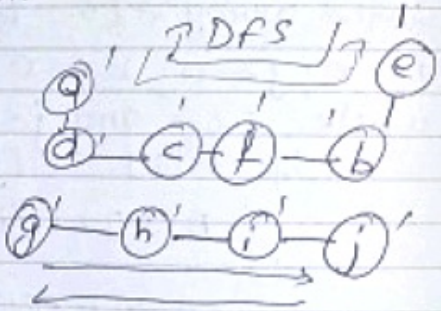
Depth First Search & Breadth First Search

→ Decrease and conquer.

Disjoint Graph.



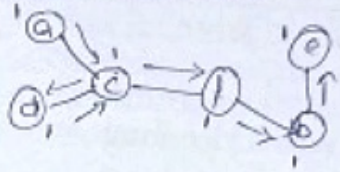
After vertex it should be mode as one'



Time of Experiment : ..... Page No. .....

Experiment No. ..... Experiment Result : ..... Date .....

In alphabetical order.



ALGORITHM DFS(G)

// Implement DFS traversal of a given Graph

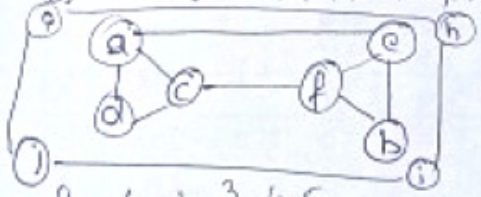
// I/P: Graph G(V, E)

// O/P: Graph G with its vertices marked with // consecutive integers. In the order they have been // visited.

```
count ← 0
for each vertex v in V do
    if v is marked 0
        dfs(v)
dfs(v)
count ← count + 1;
for each vertex w in V adjacent to v do.
        if w is marked with 0
            dfs(w)
```

BFS

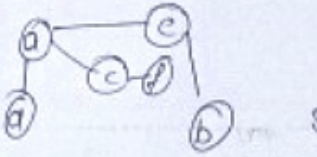
Queue is used to implement BFS.



0 1 2 3 4 5

q	c	d	e	f	b

all adjacents of a first node we have written.



d	e	f	h

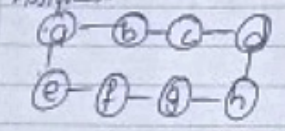
e	f	b

ALGORITHM bfs(G)
// Implements bfs traversal of a given Graph
// I/P:- Graph G=(V,E)
// O/P:- Graph G with its vertices marked with
// consecutive integers in the order the have been
// visited.

```
count ← 0
for each vertex v on V do
    if v is masked with 0
        bfs(v)

bfs(v)
count ← count + 1;
while the queue is not empty do
    for each vertex w in adjacent to the
                                front vertex
        count ← count + 1;
        add w to the queue.
    remove front vertex from the queue.
```

Assignment



29	34	45	45	68	89	90
29	34	34	45	68	89	90
29	29	34	45	68	89	90
17	29	34	45	68	89	90.

Date of Experiment : ..... Experiment No. ..... Experiment Result : ..... Page No. ..... Date .....

18/05/2023

Insertion Sort

ALGORITHM: insertionsort(A[0.....n-1])
// Sorts a given array by insertion sort.
// I/P:- An array A[0....n-1] of n orderable numbers
// O/P:- A[0---n-1] sorted in ascending order.

```
for i ← 1 to n-1 do
    v ← A[i]
    j ← i-1
    while j ≥ 0 and A[j] > v do
        A[j+1] ← A[j]      A[1]=89
        j ← j-1
    A[j+1] ← v             A[0]=45
```

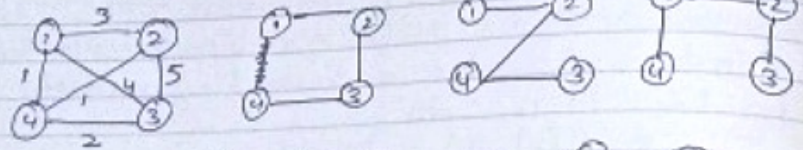
ex	0	1	2	3	4	5	6	i	n	n-1	v	A[j+1] = v
	89	45	68	90	29	34	17					j
	89	89	68	90	29	34	17	1	7	6	45	0
2	45	89	68	90	29	34	17					-1
	45	89	89	90	29	34	17	2			68	1
	45	68	89	90	29	34	17					0
	45	68	89	90	29	34	17	3			90	2
3	45	68	89	90	29	34	17	4			29	3
4	45	68	89	90	90	34	17					2
	45	68	89	89	90	34	17					1
	45	68	68	89	90	34	17					0
	45	45	68	89	90	34	17	5			34	-1
	29	45	68	89	90	34	17					4
5	29	45	68	89	90	90	17					3
	29	45	68	89	89	90	17					2
	29	45	68	68	89	90	17					1
	29	45	45	68	89	90	17	6			17	0
	29	34	45	68	89	90	17					5
	29	34	45	68	89	90	90					4
	29	34	45	68	89	89	90					3
	29	34	45	68	68	89	90					2
											-1	1
												0

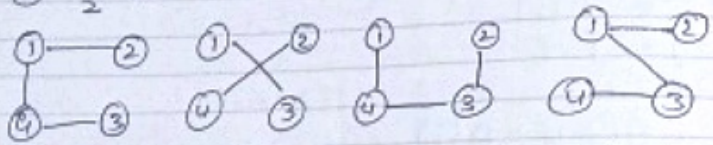
any node as starting node.

Prim's Algorithm. n^{n-2} n= vertex

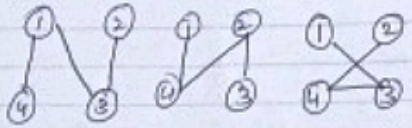
spanning tree:-

G(V,E) T(V,e)





If we have weighted graph.



ALGORITHM prim(G)

// Prim's Algo for constructing a minimum spanning tree

// I/P: A weighted connected graph $G=(V,E)$

// O/P: E_T the set of edges with all vertices

// which will make MST

$V_T \leftarrow [V_0]$

$E_T \leftarrow 0$

for $i \leftarrow 1$ to $|V| - 1$ do

find a minimum weight edge

// adjacent from v to u.

$e^* = (V^*, u^*)$ among all edges (v,u)

such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

return E_T

Date

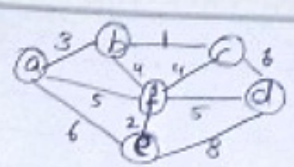
Page No.

Experiment No.

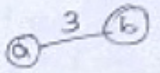
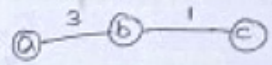
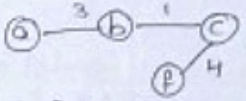
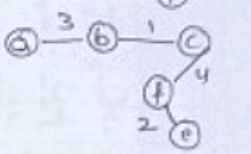
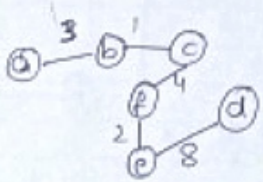
Experiment Result :

Date of Experiment :

Consider the Graph



start vertex a.

Sl no.	Tree Vertex	Remaining Vertices	Graph
1	a(- -)	b(a,3) c(a,∞) d(a,∞) e(a,6) f(a,5)	
2)	b(a,3)	c(b,1), d(b,∞) e(b,∞), f(b,4)	
3)	c(b,1)	d(c,6) f(c,4) e(c,∞)	
4)	f(c,4)	d(f,5) e(f,2)	
5)	e(f,2)	d(e,8)	
6)	d(e,8)		

$= 3+1+4+2+8 = 18$

~~Alg~~. ALGORITHM

Kriskal. Algo. Krushkals

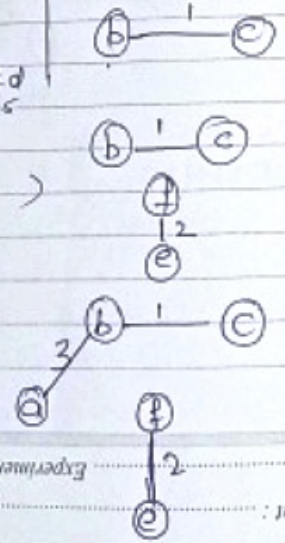
ALGORITHM

$E_T \leftarrow 0$
ecounter $\leftarrow 0$
$k \leftarrow 0$
while ecounter $< |V| - 1$ do
$k \leftarrow k+1$
if $E_T \cup \{e_{ik}\}$ is acyclic
$E_T \leftarrow E_T \cup \{e_{ik}\}$;
ecounter $\leftarrow$ ecounter $+1$

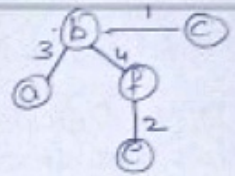
return E_T
(For same graph) in last page

Sln	Tree edges	sorted edges list
*		bc fe ab bf 1 2 3 4 cf af fd cd 4 5 5 6 ae ed 6 8
1	bc 1	fe ab bf cf af fd 2 3 4 4 5 5 cd ae ed) 8 6 8
2)	fe 2	( )
3)	ab 3	bf cd 4 8

Graph.

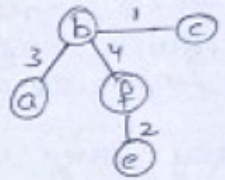


4) bf cf af fd cd ae
4 ed



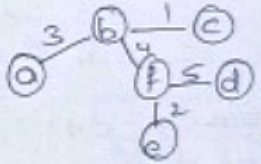
5) cf af fd cd ae
ed

(This is not acyclic so we should cf.)



6) af fd cd ae cd.
(af)

7) fd cd ae ed



8) cd ae ed
discord

$E_T = 3+1+4+5+2$
$= 15$

9) ae
discord.

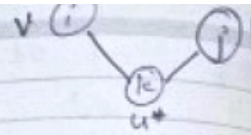
∴ Compare to prim's cruskal is efficient.

10) ed discord.

Dijkstra Algorithm
Shortest path from source to destination
→ single source shortest path problem.

(see) floid Algo

Name of Experiment : ..... Experiment No. .....
Page No ..... Experimen Result : ..... Date .....

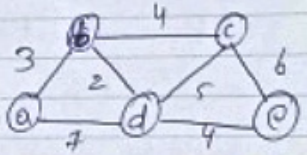


Dijkstra's

ALGORITHM Dijkstras (G, s)
// Dijkstra's algorithm for single source shortest path
// I/p: A weighted connected graph G = (V, E) with
// non-negative weights & its vertex s
// o/p:- The length dv of a shortest path from s to v

```
for every vertex v in V do
        dv ← ∞ ;
for i ← 0 to |v| - 1 do
        Vt = Vt ∪ {u*}
        for every vertex in V-Vt that is
                adjacent to u* do
                if du* + w(u*, u) < du
                        du ← du* + w(u*, u);
dv = dv + du
return dv
```

example:-

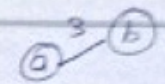


A to e

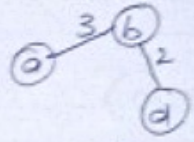
Slno	Tree vertex	Remaining vertex	Graph.
1	a(-,0)	b(a,3) c(a,∞), d(a,7), e(a,∞)	
2	b(a,3)	c(b,4), d(b,2) e(b,∞)	

Name of Experiment : ..... Experiment No. ..... Experiment Result : ..... Page No. ..... Date .....

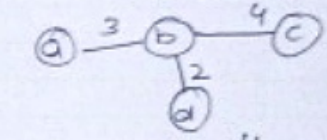
2) b(a,3) c(b,3+4) d(b,5) e(a,∞)



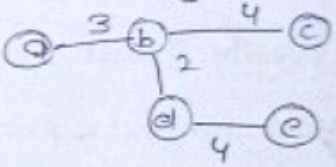
3) d(b,5) c(b,7), e(d,5+4)

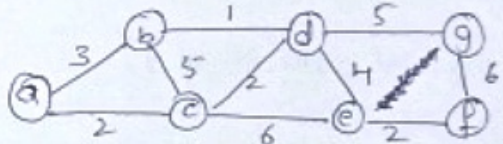


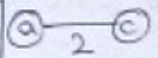
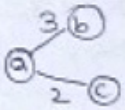
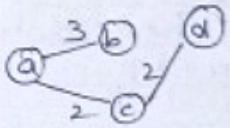
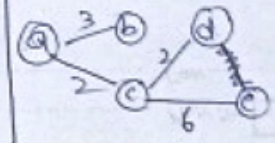
4) c(b,7) e(d,9)



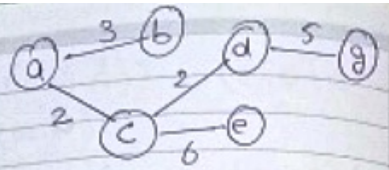
5) e(d,9) -



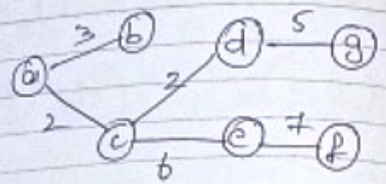


Slno	Tree vertex	Remaining vertex	Graph.
1	a(-,0)	b(a,3), c(a,2), d(a,∞), e(a,∞), g(a,∞), f(a,∞)	
2)	c(a,2)	b(a,3), d(c,2+2), e(c,2+6), f(a,∞), g(a,∞)	
3)	b(a,3)	d(c,4), e(c,8), f(a,∞), g(a,∞)	
4)	d(c,4)	e(c,8), f(a,∞), g(d,4+5)	
5)	e(c,8)	f(e,8+2=10), g(d,9)	
6)	g(d,9)	f(e,10)	

next page



f(e,10) –



Priority Queue.

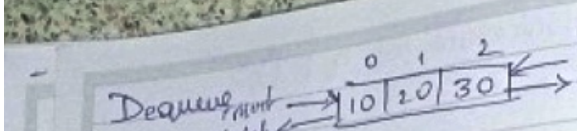
```
void insert_by_priority(int data, int q[])
{
   if (rear >= n-1)
   {
      printf("\n Queue is full \n");
      return;
   }
   if (front == 0 && rear == -1)
   {
      rear ++;
      q[rear] = data;
      return;
   }
   else
      check(data, q);
   rear++;
}
```

```
void check (int data, int q[])
{
   int i,j;
   for(i=0; i<=rear; i++)
   {
      if(data >= q[i])
      {
         for(j=rear+1; j>i; j--)
         {
            q[j] = q[j-1];
         }
         q[i] = data;
         return;
      }//end of if
   }//end of for loop.
   q[i] = data;
}
```

front	rear	data	i	j	q
0	-1				
	0	20	0		20 10
		10	1		20 10 10
		30	0	2	20 20 10
	2			1	
				0	30 20 10

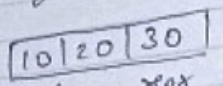
Dequeue

Dequeue

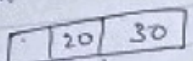


front = 0
rear = -1

insert_rear } ordinary
delete_front } queue.

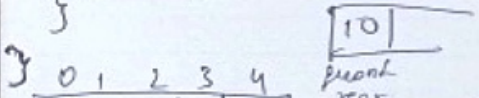


after delete_front there is a room for insertion




```
void insert_front(int data, int q[])
{
  if(front==0 && rear==-1)  ②
  {
    front++;
    q[front]=Data;
    rear++;
    return;
  }
  if(front==0)  ①
  {
    not possible to insert;
    return;
  }
  ③
  if(front>0 && front<=rear)
  {
    front--;
    q[front]=data;
    return;
  }
}
```

```
void delete_rear(int q[]
{
  if(rear==-1)
  {
    no elements in queue;
    return;
  }
  if(front==rear)
  {
    printf("Deleted elements is : %d\n", q[rear]);
    front=-1;
    rear=-1;
    return;
  }
  if(rear
  {
    pf(" Deleted item : %d\n", q[rear]);
    rear--;
    return;
  }
}
```



10
front = -1
rear = -1

rear = 3.

delete_front 10
front = 1

40
rear = 2

D-f 20
front = 2
rear = 2.

Towers of Anna