1. C PROGRAMMING

1. Data types

float, int, double, char.

[Size of int not defined, depends on h/w]

Modifiers → long, unsigned

- long int (8)
- long double (10B)
- unsigned int
- unsigned char. [0 to 255]

- **int** Size = 4B/2B → depends on word size of machine. Now 64 bit processor - when C come, some were 16 & 32: 16b — DOS - Turbo C (2) 32b - Unix. (4) (64 bit - 8 bit compiler not yet made).

	Size
• int	2/4.

64 bit → 32b compiler. → Dual core 32+32

	Size	Range
int	2/4	-32768 to 32767
float	4	-3.4×10^{-38} to 3.4×10^{38}
double	8	-1.7×10^{-308} to 1.7×10^{308}
char	1	-128 to 127.

IEEE 754

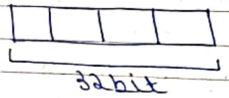
DATA TYPES

- float : 172.59.

1) ⇒ idea (not uniform) [Fixed point decimal]

2) 0.758 → 758×10^{-3}
   2466.12 → 246612×10^{-2}
   (mantissa, exponent)

[Floating point decimal]

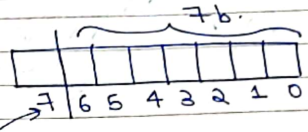


24 bit - mantissa
8 bit - Exponent.

- 350000 → 35×10^{4}
- -32.64 → -3264×10^{-2}
  (unsigned, bias)

[in C, -ve no stored in 2s compl...

- char range ? → has integral values only.



$2^7 = 128$ no possible : 0 to 127.

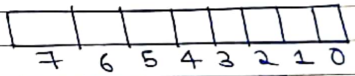
signed bit: 0 = +ve, 1 = -ve.

i). 10000000 → -128 (All -ve stored in 2s complement form.)

11111111.

range ⇒ -128 to 127. $(-2^{n-1}$ to $2^{n-1}-1)$

ii). unsigned char:



Range = 0 to 255 → 256 nos.

```
char x = 65;
printf("%d", x);   = 65
```
(convert Binary → decimal.)

```
printf("%c", x);  → A
```
(seen in char table by compiler).

(show the char)

* in memory → 0s/1s.

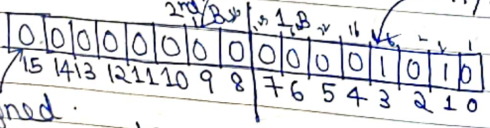
cout << x ; → A → default as ASCII value of x.
[C++]

* A = 65
B = 66
.
.
.
Z = 90
.
.
a = 97
b = 98.
.
.
z = 122
.
.
0 = 48
1 = 49
.
.
9 = 57

ASCii Codes.
↳ American Std. Code for Info interchange.

* record CS → C Programming.

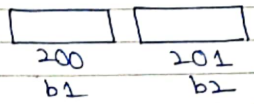
* int a = 310. n/256



signed.

b2 = 0
b1 = 10
n = 1100
4 | 76

Mem → 2B.



in Physical Mem

Q Which byte will go where?

- 0 | 10 (200 | 201) big-endian. [MSB to LSB]. (L-R)
- 10 | 0 (200 | 201) little endian (R-L) [LSB to MSB]

↳ 2 Methods.

Most m/cs follow this now. little Endian

- int a = 10; 0 | 10 on Paper → 10 | 0 Mem (28)
- int a = 120; 0 | 120 → 120 | 0.
- int a = 300; 1 | 44 → 44 | 1
- int a = 500; 1 | 244 → 244 | 1
  38 → 38
- int a = 550. 2 | 38 → 38 | 2

120.

1024	512	256	128	64	32	16	8	4	2	1
			1	1	1	1	0	0	0	
0	1	0	0	0	0	0	1	0	0	
		1	1	1	1	1	0	1	0	6

1 . 0 1 0 ~11

Q union test
{
int a;
char b;
};
main()
{
union test t;
t.a = 325;
printf("%c", t.b);
}

int = 2B (common)

A 65

→ E.

Little order

a

69	1
200	201

b.

1 69

E = 69

int a = 2B
char b = 1B.

max(2, 1) = 2B space allocated.

Q void main()
{
int a = 321;
char *p;
p = (char *) &a;
printf("%c", *p);
}

add of int datatype

type casting (∵ need to point to int dt)

p will point to a char datatype

∵ you are telling char pointer to later value.
∴ it will read 1B.

∴ it → access

1	1
65	1
200	201

Little order

p
200 ← add of a in p.

∴ Ans A (read 1B)

Q char a = 127;
a++;
printf("%d", a);

E can apply ++ on char

0	1	1	1	1	1	1	1
7	6	5	4	3	2	1	0

127 + 1 + 1
10000000

−128

(−128 to 127)

cyclic behaviour.
∴ overflow.

• if a = a + 5; → −124

Q main()
{
char a=10;
while(a>0)
{
printf("Hi");
a++;
}
}

not infinite loop.

⇒ 118 times printed

10, 11 ..... 126, 127.

$127 = 10 + (n-1)$
$127 = 9 + n$

Q How many times condⁿ will be checked?

→ 119. ✓

Q main()
{
unsigned char x=126;
while(x>0)
{
pf("Hi");
x++;
}
}

126 ✓
127 ✓
⋮
255 ✓

255 − 126 + 1

Range → 0 to 255.

(255 − 126) + 1 = 130

How many times printed?

including 126.

* Operators

{+, −, *, /, %}

Low Precedence | High Precedence

d = v*v − u*u/2*a;

↳ $v^2 - \frac{u^2}{2} * a$.

d = (v*v − u*u)/(2*a);

1)* int a=5;
float b=12.6f;
double c=17.2;
x = a+b*c;

double.

dT of exp = largest dT used in the exp.
∴ double.

2) x = a+b+c*d; → separate m/c inst. for this.

(ii) Highest prec? → *

(iii) 3 Things to be seen. [which will be executed first : null (not necessary)]

- precedence
- Associativity.
- order of evaluation (how compiler exec)? (undefined)

(Order of evaluation is undefined in C lang).

((a+b) + (c*d))

h/w dependent

(is left to Right associative)
like, Parantheis order defined.

Size of int: undefined.
order of eval: undefined.
→ can define (if fully parenthesized).
can tell result. ✓

Q int x; int a=3, b=6; GATE QS

x = f1(10) * f2(a) + f3(b);

- which funct^n will be called first?
  → undivided.
- result = known ✓

* HISTORY OF C

- developed by Dennis Ritchie.
- but long that time.
- C lang was developed to implement Unix on main frame m/c. (32 bit).
  C's design was opened.

- 16 bit / 32 bit implementation, was practised.

. ++, -- (post/pre increment).

→ int a=5;
a++;
printf(a); → 6

++a;
printf(a); → 6

. Pre-increment — Post-inc

int a=5, b=0;
b = a++;
pf(a,b);
(6,5)
[increment after]

int a=5, b=0;
b = ++a;
pf(a,b);
(6,6)
[inc. before].

Q int a=5 b=4;
b = b + 2 * a++;
printf(b);
5x2+4.
14 ✓

int a=5, b=4;
b = b + 2 * ++a;
pf(b);
6x2+4
16. ✓

++ → highest prec.

Q int a=5, b;

b= a++ + ++a + ++a + a--;

↳ compiler Specific ans. (undefined).

Q b= a++ + ++a; (Undefined).

op1

5 + 7

⇒ 12.

post > pre inc prec. inc

a= 5 6 7

op2

does it come to a

a

5 6 7

7+7=14

a= 5 6 7

7+7=14

a | b

5 | 5.

5+7=12

↳ holding prev value.

[Shouldn't use inc/dec operator on the same variable, more than 1 time, in a single expression].

Q1 int a=5; → 775

a 5 → 7 7 5

printf("%d %d %d", a++, ++a, a++);

an implem. (platform after pre)

8 6

7 5

Q void fun(int a, int b, int c)

{ ≡ L to R. formal ← Concept

}

void main()

{ int x=5, y=2, z=7;

fun(x, y, z);

} Right to L. ↑ actual.

x

y

z

Stack

x=a

y=b

z=c.

reading: L → R

(C/C++)

| copying/reading: R → L (stack)
| filling: L → R

Q1: Soln.

R → L.

Q
```
void fun(int a)
{
 if(a>0)
 {
  printf("Hi");
  fun(a--);
 }
}
fun(3);
```

→ recursive function

X[3 times] Ans.

Sol:

Hi 3>0
Hi Hi
Sol3

[infinite calling]
↳ not possible in recursion
: Stack overflow.

fun(--a); ✓

* Conditional operators

<	&&
<=	||
>	!
>=	
==	
!=	

These give true/false.
↓ ↓
Any other value 0

Q
```
int a=5, b=5;
while(a=b)
{
 printf("Hello");
 a++;
}
```

→ always a=5.

→ infinite loop

[in Java → error] : true ✓
C/C++ : 1/2/... : true

* Short circuit

Q int a=2, b=3, c=4;

if(a<b && a<c) → true
T T

if(a>b && a>c) → false.
F
↳ don't need to check this cond if 1st is F [in and operator]

Q int a=2, b=3, c=4;
T
if(a<b || a<c) → don't need to check : True
in OR (if 1st is true)

if(a>b || a>c) : false
F F

Q int a=2, b=3, c=4;

if(a<b || ++a<c)
 T
ps(a,b,c);

↳ this condⁿ won't be checked (based on SC).

⇒ (2,3,4)

⇒ ++(highP), can happen; this will be evaluated (depends on order of eval.)

int A[n];

Q if(i<n && A[i]>10)

↳ won't check this one if i<n : F.

int A[10];

if(A[i]>50) : error (if i>10)

if(i<10 && A[i]>50)
 F

↳ no check.

Page No.: 17 Date: YOUVA

* Bitwise Operators

↳ can perform on integer type data (int/char).

& - AND
| - OR
~ - NOT
^ - EXOR
<< - Left Shift
>> - Right Shift

int a=10, b=3

⇒ Binary form (8 bit)

a ⟶ 00001010 : 10
b ⟶ 00000011 : 3
1) a&b ⟶ 00000010 : 2

main()
{
 int a=10, b=3;
 printf(a & b); ⟶ 2
}

a ⟶ 00001010
b ⟶ 00000011
2) a|b ⟶ 00001011 : 11
3) a^b ⟶ 00001001 : 9 (tossing a coin H/T).
4) a<<1 ⟶

Q. a → 00001010 ⇒ 10.

a<<1 → 00010100. ⇒ 20. (Unary op)

[Val of a = unchanged]

a = a<<1 [a changed]. *2¹

pf(a<<2) → 40. *2².

pf(a>>1) → 5 *1/2¹.

Right Shift by 1 place.

(1 byte).

Q. a → 00001010 [10]

~a → 11110101 [-11].

2s complement

char a = 10;

00001011 ⇒ -11.

Q. 8 ~ → -9.

6 ~ → -7.

Q.
```
int n=512;
int count=0;
while(n>0)
{
  count++;
  n=n>>1;
}
pf(count);
```

RS = /2ⁿ

c=1.
n=256.
c=2.
n=128.
c=3
n=64
n=4.
n=32

512 256 128 64 32 16 8 4 2 1.

n=5 n=16 c=6 n=8 c=7 n=4 c=8 n=2 c=9 n=1

⇒

count	n
1.	256
2	128
3	64
4	32
5	16
6	8
7	4
8	2
9	1
10	0

← Tabular (approach).

Ans = 10.

512 → 1000000000

511 → 111111111 : 9 times.

2ⁿ takes n+1 bits like 4 takes 3.

* Switch Case

```
int a=14, b=3;
switch(a & b)
{
case 1: "One";
case 2: "Two";
case 3: "three";
case 4: "Four";
}
```

(Break not there)

```
16 8 4 2 1
 1  1  1  0
 0  0  1  1
 0  0  1  0   → 2.
```

o/p → "two" "three" "four".

case 1: ✓
case 'a': ✓
case 1 : X ← not Syntax error. [nothing will be printed] (because a label).
case 1.1: X.

- Switch without cases.

* Storage classes

1. Auto
2. Static
3. Extern
4. Register

→ void fun()/add() ← Stack Allocation Strategy.
{
int a=10, b=5, c;
c=a+b;
pf(c); pf(c,x,y); X.
}

main()
{
int x=7, y=10; → visibility/Scope of var. within block.
add();
pf(x,y);
}

auto

main: x = 7, y = 10

add(): a = 10, b = 5, c = 15. (removed from stack)

- lifetime of var: execution of fun^n.
  (extent)
  → how much time in mem?

[till fun^n AR present in Stack mem]

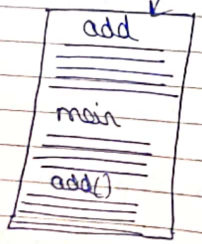
Add.c

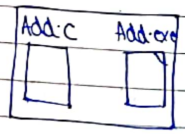
```
int add(int a, int b)
{
  int c;
  c = a+b;
  return(c);
}

main()
{
int x=10, y=5, z;
  z = add(x,y);
  pf(z);
}
```

- for compiler, need to save.
- Saved in HD as add.c.

[m/c code → add.exe]

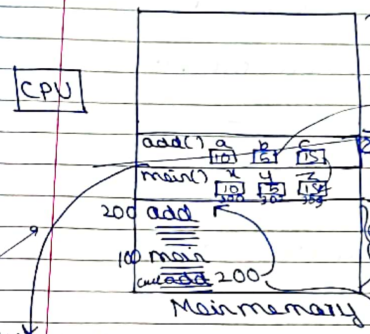
how will it look?





- when run?

Add.exe can run independently.
Load.



CPU → exec m/c code [program]

Activation record / Stack frame.

[to run]
HD → MM.
[loading]
(bringing program from HD to MM).

MM into Buffer.
(Every byte own Address).

- write this add, when prog came into MM. (during loading).
[Function Binding]
- loader (with OS) does that. [gives CPU beginning address of main]

* Now execution:

- CPU is kept in wait loop, when no process.
- int x=10, y=5, z;
  - Now memory milgyi,
  - Main → [allocate memory to var, & keep with garbage]
  - data binding [now main knows its var]
  [run time]

funct^n knows the range of its data.

→ when funct^n called, mem of var created.

Q Current funct^n running will access the TOS of stack.

↑ auto Storage Class.

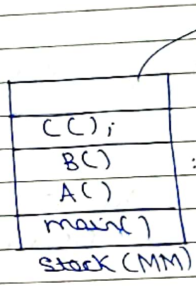
- [AR created for every function called, also printf()]

11/8/19 Sunday

Lecture 2

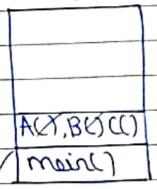
```
C()
{
  ≡
}
B()
{
  ≡
  C();
}
A()
{
  ≡
  B()
}
main()
{
  A();
}
```

→ When C finishes, go to next line after funct call C in calling func: B



→ stopping in C.

```
*  C()
   {
     ≡
   }
   B()
   {
     ≡
   }
   A()
   {
     ≡
   }

   main()
   {
     A();
     B();
     C();
   }
```



At a time,
Stack Size = 2.

Control Flow of prog

```
A()
{ ==
}  → returns back to calling func. to →.

main()
{ ==
  A(); → control transferred to A()
}
```

*
```
A()
{
  if(_)
  { ==
  }
  else
  {
    A();
  }
}

main()
{
  A();
}
```
← funct^{n} calling itself (recursion).

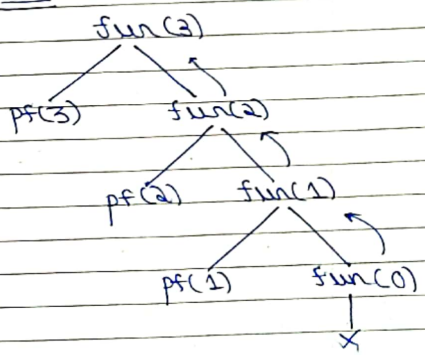
STACK IN RECURSION

Page No. 27
Date:

Q1
```
void fun(int n)
{
  if(n>0)
  {
1. printf(n);   → printing while Ascending.
2. fun(n-1);
  }
}

void main()
{
  int a=3;
  fun(a);
}
```

o/p
3, 2, 1

Sol^{n}:



o/p
3
2
1

→ returns back to main (func by function)

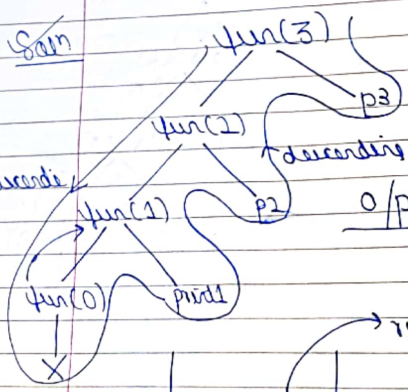
Q2
```
void fun(int n)
{
  if(n>0)
  {
    fun(n-1);
    pf(n);
  }
}
```

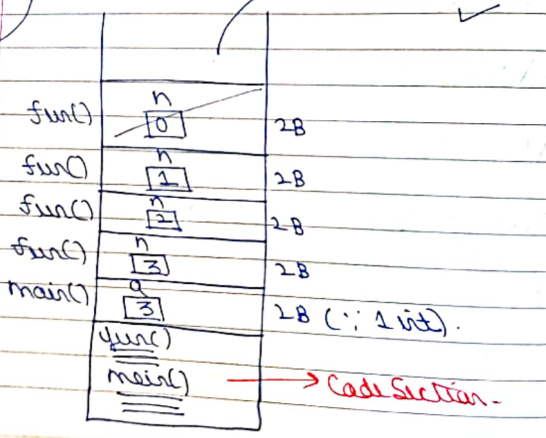
```
void main()
{
  int a=3;
  fun(a);
```

fun(3)
fun(2)
fun(1)
fun(0)

o/p 1,2,3

Soln





(current executing funct^n will access its top most AR)

⇒

2 Phases of recursion

1. Calling / Ascending
2. Returning / Descending.

Q. n=3
while(n>0) → In loop, only ascending.
{
pf(n); o/p 3,2,1
n--;
}

for printing 1,2,3: change logic, code.

1. [In recursion, both Asc & Desc].
   - [recursion > loop]
   - loop doesn't use Stack, it's part of funct^n (can't Store)
   - recursion Stack; depends on funct calls.

→ Loop is space effective.
→ Loop is better than recursion in our term (in terms of speed & Space)

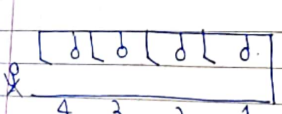
* Recursion v/s Loop

- Why recursion?

(Most problems solved with recurrence relation in Maths, thats why, recursion is prog introduced)

- [Every recursion can be converted into a loop & vice-versa.]

* Types of Recursion → like Spring/rubber band.



① 1. Switch on Bulb.
2. Go to next room

② 1. Go to next room
2. Switch on bulb.

```
void fun(int n)
{
  if (n>0)
  {
    1. ——→ executed at calling time
    fun(n-1);
    3. ——→ executed at returning time
  }
}
```

n+1 ARs will be created (excluding main)

$S(n) = n+1$

O(n) = Space complexity
↳ order of.

- $p(x) = x^2 + 3x + 5$

$p(n) = n^2 + 3n + 5$ (Quadratic eqn).

n, n^2, n^3 → more.

Head recursion

```
void fun(int n)
{
  if(n>0)
  {
    fun(n-1);
    ==
    ==
  }
}
```

execute while descending

Tail recursion

```
void fun(int n)
{
  if(n>0)
  {
    ==
    ==
    fun(n-1);
  }
}
```

executed while ascending

- Tail recursion Similar to loop. (easily convertible).
- ==
  fun(n-1)
  ==
  (difficult to convert).

```
void fun(int n)
{
  if(n>0)
  {
    fun(n-1);
    fun(n-1);
    ==
  }
}
```

If a recursion is calling itself, more than 1 time → Tree recursion
If 1 time → linear recursion.

```
void fun1(int n)
{
  if(n>0)
  {
    ==
    fun2(n-1);
  }
}
void fun2(int n)
{
  if(n>0)
  {
    ==
    fun1(n/2);
    ==
  }
}
```

Indirect recursion

f1 calls f2
f2 calls f1.
:
(will stop somewhere)
(Taylor makes recursion)

```
void fun(int n)
{
  if(n>0)
  {
    ==
    ==
    return fun(fun(n-1));
  }
}
```

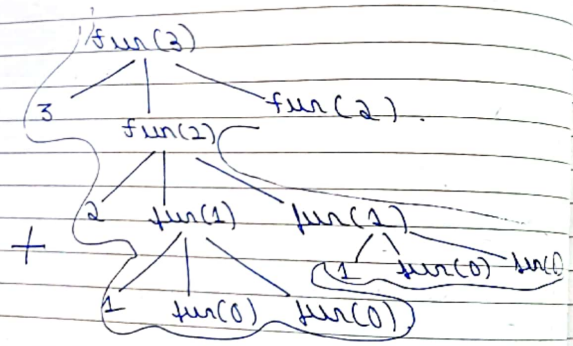
result will be given to this call.

Nested recursion

- recursive call is acting as a parameter for recursive call.

Tail recursion

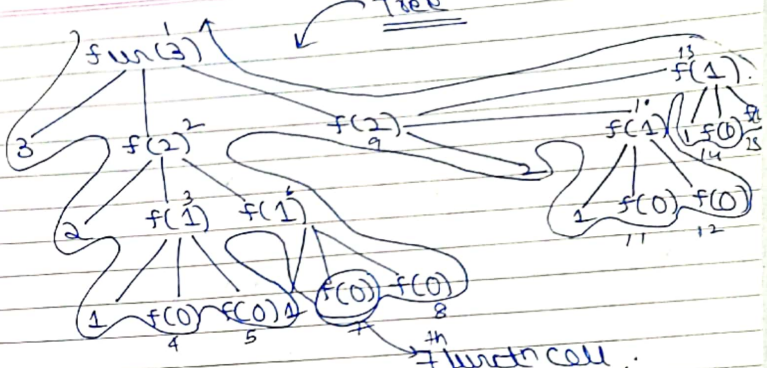
```
void fun(int n)
{
  if(n>0)
  {
    pf(n);
    fun(n-1);
    fun(n-1);
  }
}
main()
{
  int a=3;
  fun(a);
}
```



3 2 1 1 1 1 2 3

3 2 1 1 3 2 1 1

Tree



O/P 3 2 1 1 2 1 1.

- Total 15 function calls

n=3 ⟶ 15

n=4 ⟶ 31 calls.

$$n \longrightarrow 2^{n+1} - 1 \text{ calls}$$

- Time depends upon no. of prints.
  - work ↗ depends on no. of calls.

∴ Time taken by funct^n

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

$S(n) = 2^n - 1$ ∴ $O(2^n)$

[exponential time].

* $S(n) = n+1$. (Height of a tree)

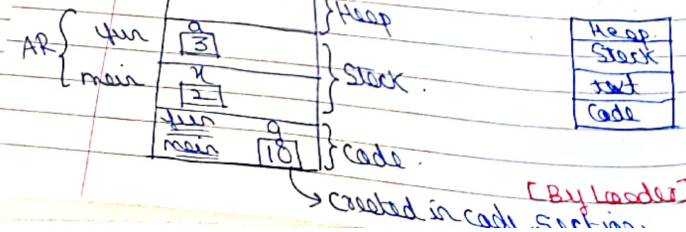
$O(n)$.

↳ Space taken by func depends upon height of tree / no. of ARS.

STORAGE CLASSES (returning)

```
int g=10;   // Global Var.
            [all funct can access it]
            (not created in Stack)
void fun()
{
int a=3;  → local to funct.
g++;
pf(g);      11
}
main()
{
int x=2;
pf(g);      10
fun();
pf(g);      11
}
```

O/P → 10, 11, 11.



↳ created in code section. [By Loader]

→ [GV space created during loading time, before execution, in Code Section]
→ Loader creates GV, ∴ puts default 0, if uninitialised. (during loading time)
→ all funct's can access global V.

Ed
compile time
loading time
Runtime ×
Binding time

⇒
1) Prep time
Editing time
Compile "
loading "
Binding "
Runtime "
] Static.
] dynamic.

✓ G → during loading. (Static time)
[Static memory allocn]
✓ (x & a) → memory made during run-time
(Dynamic)

Stack mem	→ Dynamic (AR)
Code mem	→ Static

Date:

* int a=3 ;

Attributes of a Variable :

1. Name
2. Type
3. Size
4. Value
5. Address
6. Scope
7. Extent

. Name → Design/editing — Static
. 2 → Editing — Static
. 3 → Compiler time — Static
. 4 → Compiler/run time (dynamic)
. 5 → dynamic (if add in code Section, by exe → Static)
. 6 → Compile time (Static)
. 7 → Compile time (Static)

[Only memory is dynamically allocated in Stack]

- [If Type & Size needs to be dynamic, go for Heap]

- Global → Scope (all functn)
  → Lifetime (prog)

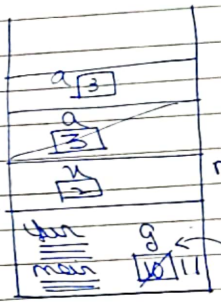
Page No.: 39
Date:

* STATIC IN C → extent : Global
  Scope : Local

→ like Global, but only that part - functn can access
- not main

```
Void fun()
{
  int a=3;
* static int g=10;
  a++;
  pf(g);
}
main()
{
  int x=2;
  fun();
  fun();
}
```

g=10 → not exec many time (init 1 time during load)



o/p = 11, 12

[Static var (by default)] = 0

a → belongs to function call
g → belongs to function (Static/Global)

* Storage Classes

	Class	Scope	Extent
Stack	auto	local	local
Created in code	extern	global	global
	Static	local	global
Created in CPU or stack	register	local	local

→ fastest of all variables [created in CPU]

auto → AR (inside Stack)

```
* int a=10;
  void fun()
  {
    int a=5;
    pf(a);
  }
```

o/p → 5
(will look in its own AR first).

* Scoping Rule (not in Gate Syllabus)

1. Static Scoping
2. Dynamic Scoping → depends on
   - Nested function
   - Free variable

```
Main()
{
1. int a=10;          → in AR of main
3. function fun()     ← Nested function
4. {
5.   int b=5;
6.   print(a+b);      → Free V (not local)
7.   a=a+b;             [gets it from parent function]
8. }                  - funct accessing non local V.
2. fun();
9. print(a);
}
```

after 1 to directly 2.
then 3. (AR of called).

[1 funct puts a pointer to its Parent AR]

o/p = 15, 15

→ Var Scoping (indirect Access).

Static Scoping

```
Main()
{
① int a=10;
③ fun1()
  {
④   int b=5;
⑥   fun2()
    {
⑦     int c=2;
⑧     printf(a,b,c);
    }
⑤   fun2();
  }
② fun1();
⑨ }
```

→ o/p 10,5,2

* If a funct^n is accessing free V, it will look for it in its parent funct^n / GP funct^n & so on.
[At compile-time]

Dart Is Nested functions OS

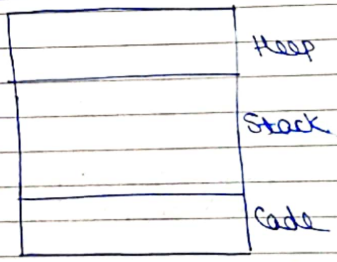
Dynamic Scoping

```
Main()
{
1   int a = 10;
9 6 Display()
    {
10  print(a);
    }
3   fun()
    {
4     int a=5;
5     Display();
    }
2   fun();
8   Display();
}
```

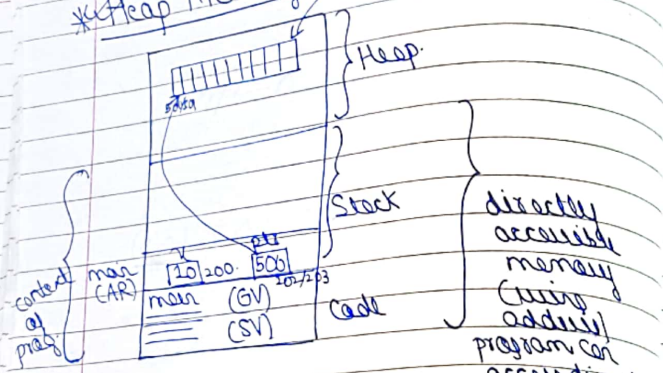
a → free V.

→ 5, 10 (o/p)

- Main & fun both calling display.
- FV are binded with that of calling funct^n [pointer to calling funct^n]



Heap Memory



- Heap → external/outside to a program.
- Heap memory is treated like resource. (like rent things) → req/return.
  - eg: Ola Cabs
  - acquire/release memory.
- Its indirectly accessible memory. [not directly my thing].
- (when heap/when stack?) prog decides.
- Use Pointers.

```
void main()
{
    int x = 10;
    int *ptr;
```

integer pointer will carry address of an integer variable.

returns SA. → 20B

```
ptr = (int*) malloc (10 * sizeof(int))
```

- ptr points to int.
- to request memory in heap.

[SA is sufficient to hold entire chunk]

[main goes to ptr & goes to address]
∴ 2 Step (Indirect Access).

```
ptr[0] = 15;
.
.
.
free(ptr);
```

GV/SV → Code
remaining V → Stack
malloc → Heap.
Name, ptr = dynamic name

C++

Gate

```
ptr = new int[10];

delete[] ptr;   // delete array ptr.
```

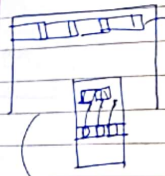
* When you dont need heap, release it.

if pos (ptr); (not written).

• (if prog ends, heap M deleted).
[Heap M released]

chrome

• [if all apps need to be in Stack, statically size of stack must be decided by chrome]
So need to take in ~~Stack~~ Heap.
[Stack contain Pointers to heap]

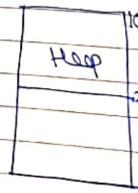


for every tab, heap size getting full.
· Heap full

We need to free heap space, after closing that particular tab.

[If deciding at run time → Heap]

Free list ; Used list maint in Heap mem



* Free list: 2030 ~~2000~~ - 10K ; 2000-2009.

Used list:

2000-2009.
2010-2029.

• If we are not releasing, and not even using.
→ Memory leak
- Memory leak more & more: Crash.

Stack → dynamic
Heap → fully dynamic

→ Size of stack pre decided.
like: political meeting
(Stage, benches (decided), ground space with chair (+/-)).

[Heap

Date 48

* POINTER

1 used for storing address
2 used to access data indirectly.

int x = 10;
int *p;

(all peripherals → resource).
3. you can access resource using pointer.

eg: FILE *fp;
Socket *SK; (accessing internet)

Can't access resource without Pointer.

* syntax → Main

- int x = 10; → data V.
- int *p; → address V (Pointer)

declaration of pointer.

Memory

x	ptr
10	garbage
200/201	250/251

ptr taking 2B now.

- p = &x; (initialisation)

ptr
200
250/251

Page No. 49 YOUVA
Date

- printf("%d", *p); 10
  - go to add 200, & fetch value.
  - dereferencing / dereferencing
  - going to that add.
- printf(x) = 10
  pf(p) = 200
  pf(&x) = 200
  pf(*p) = 10
  pf(&p) = 250

- ~~int~~ pointer can only point an int data.

int a = 10; float b1 = 12.5
int *p1; float *p2;

p1 = &a; p2 = &b;

p1 = (int*) &b;

1) int pointer can point to float data, but you need type casting for that.

2) int *p1; — 2B 2B
   float *p2; — 4B 2B
   char *p3; — 1B 2B.

pointer → unsigned integer

[Heap]

* POINTER

1. used for storing address
2. used to access data indirectly.

```
int x = 10;
int *p;
```

(all peripherals → resource)

3. You can access resource using pointer.

eg: FILE *fp;
Socket *SK; (accessing internet)

Can access resource without Pointer.

* Syntax → Main

- int x = 10; → data V.
- int *p; → address V (Pointer)

declaration of pointer.

Memory

x	ptr
10	garbage
200/201	250/251

pts taking 2B raw.

- p = &x;

ptr: 200 (250/251)

initialization

- printf("%d", *p); → go to add 200, & fetch value. → 10
  → dereferencing / dedereferencing - going to that add.
- printf(x) = 10
  pf(p) = 200
  pf(&x) = 200
  pf(*p) = 10
  pf(&p) = 250.

- ~~int~~ pointer can only point an int data.

```
int a = 10;          float b1 = 12.5
int *p1;             float *p2;
p1 = &a;             p2 = &b;
p1 = (int*) &b;
```

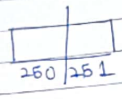
1) int pointer can point to float data, but you need type casting for that.

2)
```
int *p1;   — 2B   2B
float *p2; — 4B   2B
char *p3;  — 1B   2B
```

pointer → unsigned integer

Size of pointer = Size of int (in C compiler)

if int → 4, p → 4 ✓



⇒ 2B (16 bit).

Range of Address

0 to $2^{16}-1$: 0 to 65,535.

- [2B pointer can access memory from 0 to 65,535.
  ↓
  2^{16} ⇒ 64 KB (max).]

⇒ ~~Large~~ Size Memory can be broken down into chunks of 64 KB memory.

int *p1; pf(*p1);

↳ if dereference, it goes there, & access 2B.

- char *p3; pf(*p3);

↳ dereferencing time, knows data of DT to point.

↳ if dereferencing, it goes there & access 1 Byte.

Page No. 51

Date

* Pointer to an Array

- int A[5] = {2, 4, 7, 10, 12};
- int *p;
- p = A; // SA of array → add of array integer.
- p = &A[0]; (same)
- p = &A X

	0	1	2	3	4
A	2	4	7	10	12
	200/1	2/3	4/5	6/7	8/9

(address of each Byte)

p: 300/301

you will get value 200, but next, you will go outside of array.

∵ &A : add of entire array. [add of array object].

in m/c code, array won't be there. (SA happens)

- name of array → Address.
- A[0] = element.

- pf(A); — 200
- pf(p); — 200.
- pf(*p); — 2
- pf(A[2]); — 7
- pf(2[A]); — 7
- pf(A+2); — 204.
- pf(*(A+2)); — 7.
- pf(*(p+3)); — 10
- pf(*(3+p)); — 10
- pf(p[1]); — 4.

↳ Pointer with an array name

* gives Value

[Pointer to array]

1- Can we pointer like an Array name.

[You cant change address of A].

2- You can we array name, like a pointer.

Array name ⇔ pointer to an array
(Similar) ✓

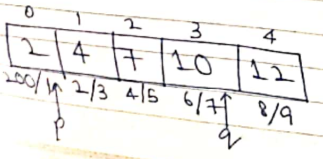
Ex int A[5] = {2, 4, 7, 10, 12}
int *p = &A[2];
printf("%d", p[-1]); ⇒ 4.
printf("%d", *(p-1)); ⇒ 4.

* Pointer Arithmetic [5 operations]

- All with are not allowed on pointers [only limited & meaningful].

① int A[5] = {2, 4, 7, 10, 12}
int *p = A;

① p++; p = 202. (i=5, i++; next int.)
↳ next data item.



if p = 200, then p++, moves p to 202.

(p++ dependent on dT of pointer)

* Pointer Arithmetic is dependent on data type of a pointer.
P++ ⟶ if P = char, p moves by 1B
p++ ⟶ if P = float, p moves by 4B

② p--;

③ p+3, if p is at 200; it gives add of 3rd data element i.e. 206. [Just get, don't move p].

④ p-3;

int A[5] = {2, 4, 7, 10, 12}
int *p, *q;
p = &A[0];
q = &A[3];

• int d = q - p; (Subtracting 2 pointers)
↳ 6/size of dataT = 3.
↳ How many elements away from each other?

• int d = p - q; ⟶ -3

Scanned with CamScanner

⇒ (cant add 2 address
(∵ Pointer addition not allowed)
↳ Meaningless.

Q. int A[5] = {2, 4, 7, 10, 12}
int *p = A;

Preced
H post ++, .
L pre++, *

1. ++*p; = 3
2. * ++p; = 4
3. * p++; = 2

Operator	Precedence	Associativity
post ++, --, ., →	high	L-R
pre ++, --, *	Low	R-L

1. *p = 2, then inc → 3
++(*p); [Value will inc]
2. 4
3. * p++;
↳ take the value, & then inc the pointer i.e address
2.

* Pointer to Pointer (Double Pointer)

NOTE: int A[5]
bA = 2000, size of Int 2.
pf(&A+1); ⟶ 2010
pf(A+1); ⟶ 2002.

* 2D Array

int A[3][4] = {{2,4,6,8}, {1,3,5,7}, {1,2,3,4}}

A	0	1	2	3
0	2 200/1	4 202/3	6 204/5	8 206/7
1	1 208/9	3 210/11	5 212/13	7 214/15
2	1 216/17	2 218/19	3 220/21	4 222/23
3	224/25	226/27	228/29	230

→ collection of 1D Arrays.
A(3×4)
[Full in Stack]

• In Memory ⟶ 1D Array (row Major).

⇒ [2D Array is an array of 1D Arrays]

pf(A[1][2]); — 5 — 10
pf(&A[1][2]); — 212 — 604
pf(A[1]); — 208 — 600
pf(A[1][0]); — 1 — 5
pf(*A[1]); — 1 — 5

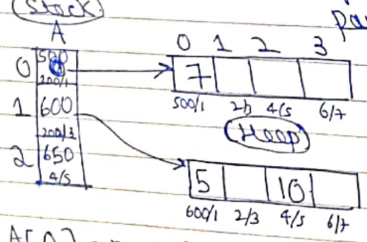
200 pf(A); — 200
500 pf(*A); — 200
7 pf(**A); — 2
202 pf(A+1); — 208.
600 pf(*(A+1)); — 208.
604 pf(*(A+1)+2) — 212
10 pf(*(*(A+1)+2)); — 5.
604 pf(A[1]+2) — 212
10. pf(*(A[1]+2) — 5.

10
604
600
5
5
200
5

A[i][j] —— *(*(A+i)+j)
& A[i][j] —— *(A+i)+j
A[i] —— *(A+i)

* int A[3][4]; → stack Section

* int *A[3]; → array of int* [Stack] pointers of size 3



* A[0] = new int[4];
A[1] = new int[4];
A[2] = new int[4];

• 2D Array → double Pointer.

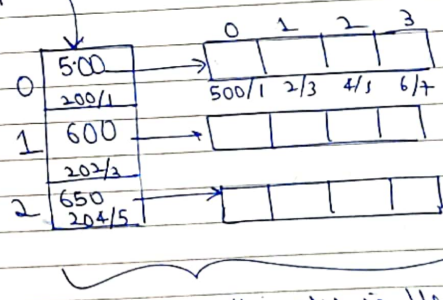
• nD Array → n level pointer.

Q int **A;
A = new int *[3];
→ array of int pointers size 3.

int **A;
A = new int *[3];
A[0] = new int[4]
A[1] = new int[4];
:

[Fully dynamic 2D Array]

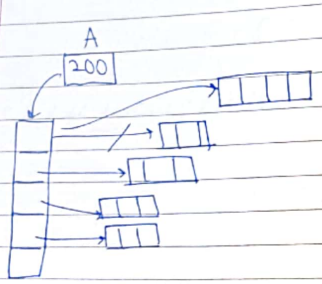
A (only this in Stack).



everything else in Heap.

Q int **@ A;
A = new int *[3];
A[0] = new int [4];
A[1] = new int [6];
A[2] = new int [10];

→ allowed.



(Depending upon prog req, can be dynamic)

- int A[3][4];
- int * A[3]; →
- int **[A]; → only pointer in Heap.

Pointer to a function

→ revolution in Software industry.

```
void display()
{
printf("Hello");
}
```


[func must have addr]
(Prev lec)

```
void main()
{
  void (*fp)();   ← declaration
  fp = display;   ← initialisation
  (*fp)();        ← dereferencing [func call]
}
```

→ must be in brackets.

Q
```
int max(int a, int b)
{
  return a>b ? a:b;
}
int min(int x, int y)
{
  return x<y ? x:y;
}
void main()
{
  int (*fp)(int,int); // can point to both.
  fp = max;
  pf((*fp)(10,5)); — 10
  fp = min;
  pf((*fp)(10,5)); — 5
}
```

[A pointer to a func can point to all those func who prototype is same as that of declaration].

Similar to polymorphism.
[1 pointer to diff diff functions].

- Pointers → pins.
  (can attach anything & very efficient).

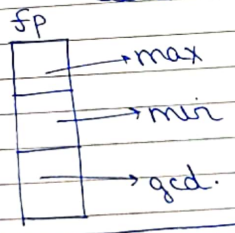
```
int f(int a)
{
==
==
}
g(int a)          [int] (default)
{
==
}
void h()          [void] default
{                 (argument)
==
==
}
int (*p)(int)
void (*q)(void)
```

1. p=f ✓
2. p=q ✓
3. p=h ✗
4. q=h ✓
5. q=f ✗

*1) declaration

- int A[5]; → Array of integers.
- int *A[5]; → Array of integer pointers.
- int f(int); → func which takes int as parameter & returns int.
- int f2(int, int); → " " 2 " "
- int *fp(int); → function which takes int as parameter & return integer pointer.
- int (*fp)(int); → L→R Assoc. (base)
  → pointer to a function.
- int (*fp[3])(int, int);
  → array of pointers to a function which takes 2 int parameters & returns int.

*



Reference in C++

& → reference
[init while declaring]

```
void main()
{
  int a=10;
  int &x=a;
```

a/x.

10

200/201

in m/c code, a ka jagah 200.

- [reference is an alias/nickname of a]
  x is alias of a.
- won't occupy any space/memory.
- useful in parameter passing.

```
x++;
pf(a); ⟶ 11
```

PARAMETER PASSING

1. Call by Value
2. Call by Address
3. Call by reference.

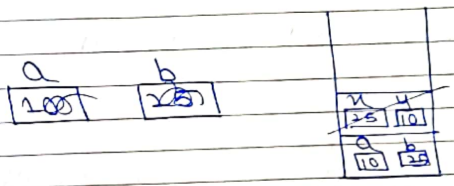
```
void swap(int x, int y)
{
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

(x, y ← formal parameters.)


```
void main()
{
  int a=10, b=25;
① swap(a,b);
  pf(a,b);
}
```

Call by Value.

(a, b ← actual Parameters.)



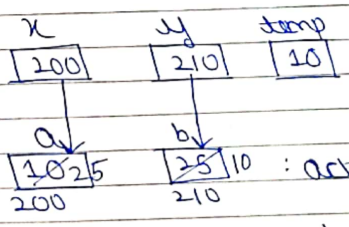
∴ a=10, b=25.

- [Changing actual Parameters]
- Value of actual P is passed to formal P, & changing formal P doesn't change actual P.

Call by Address

② swap(&a, &b)

```
void swap(int *x, int *y)
{ int temp;
  temp=*x;
  *x=*y;
  *y=temp;
```



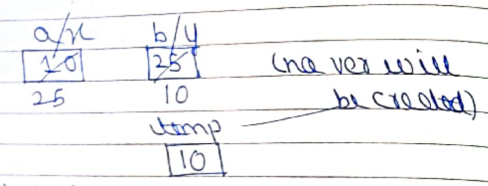
: actual P changed.

∴ a=25, b=10

NOTE: A funcn can access AR of another funcn indirectly using pointers, not directly

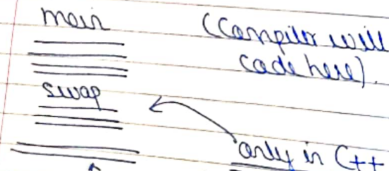
3. Call by Reference

```
void swap (int &x, int &y)
```



→ worked like call by value.
- easy.

→ m/c code



→ complex lengthy funcn, telling compiler

to paste.

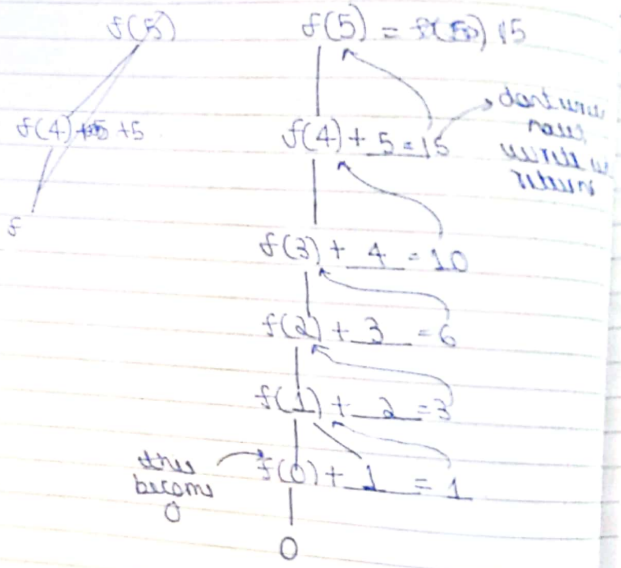
[Inline Function]

→ for small funcn's, you can use this feature.

```
inline int fun(int x)
{
}
main()
{
    fun(a);
}
```

→ copied here.

Q
```
int fun(int n)
{
  if(n==0)
    return 0;
  else
    return fun(n-1)+n;
}
f(5)
```

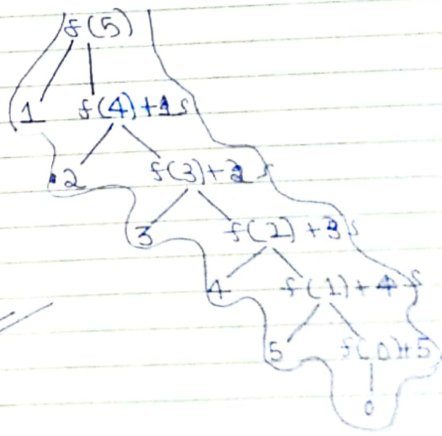


- Recursion relation for finding sum of n natural numbers

Q
```
int fun(int n)
{
  if(n==0)
    return 1;
  else
    return fun(n-1)*n;
}
fun(5);
```
→ Factorial

→ 5!

Q
```
int q=0;
int fun(int n)
{
  if(n==0)
    return 0;
  else
  {
    ++q;
    return fun(n-1)+q;
  }
}
fun(5);
```



SUITS : MINDHUNTER

q

0 1 2 3 4 5

f(5)

f(4) + 5 = 25

f(3) + 5 = 20

f(2) + 5 = 15

f(1) + 5 = 10

0 f(0) + 5 = 5

0

n^2

Q

```
int fun(int n)
{
  static int r=0;
  if(n==0)
  {
    ++r;
    return r;
  }
  else
  {
    ++r;
  } return r+fun(n-1);
}
fun(3);
```

r

0 1 2 3 4

(add done at returning time).

f(3) 16

4 + f(2)

4 + f(1) 8

4 + f(0)

4

```
int f(int n)
{
  static int i=1;
  if(n>=5) return n;
  n=n+i;
  i++;
  return f(n);
}
f(1).
```

Ascending

i 1 2 3 4

n 1 2 3 4

f(1)

n=2 f(2)

n=4 f(4)

n=7 f(7)

⇒ 7

f(1)

n=2 f(2)

n=4 f(4)

n=7 f(7) = 7

Q
```
int f (int n, int K)
{
  if (n==0) return 0;
  else if (n%2)
    return f(n/2, 2*K) + K;
  else
    return f(n/2, 2*K) - K;
}
```

f(20,1)

f(20,1)
|
f(10,2) − 1 = 9
10
|
f(5,4) − 2 = 10
12
|
f(2,8) + 4 = 12
8
|
f(1,16) − 8 = 8
16
|
f(0,32) + 16 = 16
0

⇒ (a) 9 ✓

f(20,1)
|
f(10,2) − 1 = 9
|
f(5,4) − 2 = 10
|
f(2,8) + 4 = 12
|
f(1,16) − 8 = 8
|
0 f(0,32) + 16 = 16

Q
```
int f(int n, int *x)
{
  if (n==0)
    return 0;
  else
  {
    ++(*x);
    return *x + f(n-1, x);
  }
}

void main()
{
  int m=5;
  printf (f(3, &m));
}
```

m
5
200

f(3,200)

6 — +f(2,200)

7 — +f(1,200)

8 — +f(0,200)

8×3=24 ✓

Q int f(int n)
{
static int r=0;
if(n<=0) return 1;
if(n>3)
{
r=n;
return f(n-2)+2;
}
return f(n-1)+r;
}
f(5)

x
~~0~~ 5

f(5)
|
f(3)+2 = 18
|
f(2) + 5 = 16
|
f(1) + 5 = 11
|
1 f(0) + 5 = 6 ✓

Q int f(int &x, int c)
{
c=c-1;
if(c==0)
return 1;
x=x+1;
return f(x,c) * x;
}
main()
{
int p=5;
f(p,p);
}

(Head) recursive

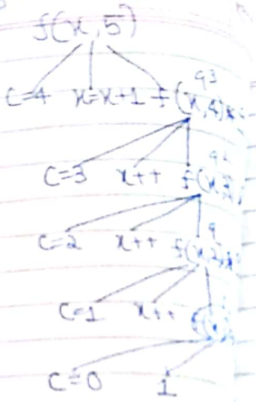
P/x c
5 5

c=4, x=6,
f(5,5)
|
f(6,4) * ___
|
f(7,3) * ___
|
f(8,2) * 720
|
f(9,1) * 90
|
1 f(10,0) * 10

Call by ref:

P/x
5 | 6789

x = p



94 Ans

```
void foo(int n, int sum)
{
  int k=0, j=0;
  if(n==0) return;
  k = n%10;        j = n/10;
  sum = sum + k;
  foo(j, sum);
  printf("%d", k);
}

main()
{
  int a=2048, sum=0;
  foo(a, sum);
  printf("%d", sum);
}
```

K=204, S=204

75

foo(2048,0)

K=0 j=0 K=204 S=204

foo(2048,0)

foo(0,204)

K = 2048 8
j = 204
S = 8

⇒

foo(2048,0)

foo(204,8)

K = 4,
j = 20
S = 12

foo(20,28)

K = 0,
j = 2
S = 30

foo(2, 30)

K = 2
j = 0
S = 32

foo(0,32)

32

sum → 10

a) 2048, 0

Q
```
void gst(int n)
{
  if(n<1)
    return;
  gst(n-1);
  gst(n-3);
  printf("%d", n);
}
```

Soln. gst(6)

gst(6)
gst(5)
gst(4) X

⇒

n	0	1	2	3	4	5	6
gst(n)	1	3	5	7	11		25
				7	9	11	13

⇒ gst(1)

gst(0) gst(-2)

⇒ gst(2)

gst(1) gst(-1)

gst(0) gst(-2)

gst(3)

gst(2) gst(0)

5 1

gst(5)

gst(4)

gst(3) gst(1)

Q
```
int fun(int n)
{
  int x=1, K;
  if(n==1) return x;
  for(K=1; K<n; K++)
  {
    x = x + fun(K) * fun(n-K);
  }
  return x;
}
```
fun(5).

Soln.

x=1+

n	0	1	2	3	4	5
fun(n)		1	2			

f(2)

K=1, 1<2.

x=

2 → 1 loop

x = 1 + f(1) * f(1).

3 → 2 loop

K=1

x = 1 + f(1) * f(2) x = 3

3 + f(2) * f(1) 2 x

n	1	2	3	4	5
f(n)	1	2	5	15	51

- n=2
  x = 1 + f(1) * f(1)

- n=3
  x = x + f(1) * f(2) + f(2) * f(1)
  (1) (2) (2) (1)

- n=4
  x = x + f(1) * f(3) + f(2) * f(2)
  + f(3) * f(1)

  1 + 5 + 4 + 5

- n=5
  x = x + f(1)f(4) + f(2)f(3) + f(3)f(2)
  + f(4)f(1) + f(5)

  ⇒ 1 + 15 + 10 + 10 + 15 = 51

```
int f(int *a, int n)
{
  if (n<=0) return 0;
  else if (*a % 2 == 0)
    return *a + f(a+1, n-1);
  else
    return *a - f(a+1, n-1);
}
```

```
main()
{
  int a[] = {12, 7, 13, 4, 11, 16};
  pf(f(a,6));
}
```

f(a,6)
12 + f(a+1,5)
⇒ 12 + (7 - (13 - (4 + (11 - (16 + 0)))))
⇒ 15

f(a,6)
12 + f(a+1,5)
7 - f(a+1,4)
13 - f(a+1,3)
4 + f(a+1,2)
11 - f(a+1,1)
6 + f(a+1,0)

Q
```
int fun(int n)
{
  if (n>100)
    return n-10;
  else
    return fun(fun(n+11));
}
fun(95);
```

Soln:

fun(95)
|
fun(96)
|
fun(97)
|
fun(98)
|
fun(99)

(92)

Q
```
void f(int *p, int *q)
{
  p=q;
  *p=2;
}
main()
{int i=0, j=1;
{
  f(&i, &j);
  pf(i,j);
}
```

→ 0, 2

Q
```
int *f()
{
  int x=10;
  return &x;
}
```

→ ∵ when function ends AR deleted ∴ X

```
int *g()
{
  int *p;
  *p=10;
  return p;
}
```

p → ?

```
int *h()
{
  int *q=new int;
  *q=10;
  return q;
}
```

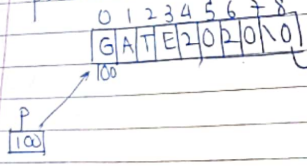
q → 200 → 10 in heap.

17/8/19 Saturday

Lecture 3

Q. `char x='A', y='B'; printf("%d", x-y);` ⇒ -1

Q. `char *p="GATE2020"; printf("%s", p); printf("%s", p+p[3]-p[1]);`



⇒ C/C++.

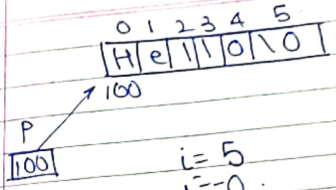
A - 65
B
C
D
E - 69

o/p: GATE2020 , 2020

100 + 'E' - 'A'
104

- printf stops when it's find null.

Q. `char *p="Hello"; char Q[10]; i=strlen(p); j=0; for(; i>=0; i--, j++) Q[j]=p[i]; Q[j]='\0'; printf("%s", Q);`



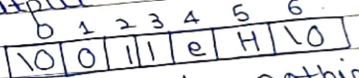
i = 5
j = 0.

Q. i = 0

Q[0] = p[5] , Q[1] = p[4] .

No output.

Q:



∵ 1st letter is NULL, nothing is printed.

Q. `char *p="GATE2020";`

→ literals occupy memory

[String literals go into Code Section.]

- Not in Stack/Heap.

`char Q[]="Hello";`

∵ String literal, goes into Code Section, & its copy goes into Q. array.

char *p = "Hello";

Q for(i=strlen(P), j=0; i>j; i--, j++)
    swap(p[i], p[j]);

printf("%s", p);

→ Prints nothing.

TOPIC: → Next

DATA STRUCTURES

1) ARRAYS

size

10 int A[10];
4 int A[] = {5, 9, 6, 3};
10 int A[10] = {5, 2, 4}; } Stack

int *A = new int[10]; } Heap.

→ pointer in Stack.

- int A[5] = {2, 8, 6, 9, 3}

	0	1	2	3	4
A	2	8	6	9	3
	200/1	2/3	4/5	6/7	8/9

L_0

A[3] = 15; → how compiler refers to location & writes add in m/c.

- in order to access any loc, we need L_0.

A[3] = 15;

→ Address(A[3]) = $L_0 + (3 * 2)$

comp. writes this

* $\boxed{\text{Address}(A[i]) = L_0 + i * w}$

compiler uses this

writes add 200 at that time (Data Binding) [Logical Address]

Size of DT

L_0 : Base Address
i : Index
w : Size of data type used.

In C : Array Starts from 0. (Why?)

$\text{Address}(A[i]) = L_0 + (i-1) * w$;

↳ 1 opⁿ increased ↑.
∴ slower.

Prev Lang:

var A[-3 ... 3] / [5 .. 10]

$\text{Add}(A[i]) = L_0 + (i+3) * w$;

2) 2D ARRAYS

int A[3][4];
m n

A

	0	1	2	3
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{10}	a_{11}	a_{12}	a_{13}
2	a_{20}	a_{21}	a_{22}	a_{23}

A[1][2] = 15;

∵ MM is linear.

- A 2D Array is created ↴

A

0	1	2	3	4	5	6	7	8	9	10	11
a_{00}	a_{01}	a_{02}	a_{03}	a_{10}	a_{11}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{23}
200/1	2/3	4/5	6/7	8/9	10/11	12/13	14/15	16/17	18/19	20/21	22/23
row 0				row 1				row 2			

L_0

compiler access it in 1D form

∴ Mapping from 2D → 1D:

1. Row Major Mapping (Lexicographic Rep)
2. Column Major Mapping.

Ans) Add(A[2][1]) = $L_0 + [2*4+1]*2$

row in 2/16/H.

Row Major ↴

- $\boxed{\text{Add}(A[i][j]) = L_0 + [i*n + j]*w}$

no. of cols.

Dim of Array : m * n

- $\boxed{\text{Add}(A[i][j]) = L_0 + [j*m + i]*w}$

↑ Column Major.

* Both are equally efficient.
  ↳ just see the operations.

NOTE: Row Major is used in C language.

Q 3 Matrices → A, B, C (size = n×n)
(stored in 2D)

A = [ ] B = [ ] C = [ ]

C = A × B (is compatible too).

Which comb is efficient?
(a) A - CM, B - RM
(b) A - RM, B - CM
(c) Both RowM.
(d) ~~Both CM.~~

↳ independent of any representation.

* Be it in cache/MM; efficiency is same.

Q → which formula will be better?
↳ Both formula equally efficient.
(∵ Same operations).

• Var A[1...3][1....4] int. → start from 1.

Add[A[i][j]] = $L_0 + [(i-1)*n + (j-1)] * w$.

• Var A[1.....10][1.....15]
$L_0 = 100$
$w = 1$.
Add(A[i][j]) = $L_0 + [(i-1)*15 + (j-1)]$
$\Rightarrow L_0 + [(i-1)15 + (j-1)]$
$L_0 + 15i - 15 + j - 1$
$84 + 15i + j$ (C)

Q fun1(int *p) → can point to int var / array of integers (mostly used).
{
==
}

fun2(int p[]) → this is also a pointer but points to array.
{
==
}

2 ways.

void main()
{
int A[10];
==
}

[Sep new array not created]

void main()
{
int A[5][6];
==
}

if want to send 2D Array:

```
void main()
{
  int A[5][6];
}

fun2(int p[][6])
{
}

fun1(int *p[])
{
}
```

(total cols 6) can access any of col no. 6. (due to form)

array of pointers.

- if 3 dim: 1st can be left, rest you need to write.

* 4D Array

int A[D1][D2][D3][D4];

$$\text{Addr}(A[i_1][i_2][i_3][i_4]) = L_0 + [i_1 * D_2 * D_3 * D_4 + i_2 * D_3 * D_4 + i_3 * D_4 + i_4] * w$$

Row Major

- A[i][j]
  row → (left to R)
  col ← (R to L)

$$\text{Add}(A[i_1][i_2][i_3][i_4]) = L_0 + [i_4 * D_3 * D_2 * D_1 + i_3 * D_2 * D_1 + i_2 * D_1 + i_1] * w$$

Column Major.

- Mul operation was costly (∵ successive addition).

- 4D Array: $\frac{n(n-1)}{2} \Rightarrow O(n^2)$.
  (No of mul).

For nD Array → $O(n^2)$
↳ Mult. taking n^2 time

- $L_0 + [i_1 * D_2 * D_3 * D_4 + i_2 * D_3 * D_4 + i_3 * D_4 + i_4] * w$

$$L_0 + \sum_{p=1}^{n} \left[i_p * \prod_{q=p+1}^{n} D_q \right] * w$$

Row M. column formula.

* 3D Array

1st dim not used in formula

Type A[l][m][n];

Add (A[i][j][K]) =

$L_0 + (i*m*n + j*n + K)*w$

Q ? X[?][?][?] → DT & dim unknown

[i j K].

32 8

$t_0 = i * 1024$
$t_1 = j * 32$
$t_2 = K * 4$
$t_3 = t_0 + t_1$
$t_4 = t_3 + t_2$.
$t_5 = X[t_4]$

CD Q

⇒ $t_4 = t_0 + t_1 + K * 4$ ✓

$t_4 \Rightarrow i*1024 + j*32 + K*4$

$\Rightarrow t_4 \Rightarrow (i * \overset{256}{\cancel{32}} + j*8 + K)\,4$

A[2][4][8]

int A[ ][4][8]

32

X[ ][ ][ ]
l m n

$(i \cdot mn + j \cdot n + K)w$

32 8 4

w = 4.

$j * \frac{n*w}{32}$ n = 8.

$i * \frac{m*n*w}{32}$ ⇒ m = 32.

A X[ ][ ][ ]
l m = 32 n = 8 w = 4.

I you can't find.

* HORNER'S RULE

$f(x) = 4x^3 + 5x^2 + 6x + 9$. deg = 3.

$f(2) = 4*x*x*x + 5*x*x + 6*x + 9$

⇒ $O(n^2)$ $1+2+3 = \frac{n(n+1)}{2}$

$x^n \Rightarrow 1+2+3 \ldots\ldots n = \frac{n(n+1)}{2} = O(n^2)$

No. of → [multiplications]

$f(x) = 9 + 6x + 5x^2 + 4x^3$ (Arrange)

$9 + x*[6 + 5x + 4x^2]$

$9 + x*[6 + x*[5 + 4*x]]$

↑ ↑ ↑

$O(n)$ = 3 mul only.

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

→ Expression rforming to reduce no. of multiplications.

$L_0 + [i_1 * D_2 * D_3 + i_2 * D_3 + i_3] * w$

$(i_3 + i_2 * D_3 + i_1 * D_2 * D_3) * w$

$\Rightarrow (i_3 + D_3 * (i_2 + i_1 * D_2)) * w$

[Can Apply Horner's rule, on row M & column M formulae].

Q $f(x) = 9x^4 + 3x^2 + x$

$f(x) = x + 3x^2 + 9x^4$

$= x(1 + 3x + 9x^3)$

$\Rightarrow x*(1 + x*(3 + 9x^2))$

$\Rightarrow x*[1 + x*(3 + 9*x*x)]$

$\Rightarrow$ 4 multiplications.

If n-degree polynomial → No. of multiplications : n

Q $f(x) = x^5 + 2x^3 + x^2 + 1$

$f(x) = 1 + x^2 + 2x^3 + x^5$

$\Rightarrow 1 + x^2(1 + 2x + x^3)$

$= 1 + x^2(1 + x(2 + x^2))$

$= 1 + x*x*(1 + x*(2 + x*x))$

$\Rightarrow$ 4 Multiplications

Note: If the highest degree term has coefficient as 1, you need 1 multiplication less. ∴ 4

Q $f(x) = 6x^7 + 2x^2 + 4$: 7

$f(x) = x^9 + 4x^{11} + 6$: 11

Q $f(x) = 6x^5 + 4x^3 + 9x + 1$: 5 mul

$f(x) = 1 + 9x + 4x^3 + 6x^5$

$\Rightarrow 1 + x[9 + 4x^2 + 6x^4]$

$\Rightarrow 1 + x[9 + x^2(4 + 6x^2)]$

$\Rightarrow 1 + x*[9 + x*x*(4 + 6*x*x)]$

let $x*x$. $t = x*x$

$\Rightarrow 1 + x*[9 + t*[4 + 6*t]]$

4 multiplications

(If temporary variable given).

Q. $f(x) = x^5 + 2x^3 + x^2 + 1$ ⟶ 4

f()

$1 + x^2 * (1 + x * (2 + x^2))$

↳ 3 multiplications.

$t = x * x$

$1 + t * (1 + x * (2 + t))$ } 3

Q. $4x^3 + 2x^2 + 6x$ ⟶ 3

How many mul if t is given?

$6x + 2x^2 + 4x^3$

$x[6 + 2x + 4x^2]$

$x[6 + x[2 + 4x]]$ ⇒ 3

Q. $2x^7 + 4x^5 + 2x^3 + 5x$ ⟶ 7

if t is given ⟶ 5

$5x + 2x^3 + 4x^5 + 2x^7$

$x[5 + 2x^2 + 4x^4 + 2x^6]$

$x * [5 + x^2[2 + 4x^2 + 2x^4]]$

$x[5 + x^2[2 + x^2[4 + 2x^2]]]$

$x * [5 + t * [2 + tx[4 + 2t]]]$

Q. $x^5 + 9x^3 + 2x + 7$ ⟶ 4.

If t given. ⟶ 3

$7 + 2x + 9x^3 + x^5$

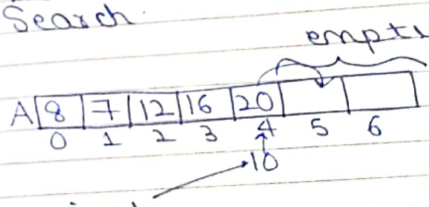
$7 + x[2 + 9x^2 + x^4]$

$7 + x[2 + x^2[9 + x^2]]$

$= 7 + x * [2 + t * [9 + t]]$

Operations on Array

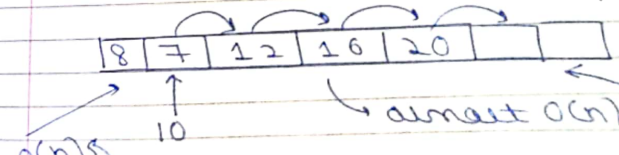
1. Insert
2. Delete
3. Search



insert

replace ⟶ replace existing one

set ⟶ replace (want new element at that place)



O(n)

(If you want to insert at the beginning)

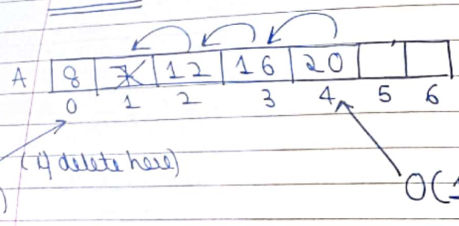
Append = Inserting at last O(1)
Insert = O(1) to O(n).

Insertion = O(n) (worst case).

- (Replace & Set will always take constant time)

delete.

length = 5.



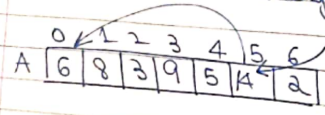
O(n) (if delete here)

O(1).

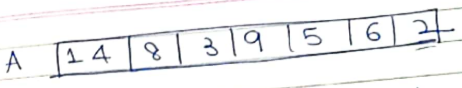
To delete last element: length ⇒ 4.

logically deleted, not physically.

* Search → Linear Search — O(n)
→ Binary Search — O(log n)

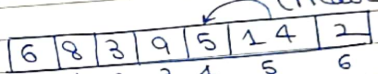


after 6 comp: 14 ✓ found (keep it in beginning)

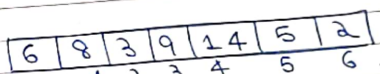


① Move to Front / move to head.
→ in order to improve Linear Search.

② (move/swap prev element)



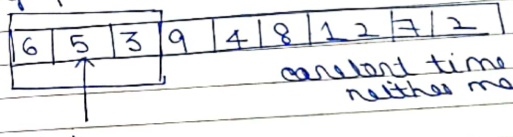
if 14 Searched



: Transposition

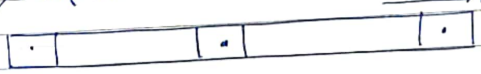
GATE Q.

- Array of unsorted elements (distinct)

① 

constant time (to set neither max/min).

② Sorted Array
whether there is an element coming, half or more than half of array



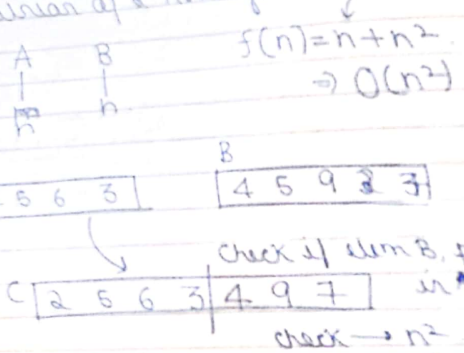
log n

- if 1 element is repeating half, or more, find that element? → O(1) [Check mid, [left, right]

Checking if an element exists, which repeats half or more time?

1. Union — $O(n^2)$ $O(m \times n)$
2. Intersection — $O(n^2)$
3. Difference — $O(n^2)$
4. contains — $O(n)$ (Linear Search)

- Union of 2 Arrays



- If Sorted (Union/Intersection)

$O(m+n) / O(2n) = O(n)$.

If Sorted : $O(\log n)$

* Structure & Union

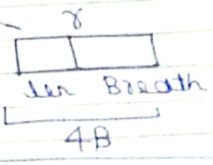
```
struct Rectangle
{
    int length;
    int breadth;
};
```

struct becomes class in C++, (but methods present)

4B (Size)

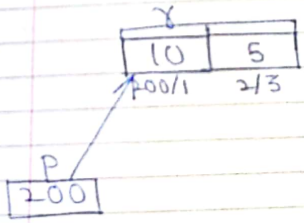
definition/design of structure (not actual)

```
main()
{
    struct Rectangle r;
    r.length = 10;
    r.breadth = 5;
```



. operator used to access members

① struct Rectangle *p;
② struct Rectangle r = {10,5};



- p = &r;
- *p.length = 20; ✗

② ① Syntax error (: p not a structure)

- (*p).length = 20;

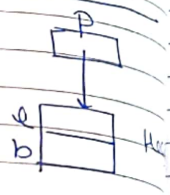
p → length = 20;

→ if p is a structure pointer

- arrow used for dereferencing.

* ⟶ [it will have structure type pointer only]

```
main()
{
struct Rectangle *p;
```



C++ `p = new Rectangle;`

C `p = (struct Rectangle *) malloc (sizeof(struct Rectangle));`

```
p ⟶ length = 15;
p ⟶ breadth = 5;
```

NOTE: ✓ `void x;` ✗

`void *p;`

↳ pointer will be declared.
- it can hold add of any data type.
[Generic pointer]

malloc just allocates memory, returns just void pointer.
we need to type cast.

`void *p;`
`int x = 10;`
`p = &x;`
`printf(*p);` ✗

↳ after going, how many bytes read?
∴ Void Pointer can't be dereferenced.

↳ can type cast into some other pointer.

✓ `++p ⟶ a;` (data will be incremented)

✓ `p++ ⟶ a;`

↳ take the value, & move the pointer [L → R associative]

-
```
Struct Rectangle
{
  int length;
  int breadth;
} r;
```
↳ Structure variable.

✓ `typedef Struct Test` ↳ alias / you can give new name.
```
{
  ==
} TEST;
main()
{ TEST x;
```

```
typedef int marks;
main()
{
marks m1, m2, m3;
```

int ka dosra naam marks.

```
typedef int m;
main()
{
m n,o,p,q;
```

red: reduces.

```
main()
{
struct {int a,b;} x;
x.a=10;
x.b=5;
}
```

anonymous struct (nameless).

```
struct A
{
int a;
float b;
struct B
{
int x;
char y;
};
};
```

→ 9 Bytes.

→ 6 Bytes

→ if declared : 3 bytes

→ 9 Bytes

```
struct A m;
m.c.x=10;
```

[nested structures]

```
struct Test
{
int data;
struct Test t;
};
```

→ 2B

→ definition not allowed.

[- Struct can't have a var of its own kind.

```
struct Test
{
int data;
struct Test *p;
};
```

→ 2 + 2 = 4B

→ LL

→ Self referential Structure [used for creating LL].

UNION

```
main()
{
pf("menu");
pf("1. int");
pf("2. float");
pf("3. double");
```

```
int a, ch;
float b;
double c;
char d;
```

X(?)

```
union Data
{ int a;
float b;
double c;
char d;
} d;
```

8B.

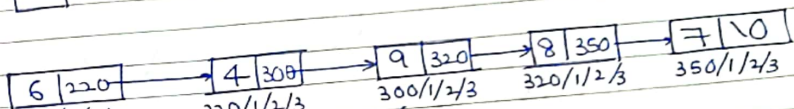
```
pf("4. char");
pf("Enter your choice");
scanf("%d", &ch);
```

Why main?
used in file handling/networking/
System Programming.

19/9/19
Sunday

Lecture 4

first



2) LINKED LIST

Which problem can be solved by LL?

Node


```
Struct node
{
  int data;      — 2
  struct node * next;  } 4B
                 2
}
```

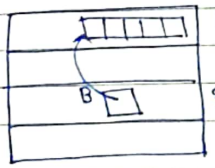
```
main()
{
  int A[10];
```

used by user.

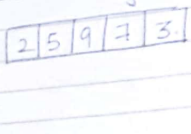
Size must be mentioned in code (not in run time)

- programmer is not confirmed how much space array will take in run time.

```
main()
{ int A[10]; ✓
  int *B, n;
  scanf(&n);
  B = new int[n];
  int c[n]; X
      [not valid]. ✓
```

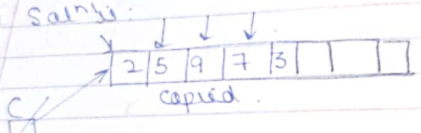


- use fixed array size, which is in Heap.



- Now, we want to extend array. X (same)

① Solution:



copied. (different Array)

* In Java, ArrayList does this. (self managed Array).

eg: Notepad → limit: 5000 chars.

: char A[5000];

in MS word → maybe asking user, how many pgs you need to type.

②

But actually, its dynamic.

Completely dynamic data structure.

↳ linked list can grow & shrink in size.

- Speed of Array/LL depends on operation use.
- LL uses more space than Array.
- LL → indirect.
- LL → only in heap. Array → Stack/heap.
  ↳ meaningless in Stack (not dynamic)
- Both array & LL is efficient.

LINKED List

- collection of nodes where each nodes contain data & pointer to next node.

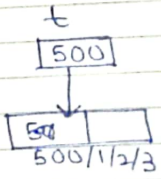
```
struct Node
{
  int data;
  struct Node * next;
};
```

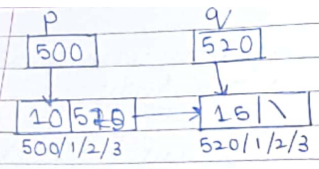
2 / 2 / 4B

↳ if we dereference, we can access 4B.

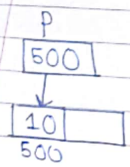
[DT of pointer: node]

```
struct Node *t;
t = new Node;          //C++
t = (struct Node *)    //C
      malloc
      (size of(struct node));
t -> data = 10;
t -> next = NULL;
```





- x = p → data; 10
- q = p → next; 520
- p = p → next; // p moves forward



* if (P != NULL) ≡ if (P)

if (P == NULL) ≡ if (!P)
T T
∴ 0 = F.

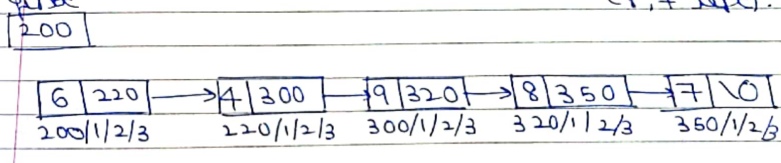
- if (p → next != NULL) : not last node.
  if (p → next == NULL) : last node
- q = p → next ? p → next : NULL;
  ↳ if p ka next node hai q will point on next node.

≡ if (p → next != NULL)
q = p → next;
else
q = NULL; ✓

✓ q = p ? p → next : p;
↳ if p is pointing on ~~next~~ node, let q point on next node.

if (p != NULL)
q = p → next;
else
q = p; ✓

- pf(first → data); = 6
- pf(first → next → data); = 4
- pf(first → next → next); = 300
- pf(first → next → next → next = 9 wale ka next NULL (9, 7 left).



- q = first → next → next → next; (q will point on 8).

How to display?

```
p = first;
printf("%d", p->data);
p = p->next;
while (p != NULL)
{
  printf("%d", p->data);
  p = p->next;
}
```

recursive

```
void display (struct node *p)
{
  if (p)
  {
    pf(p->data);
    display(p->next);
  }
}
```

→ 6 recursive calls : n+1
Stack Size : n+1 (Max AR at a time)

: reverse

```
void display (struct node *p)
{
  if (p)
  { display(p->next);
  } pf(p->data);
}
```

(with loop, we cant display the reverse)

```
count = 0;
p = first;
while (p != NULL)        // traversing the LL
{ count++;               // most used.
  p = p->next;
}
return count;
```

→ counting no of nodes - first node pt, count → 1.

```
int fun(struct node * p)
{
  if (p == NULL)
    return 0;
  else
    return fun(p->next) + 1;
}
```

→ count Nodes.
→ returning time.
→ counting starts from last

```
return 1 + fun(p->next);
```

→ also returning time.

```
int fun(struct node *p)
{
  int x=0;
  if(!p) return 0;
  x=fun(p->next);
  return ++x;
}
```
→ Count.

```
int fun(struct node *p)
{
  static int m=0;
  if(p)
  {
    ++m;
    fun(p->next);
  }
  return m;
}
```

m
0 ~~2743~~

fun(200)
fun(220)
fun(300)
fun(320)
fun(350)
fun(NULL)

⇒ return 5

```
int fun(struct node *p)
{
  if(!p) return 0;
  return fun(p->next)+p->data;
}
```
→ Sum of all nodes in LL

```
int fun(struct node *p)
{
  int x=0;
  if(p)
  {
    x=fun(p->next);
    return p->data>x?p->data:x;
  }
  return 0
}
```
→ max

```
int fun(struct node *p)
{
  if(p)
  {
    if(p->next && p->data>p->next->data)
      return 0;
    return fun(p->next);
  }
  return 1;
}
```
→ checking at call time
→ not Sorted
1 → S

- checks if LL is sorted or not.

```
struct node *search(struct node *p,
                    int Key)
{
  if (p)
  {
    if (Key == p->data)
      return p;
    return search(p->next, Key);
  }
  return 0;
}
```

call time pe check

return kar rahe

if not max 6 calls.
∴ [1,6] O(n+1)

```
struct node * search(struct node *p,
                     int Key)
{
  while (p)
  {
    if (Key == p->data);
    {
      return p;
    }
    p = p->next;
  }
  return NULL;
}
```

iterative version of search.

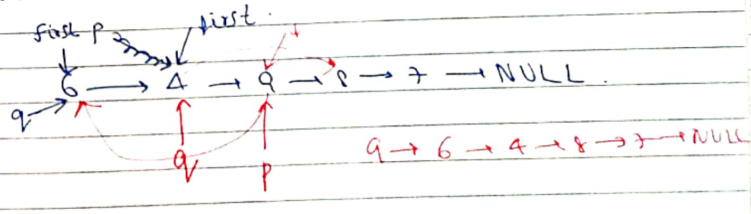
```
while (P != NULL)
{
  q = NULL;
  p = first;
  while (P != NULL)
  {
    q = p;
    p = p->next;
  }
```

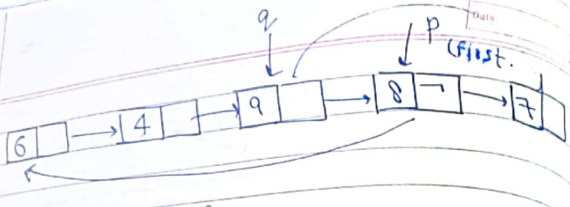
p: iterator.
q: tail pointer

abb after exec, p will be at last.

```
q = NULL;
while (p)
{
  if (Key == p->data)
  {
    q->next = p->next;
    p->next = first;
    first = p;
    return first;
  }
  q = p;
  p = p->next;
}
return NULL;
```

Move to first.



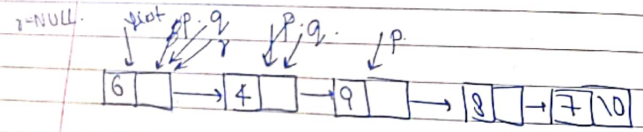


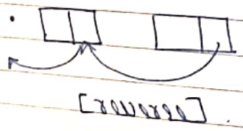
Key = 8 ;

8 → 6 → 4 → 9 → 7 . ✓

```
q = r = NULL;
p = first;
while (p != NULL)
{
    r = q;
    q = p;
    p = p->next;
}
```

sliding pointers





[reverse].

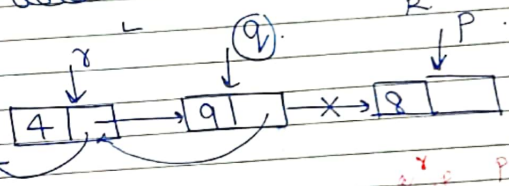
- Reverse : copy LL into array & reverse
  add into LL.
  ↳ data reverse.

Reverse the Links

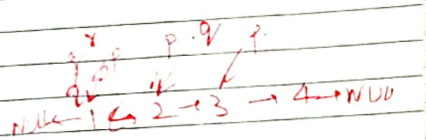
[Link movement is faster & cheaper than data movement].

∵ Pointers of same size = 2B.

→ But data movement, depends on size of data.

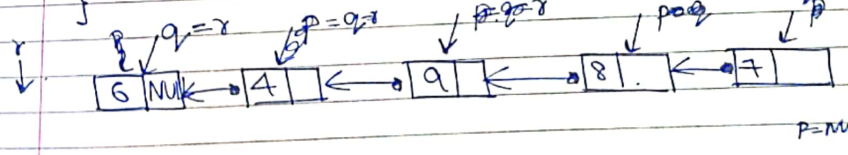


q → next = r




```
q = r = NULL;
p = first;
while (p != NULL)
{
    r = q;
    q = p;
    p = p->next;
    q->next = r;
}
```

} Sliding.



first = q ;

HW

Q
```
void Reverse (node *prev, node *curr)
{
  if (curr)
  {
    reverse (curr, curr->next);
    curr->next = prev;
  }
  else
    first = prev;
}
```

reverse (NULL, first).

→ 2 → 3 → 4 → NULL

4 → 3 → 2 → NULL

* INSERT

→ 1 link modified (Time constant)

1) Insert in before Head

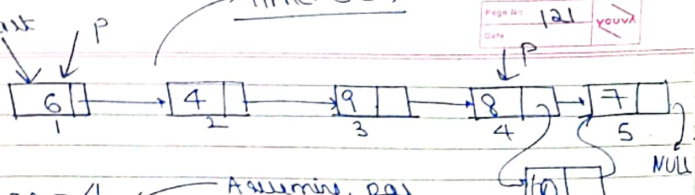
Head

10 → 6 → 4 → 9 → 8 → 7 → NULL

```
node *t = new node;
t->data = 10
t->next = first;
first = t;
```

2) Insert node at given position (after a given pos)

Time: O(n)



pos = 4 ← Assuming, pos is valid.

```
node *p;
node *t = new node;
t->data = x;
p = first;
for(i=0; i<pos-1; i++)
{
  p = p->next;
}
t->next = p->next;
p->next = t;
```

<3.

→ 2 links modified.

[after last, possible with this code]

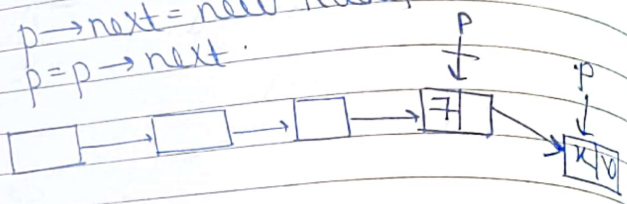
- Node ka order kaise pointer: link.
  extra pointer used for accessing LL: pointer

→ if pos^n : Last O(n)
→ if pos^n after first node = constant time
  if pos^n after last node = O(n)

Q How many pointers? → 2. (p & t).

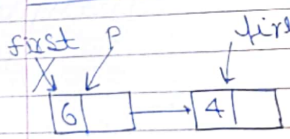
(2) Appending node (last node) → only 1 pointer enough.

```
p->next = new Node;
p = p->next;
```




```
p->data = x;
p->next = NULL;
```

* Delete: (1) first node



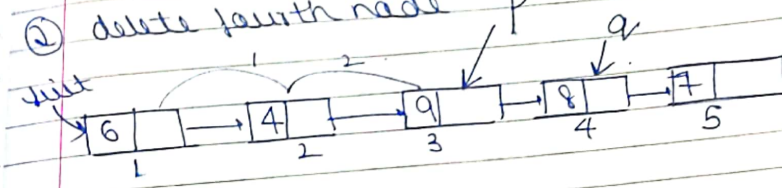

```
node *p = first;
first = first->next;
x = p->data;
free(p);          // delete node from heap
```

[0 link modified, 1 extra pointer, constant time]

[if not using extra pointer, can't release the node] → memory leak.

Page No. 123

(2) delete fourth node



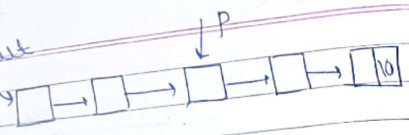
pos = 4

```
node *p = first;
for(i=0; i<pos-2; i++)
    p = p->next;
p->next = p->next->next;        (1)
node *q = p->next;
p->next = q->next;              (2)
x = q->data;
delete q;
```

[1 link modified
O(n) time → min → 2nd node, max → last node
2 extra pointer]

```
for(i=0; i<pos-1; i++)
{
    q = p;
    p = p->next;
}
q->next = p->next;
x = p->data;
delete(p);
```

Try: p exists or not



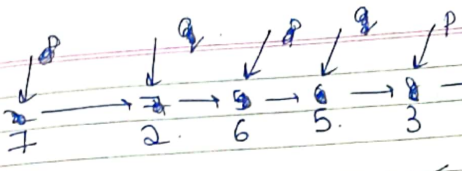
In SLL, if we want to delete p.
we need O(n).
∵ need to go till the previous element

→ next node delete : constant time.

if swap data, with the next element, (data moving) → O(1)
[But data can't be moved in Q.]

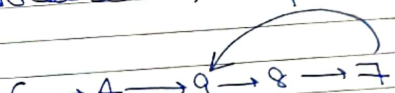
- Linear Search (LL) = O(n).
  Binary Search (LL) → O(n log n)
  → Possible but inefficient
  [n/2] = n. going to middle

```
p = first;
q = p → next;
while( p && q)
{
  t = p → data;
  p → data = q → data;
  q → data = t;
  p = q → next;
  q = p ? p → next : p;
}
```

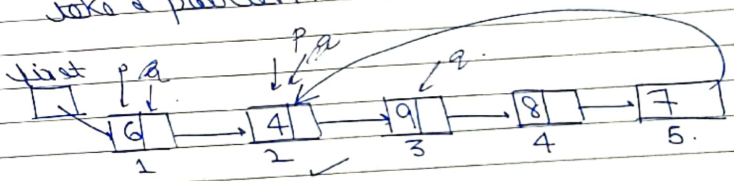



t = 5 t = 8.

If last node pointing to same node on linked list. ⇒ loop. [need not be a head node]



To detect cycle in a Graph, take 2 pointers.



q one time
p two time.

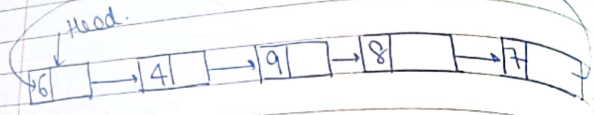
- p = q = first.

```
do
{
  p = p → next;
  q = q → next;
  q = q ? q → next : q;
} while (p && q && p != q);
if (p == q)
  printf("loop");
else
  printf("linear");
```

O(n).
5n
6n
500n ⇒ O(n)
not n²

if next step NULL.

* Circular Linked List → Circular Array



- nothing first/last in CLL.

```
p = Head
print(p->data);
```

* Display

```
p = Head;
while (p->next != Head)
{
  pf(p->data);      → one node left.
  p = p->next;
}
```

```
do{
p = Head;
  pf(p->data);
  p = p->next;
} while(p != Head);
```

✓

```
count = 0;
p = head
while (count == 0 && p != head)
{
  p = p->next;
  count++;
}
```

* Insertion

① after given position

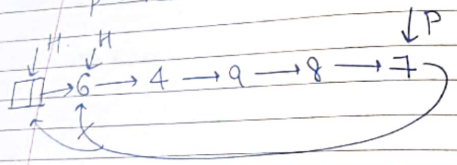
(Same as SLL)

② inserting before head

```
node *p = head;
while (p->next != Head)
{
  p = p->next;
}
node *q = new node;      q->data = x
q->next = Head;
p->next = q;
```

→ 1 Pointer.

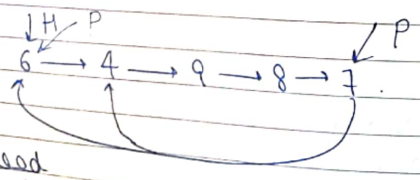
```
p = Head;
while(p → next != Head)
    p = p → next;

Head = new Node;
Head → data = x;
Head → next = p → next;
p → next = Head;
```



* Deletion

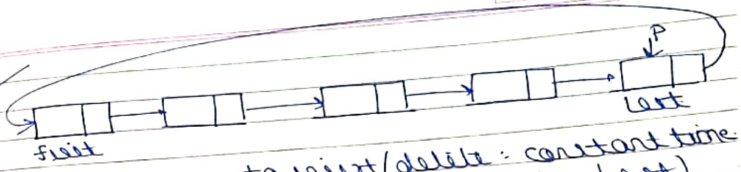
1. delete ~~first~~ any node: Same procedure.

2. delete head node




```
p = Head
while(p → next != Head)
    p = p → next;
p → next = Head → next;
x = Head → data;
delete(Head);
Head = p → next;
```


*



Queue fifo. to insert/delete: constant time. (P should point to Last).

* Doubly linked list → bidirectional Access.
- Cost increases, ∴ flexibility increases

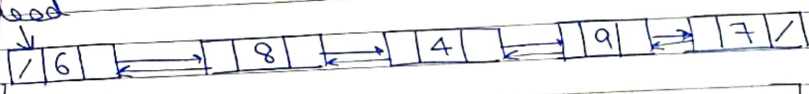
Node

prev	data	next

```
struct node
{
    struct node * prev;
    int data;
    struct node * next;
};
```

} organised.

Head



- in cell, if you want to access context list, there is a window of 2 pointers.

```
p = Head;
while(p != NULL)
{
    pf(p → data);
    p = p → next;
}
```

} traversing same as SLL.

returning no. of nodes.

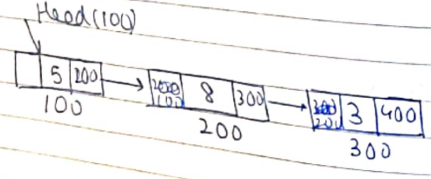
```
count=0;
p=Head;
while(p!=NULL)
{
 count++;
 p=p→next;
}
return count;
```

counting from last.

```
int count(node *p)
{
 if(p==0) return 0;
 return count(p→next)+1;
}
```

```
int count(node *p)
{
 if(p==0)
 { return 0;
 }
 else
 {
 return count(next)+count(prev)+1
 }
}
```

GATE.




```
count(100)
   |
 count
```

Try

* Stack overflow !!

⇒ To avoid this :

```
int count(node *p)
{
 if(p==0)
 { return 0;
 }
 else
 {
 if(flag==0)
 {
  flag=1;
  return count(next)+count(prev)+1;
 }
 }
}
```

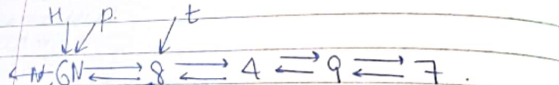
```
struct Node
{
 struct node * prev
 int data;
 int flag;
 struct node
      * next;
}
```

prev	data	flag	next

- You can detect loops in CLL, using flag in nodes.
[Dont assume flag in nodes.]

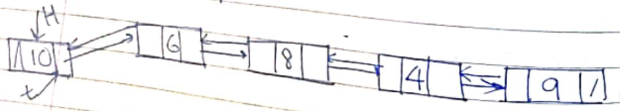
if flag exists → O(n) [No need to take 2 Pointers].

```
p = Head;
while (p)
{
    t = p → next;
    p → next = p → prev;
    p → prev = t;
    p = p → prev;
    if (p → next == NULL)
        Head = p;
}
```



Insert at Start

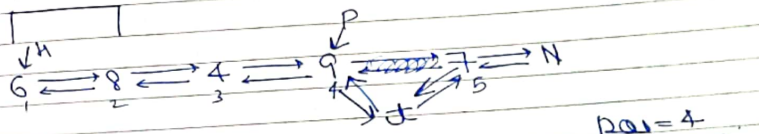
```
t = new Node;
t → data = x;
t → prev = NULL;
t → next = Head;
Head → prev = t;
Head = t;
```



Time = O(1)
Links modified = 3

⇒ Insert after given position

pos = 4.



pos = 4

```
p = Head
for (i = 0; i < pos - 1; i++)
    p = p → next;

t = new node;
t → data = x;
t → next = p → next;
t → prev = p;
p → next = t;
if (t → next)
    t → next → prev = t;
```

} last step

O(n) [No. of links modified = 4]
[3 modified, when not last element]

⇒ Deletion : (HW) → head.
→ any node.

Q If p on any element in DLL, O(1) time to delete.

[Project → Circular DLL]

Most of the time apps we need this

3 ⇄ 4 ⇄ 9 ⇄ 6.

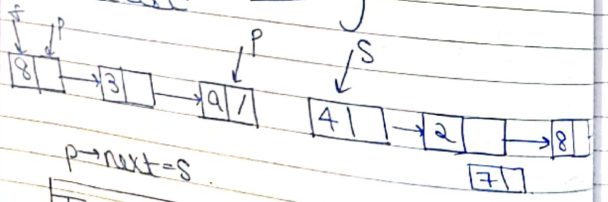
p → prev → next = p → next;

* CLL not used much.

Q Java has built in classes.
(1) CLL class.

- Singly linked list.
- Circular Singly linked list.
- Doubly linked list.
- Circular doubly linked list. O(1)

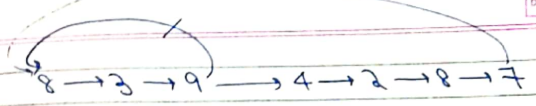
Concatenation O(n)



p → next = s

Time → O(n)

CLL O(2n) = O(n)

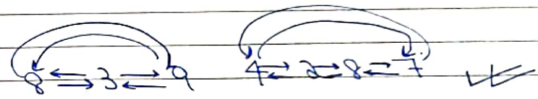


- Scan both.
O(n+n) = O(2n) → O(n).

GATE Q.

Which of the following LL can be concatenated faster than all LLs.

→ all LLs.



- reach last node of C-DLL in constant time.

3) STACK

- ADT (Abstract Data type)

DT:
- Data representation
- Operations on Data

Built in DT → int, float, double, char. (primitive)

(User defined data types = ADT (Class)
- Data repren
- Operations on data.

- ADT concept before OOPS.
  ↳ can be achieved using classes.

* Abstract:
↳ hiding internal details.

- If anyone is using that DT, they dont need to know interior details.

Physical DS:
1. Array
2. LL

Logic DS:
3. Stack
4. Queue
5. Trees
6. Graphs
7. Hash Table

(1 & 2) so for → ADT → implement all these using (1) & (2)

- Logical DS is implemented any one of the physical DS.

eg: furnitures, exam.

(If Stack imp by Array, certain disc).

→ Logical DS increases discipline on how to use the physical data structures.

* STACK

ADT:

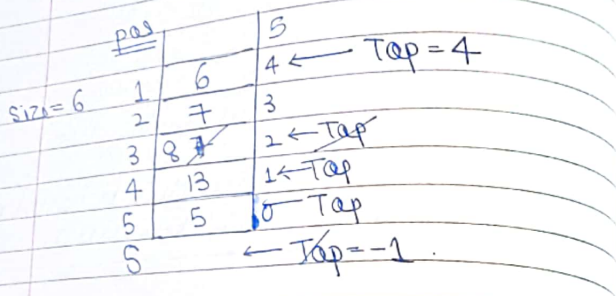
Data: (Abstract. (details not given))
1. Space for Storing elements
2. Top to point an TopMost element.

Operations: (just the definition of Stock.)
1. push(x)
2. pop()
3. peek(position)
4. StackTop()
5. isFull()
6. isEmpty().

(what position, what element?)

- Definition of some datatype is ADT.
- Stack works on LIFO.
- (Elements inserted & deleted from same end)

* Implementation of Stack using Array



- Initial
  top = -1
- empty
  if (top == -1)
- if (top == size - 1) ← Full
- void push(int x) → O(1)

```
{
  if (Top == size - 1)
    pf("Stack overflow");
  else
  {
    top++;
    S[top] = x;
  }
}
```

[Push / Pop both takes constant time]

- int pop() → O(1)

```
{
  int x = 0;
  if (Top == -1)
    pf("Stack underflow");
  else
  {
    x = S[top];
    top--;
  }
  return x;
}
```

- First element ⇒ 6

if Top = 4

position	index
1	4
2	3
3	2
4	1
5	0

index = (Top - Position) + 1

Appⁿ =

- int peek(int position)

```
{
  int x = 0;
  if (top - position + 1 < 0)
    pf("invalid position");
  else
    x = S[top - position + 1];
  return x;
}
```

Q push 10, push 20, push 10, pop, pop, push 10, pop, push 20, push 10, pop, pop, pop.

10
20
10
10
20
10

[10, 20, 10, 10, 20, 10.

* push(Stack)
- stack push (Stack S, int x)
  - Stack
  - element
- Stack pop (Stack S).

[Parameter & RT both Stack].

* pop(pop(push(push(push(St,10),20),15)))

* top1 top2

S	1	8	7	4	3	6	1	1	2	3
	0	1	2	3	4	5	6	7	8	9

↳ using 1 array for implementing 2 Stack

→ ←

top1 = -1 top2 = 10

- Stack 1 : empty — top1 == -1 ✓
- Stack 2 : empty — top2 == size ✓

full — if (top1 == top2 - 1)

Array of size m. A[m]
n Stacks implemented.
T[n] = Array of Top pointers.
B[n+1] = Boundaries ✓

```
for(i=0; i<n; i++)
{
  T[i] = i*m/n - 1;
  B[i] = i*m/n - 1;
}
B[n] = m-1;
```

m = 12
n = 3.

n = 4
5 boundaries.

T[0] = -1. T[1] = m/n - 1
B[0] = -1 B[1] = m/n - 1.

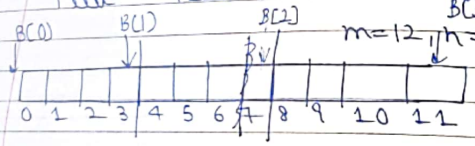
T[0] = stack 1
T[1] = stack 2
T[2] = stack 3

(a) Empty T[i] = B[i-1]
Full T[i] = = T[i+1]

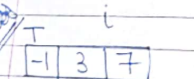
(b) Empty T[i] = = B[i-1]
Full T[i] = = B[i+1]

(c) Empty T[i] = = B[i]
Full T[i] = = B[i+1]
T[0] = B[1]

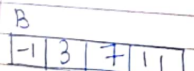
(d) Empty T[i] = T[i-1]
Full T[i] = B[i]



Initially



initial empty position.



L=1
T[1] = = B[2].

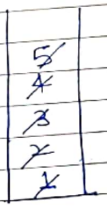
initialisation.

(Boundaries fixed, top can move).
2nd part of Stack 1.
empty condn of Stack 2

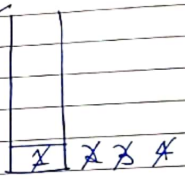
(e) T[i] = B[i];
T[i+1] = B[i+1];

(3) All elements pushed in stack one by one, in some order.
& pop (any order, any no of elements).

1, 2, 3, 4, 5

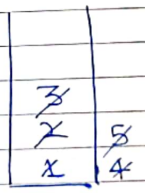


① 5 push ... 5 pop ...
5, 4, 3, 2, 1

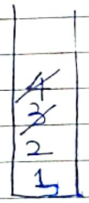


② o/p = 5 push pop
⇒ 1, 2, 3, 4, 5

③ o/p = 3 2 1 5 4



3 2 1 5 4



④ 1 2 5 4 3 → pu, pu,

⑤ 4 4, 3, 5, 1, 2. X (not possible)

12543

5
4
3
2
1

push pop, push pop, 3 push, 3 pop

⑥ 3,5,1,2,4 X

⑦ 4,1,2,3,5 X

⑧ 1,3,5,4,2 ✓

35

4

13542

⑨ 1,4,3,5,2 ✓

1 ✓

* 5! = 120 Arrangements.

No of Valid Permutation = $\frac{2n C_n}{(n+1)}$

(42) $\frac{10C_5}{6}$ $\frac{10!}{5!\times5!}\times\frac{1}{6}$ = $\frac{10\times9\times8\times7\times6\times5!}{5!\times5!}$

* Avg. Lifetime of element in the Stack:

- 1, 2, 3, 4, 5 ..... n.
- every elem is pushed onto Stack & pop everything out.

• push — x time. ✓
• pop — x time.
• delay — y time between every 2 Stack operations.

1, 2, 3, 4, 5

y x,y, x,y

$(x+y)+(x+y)+(x+y)+(x+y)$
$+x$
$5x + 4y + 5x + 4y$
$10x + 8y$.

$2nx + 2(n-1)y$

y + x + y + x + y

⇒ 5 → y ✓
4 → 2x + 3y ✓
3 → 4x + 5y ✓
2 → 6x + 7y
1 → 8x + 9y

$\frac{20x + 25y}{5}$ ⇒ 4x + 5y ✓
⇒ (n−1)x + ny

$2[1+2+3+4]x + [1+3+5+7+9]y$

$2 \cdot \frac{n(n-1)}{2} x + n^2 y$

$\Rightarrow \frac{n(n-1)x + n^2 y}{n}$

$\Rightarrow (n-1)x + ny$ ✓

* Infix to Postfix

→ all Math formula (easy)

Infix: Oprand operator Oprand

eg: a+b

Prefix: opt opnd opnd

+ab

Postfix: opnd opnd opt

ab+

Reverse Polish notation

* Operator	Precedence
+, −	1 (lowest P)
*, /	2
( )	3

- m/c : infix to postfix.

6+8 = 14 → 68+ = 14.

① Infix: a*b
prefix: *ab
postfix: ab*

② Infix: a+b*c → we don't do parentheses, as compiler parenthesis it on the basis of precedence.
postfix:

(a+(b*c)) [Fully Parenthesised, then convert]

⇒ (a+(b*c))

⇒ (a+[bc*])

⇒ abc*+ ← (Postfix) ✓

⇒ (a+(b*c))
⇒ (a+[*bc])
⇒ +a*bc ← (Prefix). ✓

③ a+b+c*d−e.

a+b+[cd*]−e (Assoc: L→R)
⇒ [ab+]+[cd*]−e
⇒ [[ab+][cd*]+]−e

⇒ ab+cd*+e− ✓ (Postfix)

⇒ a+b+c*d-e
⇒ a+b+[*cd]-e
⇒ [+ab]+[*cd]-e
⇒ [+ +ab*cd]-e
⇒ -++ab*cde

④ a*b/c+d-e.

⇒ [ab*]/c+d-e
⇒ [ab*c/]+d-e
⇒ [ab*c/d+]-e
⇒ fab*c/d+e- (Postfix)

· [*ab]/c+d-e.
⇒ [/*abc]+d-e
⇒ [+/*abcd]-e
⇒ -+/*abcde. (Prefix)

⑤ (a+b)*(c-d)/e.

⇒ [ab+]*[cd-]/e
⇒ [ab+cd-*]/e
⇒ ab+cd-*e/ (Postfix)

[+ab]*[-cd]/e
⇒ [*+ab-cd]/e
⇒ /*+ab-cde. (Prefix)

⑥ a+b*c-d/e.

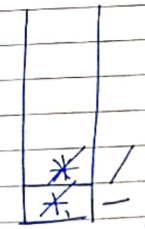
⇒ a+[bc*]-[de/]
⇒ [abc*+]-[de/]
⇒ [abc*+de/- (Postfix).

⇒ a+[*bc]-[
⇒ -+a*bc/de (Prefix)

⑦ a+(b-c)*d/e.

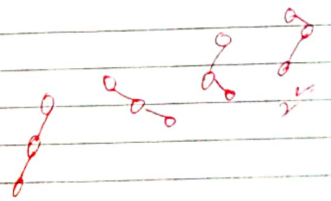
pre: +a/*-bcde
post: abc-d*e/+.

* Infix: a+b*c-d/e. if prec greater pop.



Stack. [Scanning from L→R]

postfix: abc*+de/-



Infix	Stack	Postfix
a	—	a
+	+	a
b	b +	ab
*	*, +	ab
c	*, +	abc
−	−	abc*+
d	−	abc*+d
/	/, −	abc*+d
e	/, −	abc*+de
−	−	abc*+de/-

Steps [Infix → Postfix conversion]

1. If expression has n characters, time req to convert to infix to postfix: O(n).
2. Take infix as String
3. If operator, push in Stack if precedence higher.

Operator Stack

→ If ')' is there, pop all from Stack.

Prefix: right to left

compiler works? it writes m/c code

151

* Evaluation of postfix expression

Infix: 3 + 8 * 2 − 6 / 3 (All single digits)

compiler converts into Postfix & then execute it [generating m/c code]

Postfix: 3 8 2 * + 6 3 / −

[1 Scan only] (to achieve)

pop2 pop1

8 * 2 = 16

x	3
8	16
8	19

3 + 16 = 19

6 / 3 = 2

19 − 2 = 17

3 Scans to achieve result ✓

a + b ^ c − d / e + g / + h * (−K) + (m + n) * p.

( )
| log −
^
*, /
+, −

Postfix	Stack	operation
3	3	
8	3, 8, 3	
2	3, 8, 2, 3	
*	16, 3,	8*2=16
+	19	3+16=19
6	19, 6, 19	
3	19, 6, 19	
/	2, 19	6/3=2
-	17	19-2=17

* a+b+c*d

- Compiler does by infix to postfix & evaluate postfix,

4+3+2*5

43+25*+

4+3=7

+ is carried first.

(We believe * is done first but only prec is higher.

Order of execution can be predicted (depends on compiler)

* Associativity

Associ.	operator	prec
LR	+, -	1
LR	*, /	2
LR	( )	3

(i) same prec, associativity is seen).

[parenthesis based on associat & prec, not execution]

1. ((a+b)+c)+d) $\xrightarrow{L-R}$ ab+c+d+

2. (a=(b=(c=(d=5)))) $\xrightarrow{R-L}$ abcd5====

3. a^b^c $\xrightarrow{R-L}$ abc^^ $(a^b)^c$

4. a^b^c ≡ abc ≡ a**b**c

assignment → left precedence

Unary operators R-L Assoc

- unary minus.
! factorial.
log log

- infix : -a
pre : -a
post : a-

- infix : log a
pre : log a
post : a log

- prefix : a!
pre : !a
post : a!

- $-a * b + \log(m+n)!$

post : [a-] *b + log(m+n)!

a-b* mn+! log +

[a-b*] + log(mn+)!

[a-b*] + (mn+)! log

24/8/19 Saturday

Page No. 155 Date: YOUVA

Lecture 5

Q 35 * + 6 * 42 + * 7 -

7		
64	8	8
7/48	8	8

3+5=8
8*6=48
288-7=281

in evaluation: operand stack.

QUEUE

- works on FIFO.
- elements are inserted from rear end & deleted from front end.
- Queue can be implemented using Array / LL.

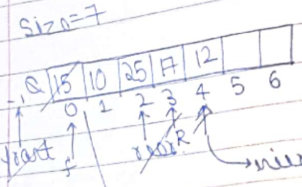
- Queue ADT → implement like class.

Data:
- Space for storing elements
- front - for deletion
- rear - for insertion

Operation
- enqueue (x)
- dequeue ()
- first () → which is the first element.
- last ()
- isEmpty ()
- isFull ()

Queue imp. using Array → 2 Pointers (rear & front)

Size = 7



[till what index we have values in Queue] → rear

we can't reuse holes in Array.

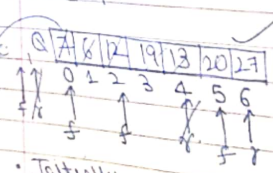
[Not ok] if Shifting: O(n). : Delete. → Insertion = O(1)
(needs to be fast)

using f: Dequeue: O(1)

element: front+1 to rear

front: from where, q begins. (b4 first element)

Size = 7



→ here, min space is there, can use.

- Initially
  front = rear = -1
- empty
  if (front == rear)
- full
  if (rear == size - 1)

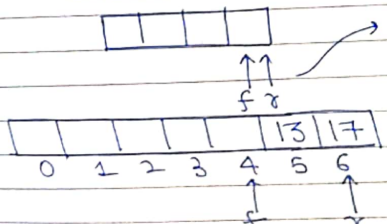
(where front: element not present
(front: 1 b4 element begins).

- void enqueue (int x) → O(1)
{
if (rear == size - 1)
print ("Queue is full");
else
{ rear ++;
Q[rear] = x;
}}

- int dequeue () → O(1)
{
int x = 0;
if (front == rear)
print ("Queue empty");
else
{
front ++;
x = Q[front];
}
return x;
}

→ if returns 0: Queue empty

⇒ Problem using queue with Array



⇒ Array can be used only once, there empty space can't be used. [∵ it shows full]

Solution:

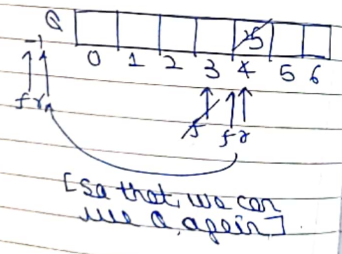
1. Resetting Pointers
2. Circular Queue → soln to a problem of a Queue [not a type of Q]

1. Resetting Pointers

- At any point, when Q becomes empty, bring f & r to -1.
- not always useful [no guarantee, (will work only when Q becomes empty)]

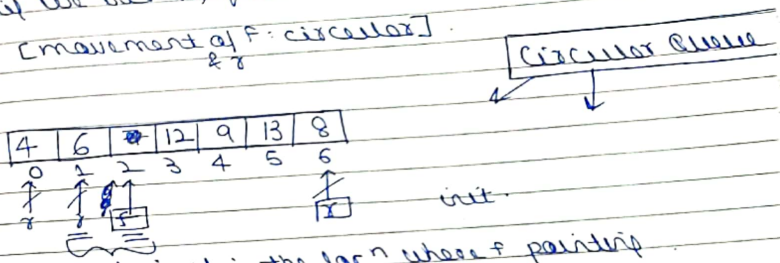
```
int dequeue()
{ int x=0;
  if(f==r)
    pf("Empty");
  else
  {
    f++;
    x=Q[f];
    if(f==r)
      f=r=-1;
  }
  return 0;
}
```

(if you write this in, then call it 3+1 times)



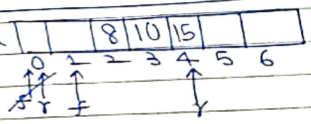
Esa that we can use Q again]

2. If we incr. f, queue size becomes smaller [movement of f & r: circular]



- Initially:

front = rear = 0;

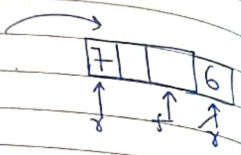


→ front always points b4 first element
→ The loc^n, where front points is always kept empty.

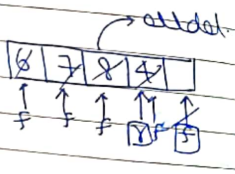
if the size of array = n, we can store n-1 elements

→ No need of using flag ∵ flag = 2 B extra.

```
void enque(int x)
{                                   ①
  if((front==0 && rear==size-1) ||
     (rear+1==front))
     pf("Queue is full");           ②
  else
  {
    if(rear==size-1)
       rear=0;
    else
       rear++;
    Q[rear]=x;
  }
}
```




```
int dequeue()
{
  int x=0;
  if(front==rear)
     pf("empty");
  else
  { if(front==size-1)
       front=0;
    else
       front++;
    x=Q[front];
  }
  return 0;
}
```



Page No.: 161 Date: YOUVA

- mod operⁿ gives circular values.

rear = (rear+1) % size; 0 1 2 3 4 0 1.

```
void enque(int x)
{
  if ((rear+1)%size==front)        → Cheaper
  { pf("Queue is full");
  }
  else
  { rear=(rear+1)%size;
    Q[rear]=x;
  }}
```

- Whenever using Array: always use circular Q:
  but method:
  [linear Q = Space waste]

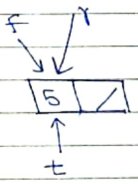
→ Trial Math of implem. Q using Array.

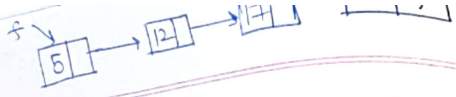
2] Queue using linked list

⇒ 2 pointer f & r

Initially
front=rear=NULL;

- when 1st node is created →





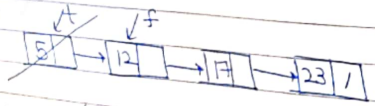
- Empty

```
if(front==NULL)  → out: rear = NULL;
```

- Enqueue

```
void enque(int x)   → O(1)
{
  t=new node;
  if(t==NULL)
    pf("Heap & Queue is full");
  else
  {
    t->data=x;
    t->next=NULL;
    rear->next=t;
    rear=t;
  }
}
```

if rear = absent.
O(n).



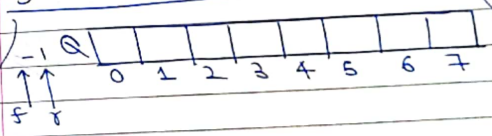

```
t=front;
front=front->next;
x=t->data;
free(t);
```


- int dequeue() → O(1)

```
{
  int x=0;
  if(front==NULL)
  {
    pf("Queue is empty");
  }
  else
  {
    t=front;
    front=front->next;
    x=t->data;
    delete(t);
  }
  return x;
}
```

* Double Ended Queue

size=8.



insert & delete using both pointers.

if front is insert & del: stack.
rear " " : stack.

→ You get built in Queue: Java.
Cant get middle element.

	Insert	delete
front	X	X
rear	✓	✓

Queue

[rear for insertion]

* DE-Queue

	Insert	delete
front	✓	✓
rear	✓	✓

- **i/p restricted DE-Queue**
  - insertion = rear
  - deletion = rear & front.
- **o/p restricted DE Queue**
  - insertion = rear & front.
  - deletion = front.

* Priority Queue

→ fixed no of Priorities → multilevel FB Queue.

5 10 7 8 22 31 24 40 Q_1 | 10 31 |
3 1 2 2 3 1 2 3 Q_2 | 7 8 24 |
Q_3 | 25 22 40 |

when insert: item & priority
when delete from highest Priority Queue.

↳: 10 will be del, then 31, then 7, 8 ...

Every element is inserted based on its Priorities.
i.e no item of P-1 then elements of P-2 are inserted.]

* Fixed no of Priorities.

→ 8 ki priority & hi hai

ex → 8, 15, 24, 7, 150, 104, 12, 6

Unlimited no of Priorities.

→ Service based on taken no: 8.

7 → Highest Priority.

first 7 (then) then 6 ....

smaller no: High P
greater no: Low P

[array]

Q | 8 | 15 | 24 | 7 | 150 | 104 | | ...

X (linear S).

Insert - O(1)
Delete - O(n)

Q | 8 | 15 | 24 | 7 ⇒ insert: O(n).
delete: O(n)

↳ maintaining desc order of priority:

insert: O(n)
delete: O(1).

→ both can be constt

Q | 150 | 24 | 15 | 8 | 7

- Heap is best DS, to implement Priority Q.
  - ↳ implemented using binary tree.
  - Insert: O(log n)
  - Delete: O(log n)

10 6 4 2 1

Page No.
Date.

If a stack is used for implementing Priority Queue, then in which order elements must be inserted ?

↳ descending order of their priorities

1 2 4 6 10 →

↳ desired list.

NOTE: Ranks: 1 > ... > 10 (priority) — Pr ∝ 1/rank

Marks: 1 < ... < 10 — Pr ∝ marks.

Q → 10, 8, 3, 9, 5, 4

```
void fun(Queue q)
{
  if(!isEmpty(q))
  {
    x = dequeue(q);
    fun(q);
    enqueue(q, x);
  }
}
```

fun(q)

→ 4 last element deleted
→ first 4 will be inserted: 4, 5, 9 ... 10.

Page No. 167
Date. YOUVA

fun(q)

x=10 fun(q) deq(q,x)

⇒ reverse

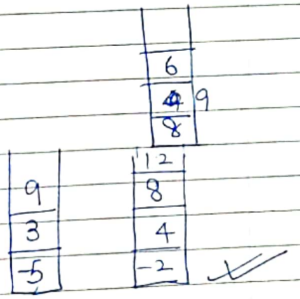
n = 11;

A[] = {8, 4, 6, -5, 3, 9, -2, 4, 8, 12, -10};
(indices 0 1 2 3 4 5 6 7 8 9 10)

```
i=0;
while(i<n)
{
  if(A[i]<0)
  {
    while(!isEmpty(stk)) print(pop(stk));
  }
  push(stk, A[i]);
  i++;
}
```

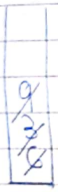
o/p

6 9 8 ✓
9 3 -5
12 8 4 -2



Q → 6, 3, 9, 0, 4, 7, 2, 8, 0, -5, 7, 0

```
while (!empty(q))
{
  x = deque(q);
  if (x)
    push(stK, x);
  else
  {
  while (!isEmpty(StK))
      print(pop(stK));
  }
}
```

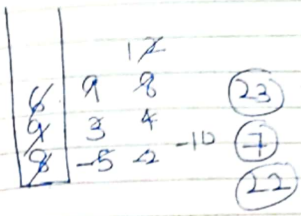


o/p →
9 3 6
8 2 7 4
7 -5

n=11; A[] = {8, 9, 6, -5, 3, 9, -2, 4, 8, 12, -10}

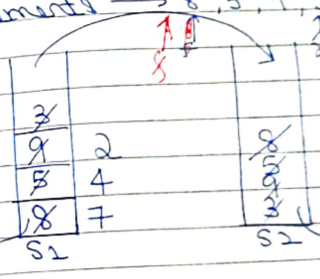
```
i=0;
while(i<n)
{
  if(A[i]<0)
  {
    sum=0;
    while(!isEmpty(stK))
      sum += pop(stK);
  } print(sum);
  push(StK, A[i++]);
```

[not imp to implement stack using 2 queues]



* IMPLEMENTING QUEUE USING 2 STACKS

elements → 8, 5, 9, 2, 7, 4, 2, 16



Stack → FIFO
[Computer itself using Stack] (System Stack)

used for insert [enqueue]

8 5 9 3

used for deletion [dequeue].

if del S1 → S2

```
void enqueue(int x)
{
  push(S1, x);
}
deque()
{
  int x = 0;
  if(!isEmpty(S2))
  {
    while(!isEmpty(S1))
    {
      push(S2, pop(S1));
    }
```

imp

If S2 is empty, transfer item from S1 to S2. [all]

x = pop(S2);
return x;
}

Q Gate Q [Complex Q]

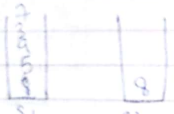
elem → 8, 5, 9, 3, 7, 4, 2, 16

① enque 1, & deque 1 (one by one)

② enque all n, then deque all n.

n visited, m ka delete m<n.

push(min, max)
pop(min, max). n=5, m=4

① 

Push
2*8 = 16

Pop
2+2+2+2+2+2+2+2

⇒ (2n) push 2m+(n-m)
pop 2m

② Enque

for 1 element 1 push
for n elements n push

Deque
for 1 elem 1 pop + 1 push + 1 pop
for m elem 2m pop + m push

171

Total push = m+n
Total pop = 2m

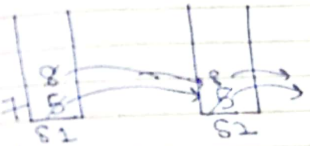
② enque for n : n push

deque for m : n pop + n push + m pop

Total push = 2n
Pop = m+n

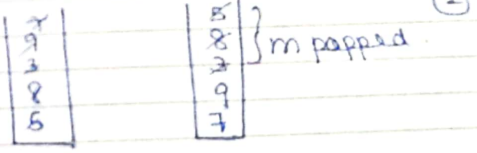
elem → 5, 8, 3, 9, 7

n=5
m=4



① → 4 oper^n's

∴ n push oper^n's



②

m+n ≤ push ≤ 2n
2m ≤ pop ≤ m+n

Stack → push, pop, reverse } operations.

How many stacks you need for Q2. 1.

① enque → push()
deque → reverse(), pop(), rev()

② enque → reverse(), push(), reverse()
deque → pop().

Q A[1.....6] (Array)

[8, 3, —, —, 12, 5] (Array elem)

r f

where front & rear are pointing?

→ points 1 b4 start element.

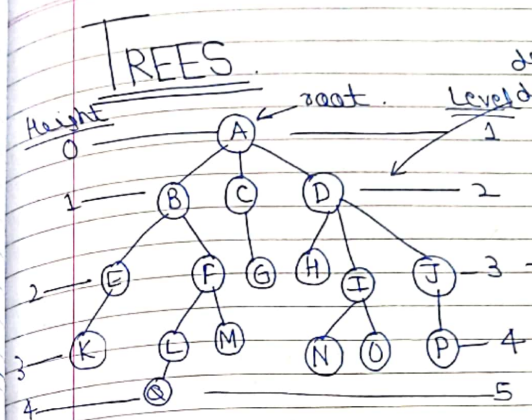
if del 1 elem / insert 2 elem.

r=4
f=5. ✓

5 6 7 8 1

1 8 7 6 5.

TREES



- (H, I, J, N, O, P) are descendants of D.

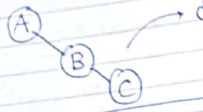
- H I J are child of D.
- E B A are ancestors of K. [along the path to root]

- deg(B) = 2 [no. of children]. (Graph concept = 3)
- deg(G) = 0.

⇒ Node whose degree is 0: leaf / terminal / external node.

Node degree > 0: non-terminal / internal / non-leaf

⇒ degree of a tree = predefined
(if deg(tree) = n;
every node can have max n children.
not more, less — ok



degree (B) ✓
degree (C) ✗

- Analysing horizontally → [count nodes]
  - divide into levels.
- Analysing vertically.
  - height [count edges]
  - We start counting from 1 [

(vert) Height: from 0 (root). [count edges]
(horz) Level: from 1 (root) [count nodes]
↑ Gate

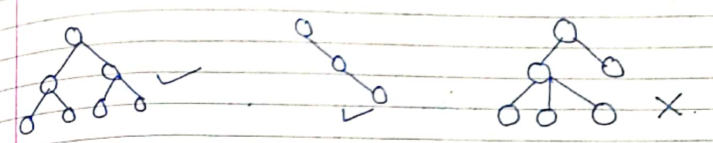
→ during Analysis (either start from 0/1)

* Height / Depth: same of tree. (opp)
  - count top - bottom / bottom - top : same
    analysis depends on your analysis.

* Height/depth of a Node

Ht	depth
0	4
1	3
2	2
3	1
4	0

* BINARY TREES (tree of degree 2).
  - every node can have atmost 2 child.



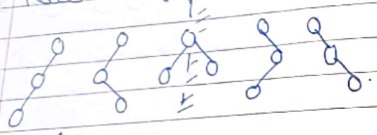
25/8/19
Sunday

Lecture 6

n=3. (unlabelled nodes)

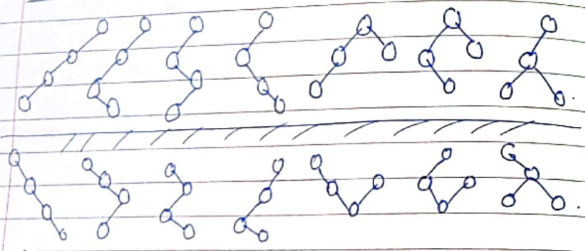
0 0 0

How many diff BT?



T(3)=5

⇒ n=4.



diff binary trees → $T(n) = \frac{2n C_n}{n+1}$ ✓ $\frac{2n C_n}{n+1}$.

- n=5 ⇒ $\frac{10 C_5}{6}$

⇒ $\frac{10!}{5!\,5!} \times \frac{1}{6}$

⇒ $\frac{10 \times 9 \times 8 \times 7 \times \cancel{6} \times \cancel{5!}}{5!\ \cancel{5!}\ \cancel{6}}$

⇒ 42.

- if n=3
Max Height = 4.

- if n=4.
max Height = 8

Max Height = 2n-1

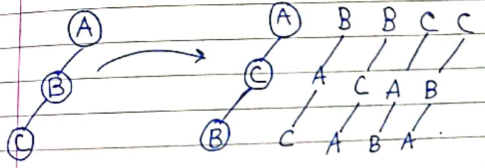
↳ How many binary trees we can draw with max height.

If n=10 nodes, how many BT can created with max height.?

⇒ Max height = 2^9 = 512.

∴ $\frac{2n C_n}{n+1} - 2^{n-1}$ = all those trees with ht less than max ht.

* No. of Binary Trees with Labeled Nodes



$T(n) = \frac{2n C_n}{n+1} * n!$

↳ Shapes ↳ permutation of labels.

* $T(n) = \frac{2nC_n}{n+1}$ → Analysis.

n	0	1	2	3	4	5
T(n)	1	1	2	5	14	42

empty binary Tree.

$T(5) = 1*42 + 1*14 + 2*5 + 5*2 + 14*1$

$\Rightarrow 42$.

- $T(5) = T_0*T_4 + T_1*T_3 + T_2*T_2 + T_3*T_1 + T_4*T_0$.

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

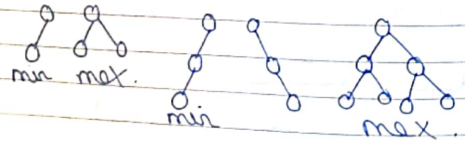
in C: this is for loop

recursive function.

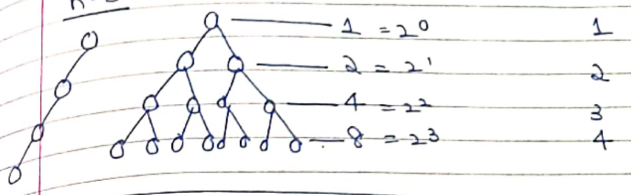
* Height V/S Nodes.

- If height h is given

h=0 h=1 h=2



h=3



min nodes n = h+1
max nodes n = $2^{h+1} - 1$

← if height h is given.

$2^0 + 2^1 + 2^2 + 2^3 = 15 = 2^4 - 1$.

$$2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 \quad (1)$$

GP Series.

- $a + ar + ar^2 + ar^3 + \cdots ar^K = \frac{a(r^{K+1} - 1)}{r-1}$

$$1 + 2 + 2^2 + 2^3 + \cdots 2^h = \frac{2^{h+1} - 1}{(2-1)} \quad (2)$$

$a = 1, r = 2$.

max nodes in level (K) = $2^K - 1$
min nodes in level (K) = 1

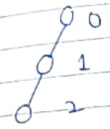
* If 'n' nodes are given : (what is min/max ht BT, you can create).

min h = $\lceil \log_2(n+1) - 1 \rceil$
max h = n − 1

$n = 2^{h+1} - 1$.
$2^{h+1} = n + 1$.
$h = \log_2(n+1) - 1$

Q n=3.

max h = 2.



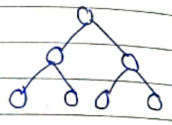
min h =  ⇒ 1

$\log_2(4) - 1 = 1$

Q n=7.

max h = 6 (skewed BT)

min h = $\log_2(8) - 1 = 2$



* n=5.

min = $\log_2(n+1) - 1$.

$h = \log_2 6 - 1 = 2.4 - 1 = \lceil 1.4 \rceil = 2$ (upper int. / ceil)

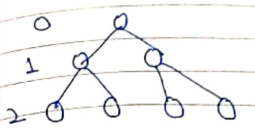
* n=8

8
2
2
2 } 3 times divide.

$\log_2 8 = 3$ ← successive division.

* n=7.

n+1 = 8.



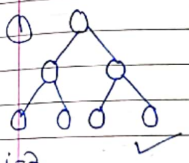
} dividing 3 times

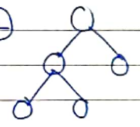
$\therefore \log_2(n+1) - 1 = 2$ $\therefore h = 2$.

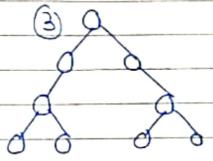
If represented in Array, no Gap b/w elements

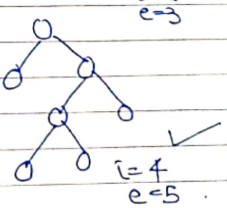
* Complete Binary Tree / Strict BT / Proper BT

- A BT in which every node can have zero child/ exactly 2 children.
- Node wouldn't have 1 child.

①  i=3 e=4

②  i=2 e=3

③  X

④  i=4 e=5

H/W: Find ht v/s node analysis of Complete BT.

strict no. of internal nodes & external nodes (leaf).

for Complete BT (not General tree)

- $e = i + 1$ → branch nodes; leaf nodes
- $n = i + e$

Q) A complete BT is having $e = 7$ nodes. Find total nodes nodes n,

$i = 6$, $n = 13$

Q) A strict BT is having $n = 17$ nodes. Find no. of internal nodes i.

$17 = 2i + 1$

$i = 8$

Q) A Strict BT is having $n = 20$ nodes. Find no. of leaf nodes.

$20 = 2i + 1$

$19 = 2i$ ✗

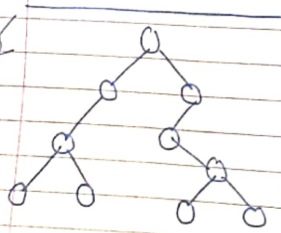
$i + e = 20$

$e - 1 + e = 20$

$2e = 21$

in SBT → no. of nodes will be odd

Q)



$i = 6$

$e = 4$

∴ not Complete BT.

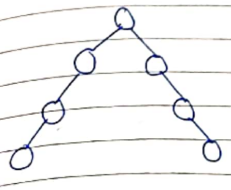
$deg(0) = 4$

$deg(1) = 3$

$deg(2) = 3$

$deg(0) = deg(2) + 1$

Q)



$deg(0) = 2$

$deg(1) = 4$

$deg(2) = 1$

$deg(0) = deg(2) + 1$ (leaf)

In BT, there is a strict relation b/w degree (0) & deg (2)

Q) If a BT is having 10 nodes with degree 2, 5 nodes with deg 1. Find total no. of nodes.

$11 + 5 + 10 = 26$

deg 0, deg 2.

Q) A BT is having $n = 20$ nodes & there are 3 nodes with deg 1. Find leaf nodes.

$20 = n + n + 1 + 3$

$16 = 2n$

$n = 8$ ✗

deg 2 → e − 1; deg 1 → 3; deg 0 → e

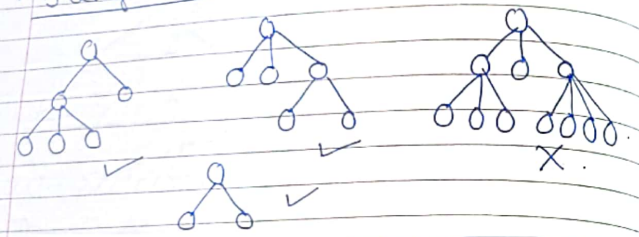
$20 = n + n - 1 + 3$

$n = 9$

* n-Ary Tree

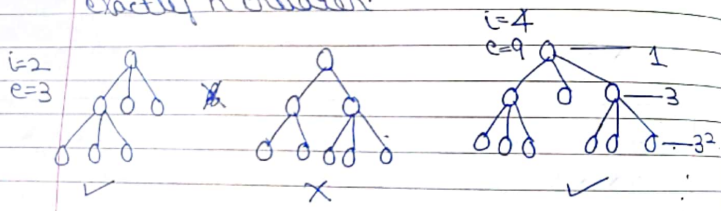
- Tree of degree n, where each node can have atmost n children.

• 3-ary tree



* Complete n-Ary Tree (n=3)

- A tree in which every node has 0 children or exactly n children.



$1+3+3^2+\ldots\ldots 3^h$: GP.

$\Rightarrow \frac{3^{h+1}-1}{3-1} \Rightarrow \frac{3^{h+1}-1}{2}$.

W: Ht v/s node analysis.

leaf nodes

$e = 2i+1$ → Complete 3 ary tree.

$e = (n-1)i+1$ → Complete n-ary tree.

Q. A Complete 3-ary tree is one, where every node can have 0/3 children, is having x internal nodes. Find no. of leaf nodes.

$e = 2x+1$

Q. A complete 3-ary tree is having n nodes. Find no. of leaf nodes.

$e + \frac{e-1}{3} = n$ (i)

$\Rightarrow 3e + e - 1 = 3n$

$3e - 1 = 3n$

$e = \frac{3n+1}{3}$

$e+i=n$

$i = n-e$

$e = (n-1)i+1$

$e = 2i+1$

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

$e = 2n-2e+1$

$3e = 2n+1$

Q. A Strict n-ary tree is having $i=10$ internal nodes and $l=41$ leaf nodes. Find n. (n → degree of tree.)

$41 = (n-1)10 + 1$

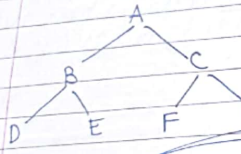
$41 = 10n - 10 + 1$

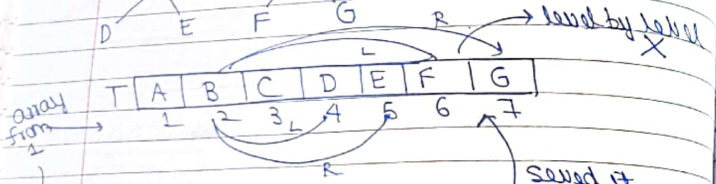
$41 = 10n - 9$

$n = 5$

* Representation of BT → non-linear.

① Array rep.
② Linked rep.

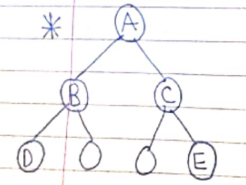




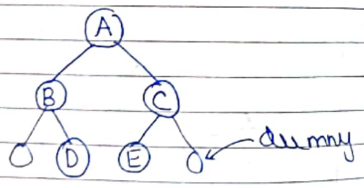
if a node is at index i

Left Child = 2 * i.
Right Child = 2 * i + 1
Parent = $\lfloor \frac{i}{2} \rfloor$.

saved it, using formula



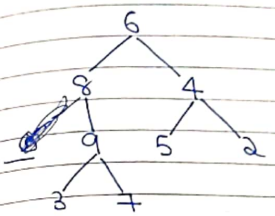
A	B	C	D	–	–	E
1	2	3	4	5	6	7



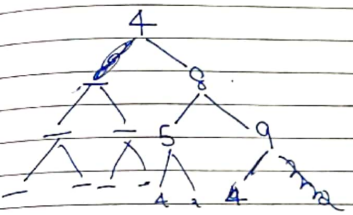
A	B	C	–	D	E	–
1	2	3	4	5	6	

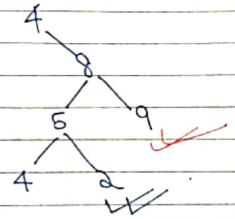
Page No. 187 Date youva

① 6, 8, 4, —, 9, 5, 2, —, —, 3, 7.

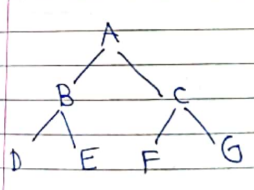


② 4, —, 8, —, —, 5, 9, —, —, —, —, 4, 2.





② Linked rep



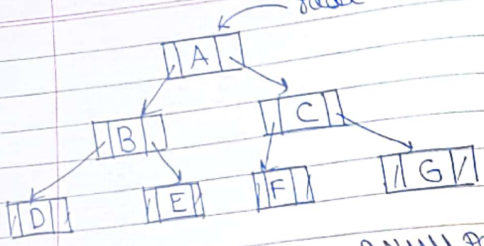
Node

Lchild	data	rchild

```
struct node
{
  struct node * lchild;
  int data;
  struct node * rchild;
};
```

B

root.



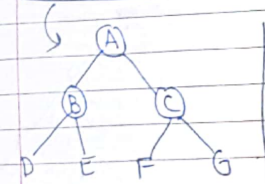
7 nodes

8 NULL Pointers

for Binary tree:

if n nodes = n+1 null pointers

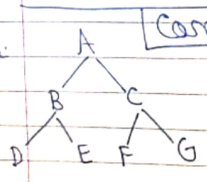
* Full Binary Tree



- A BT of height h is having max nodes $[n = 2^{h+1} - 1]$. (no Space in that height)

$$\max n = 2^{h+1} - 1$$

1. Complete BT



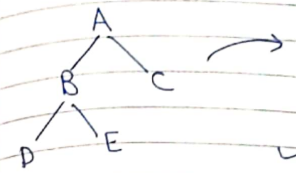
T | A | B | C | D | E | F | G |

almost complete BT

Full must be complete
Complete need not be full

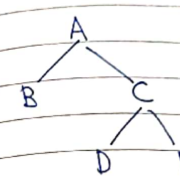
Page No. 189
Date

2.



| A | B | C | D | E |

3.

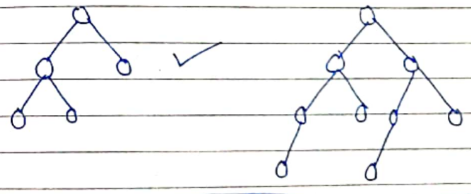


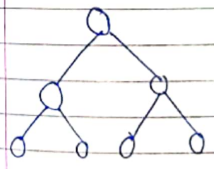
A B C — — D E

Complete : if in Array, no Gaps
Strict : 2 or 0

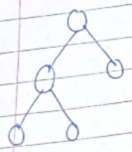
dont allow blanks in Array.

Complete BT of height h will be full BT till height h−1, & in the last level, elements will be inserted to left to right.

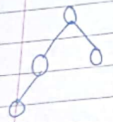




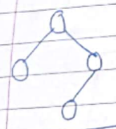
1. Full
2. Strict / proper / complete
3. almost complete



complete,
almost complete.

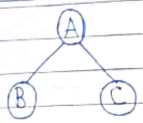


almost complete.



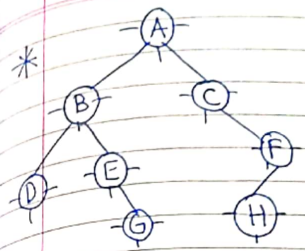
X.

TREE TRAVERSALS



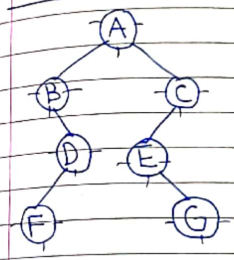
- preorder → visit(node), preorder(left), preorder(right),
  eg: A, B, C.
- Inorder → left, root, right. → inorder(left) visit(node) inor(right)
  eg: BAC
- Postorder → left, right, root.
  eg: BCA.
- Levelorder → level by level.
  eg: ABC.

Page No. 191 YOUVA
Date:

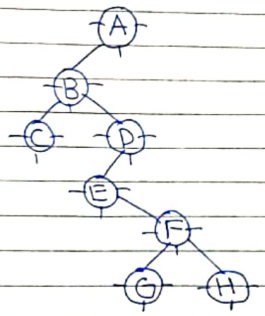


Preorder: when you left, Tick.
Inorder: when you bottom, Tick
Postorder: when you right, Tick

Preorder: ABDEGCFG ✓
Inorder: DBEGACHF ✓
Postorder: DGEBHFCA ✓
Levelorder: ABC DEFGH ✓



Pre: ABDFCEG.
In: BFDAEGC.
Post: FDBGECA.
Level: ABCDEFG

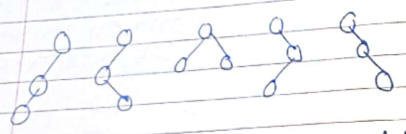


Pre: ABCDEFGH.
In: CBEGFHDA.
Post: CGHFEDBA.
Level: ABCDEFGH.

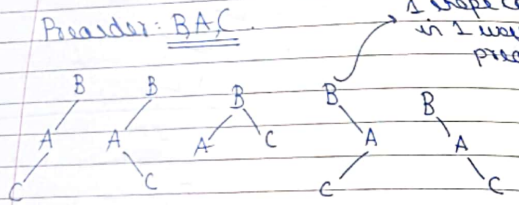
* n=3.

$\frac{2nC_n}{n+1} * n!$ BT created.

(A) (B) (C) ⇒ 5 Trees (if unlabelled)



Preorder: BAC



No. of BT (if labelled nodes) with a given Preorder = $\frac{2nC_n}{n+1}$.

* From just Preorder, we can't generate unique BT, no. of BT we can generate: $\frac{2nC_n}{n+1}$

* From preorder/inorder/postorder, we can generate total of $\left[\frac{2nC_n}{n+1}\right]$ BTs.

→ Given the tree Traversal.

→ If an unlabelled BT is given, how many ways it can be labelled, to get a specific preorder.

⇒ 1.

* Generating BT from given traversals

- To generate a tree, we need 2 traversals, & one must be inorder.

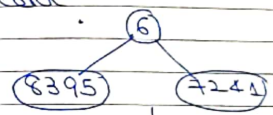
{Pre + in / Post + in} → can only generate BT.

Pre: RLR.

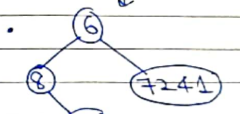
- pre ⟶ 6, 8, 3, 5, 9, 7, 4, 2, 1
- In ⟶ 8, 3, 9, 5, 6, 7, 2, 4, 1

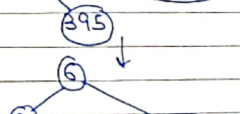
8 3 9 5 6 7 2 4 1

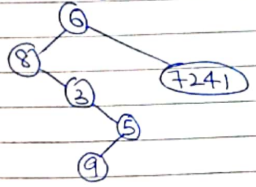
Search

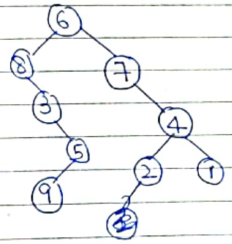


[Subtrees]

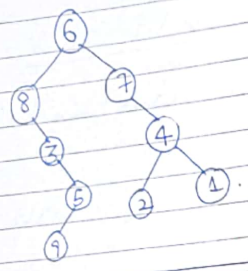








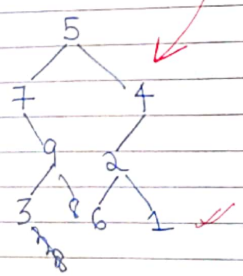
pre → 6 8 3 5 9 7 4 2 1
in → 8 3 9 5 6 7 2 4 1



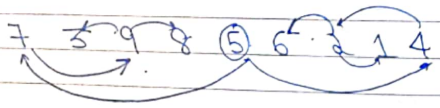
Worst Case → $O(n^2)$ — Time taken for generating tree from traversal

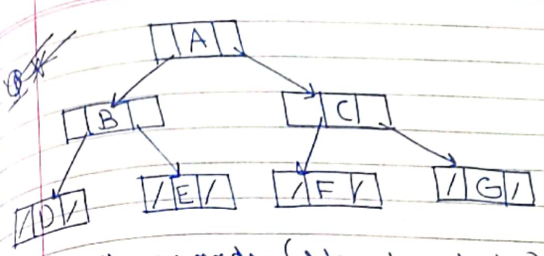
* In Postorder: right → left Scan.
In Preorder: left → right Scan.

pre: 5, 7, 9, 3, 8, 4, 2, 6, 1.
In: 7, 3, 9, 8, 5, 6, 2, 1, 4.



3 & Postorder.
8 3 9 7 6 1 2 4 5






```
void preorder (struct node *p)
{
  if (p != NULL)
  { printf("%c", p->data);
    preorder (p->lchild);
    preorder (p->rchild);
  }
}
```

n nodes → 2n + 1 cells : 15.
n+1 Null
2n+1

For n nodes → 2n + 1 cells → O(n).
[Time depends on no. of cells]

Time = O(n)
Space = O(n)
(for preorder)

→ C will be printed in pth cell.

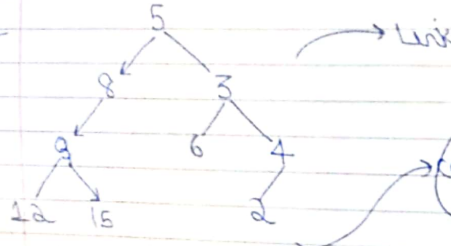
```
void inorder(struct node *p)
{
 if(p!=NULL)
 { inorder(p->lchild);
   pf("%c",p->data);
   inorder(p->rchild);
 }
}
```

```
void conInorder
 {
   con(p->rchild);
   pf(" ");
   con(p->lchild);
```

Q GCFAEBD. (Call)

[GGCFFCAEEBDDBA].

Q



→ Linked rep.

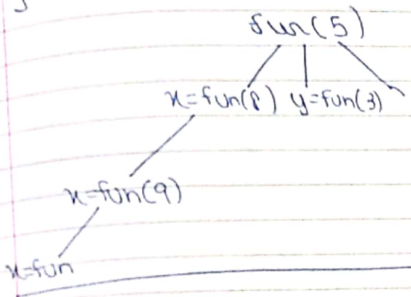
(count no. of nodes [recursive])

```
int fun(Node *p)
{
 int x=0, y=0;
 if(p)
 {
```

o/p=9


```
  x=fun(p->leftchild);
  y=fun(p->rchild);
  return x+y+1;
 }
 return 0;
}
```

(Postfix counting)



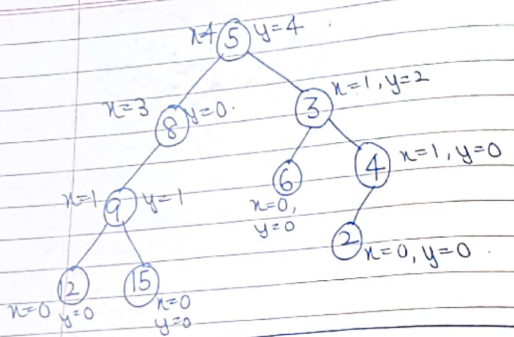
- func^n call itself 9 times.
- func^n working in postorder way.

Q This code by chp to count no. of nodes, who are having 4 descendents.

→ O(n)

[If we are visiting every node, only 1 time ⇒ O(n)].

→ go to every node & check, count++.

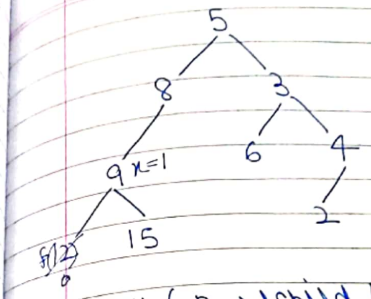


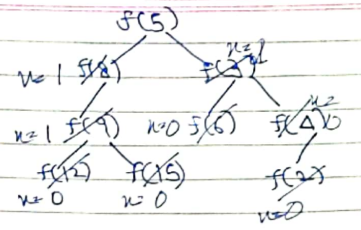
Q
```
int fun(Node *p)
{
  if(!p) return 0;
  return fun(p->lchild) + fun(p->rchild)
         +1;
}
```
→ Some.

Q
```
int fun(Node *p)
{
  int x, y;
  if(!p) return 0;
  x = fun(p->lchild);
  x = x + fun(p->rchild);
  if(p->lchild && p->rchild)
      return ++x;
  return x;
}
```
→ count nodes with deg 2.

→ Postfix (Postorder)

count when both child exist.





→ deg 1 & 2.

if(p->lchild || p->rchild) : Non-leaf

deg 0 → if(!p->lchild && !p->rchild) : leaf

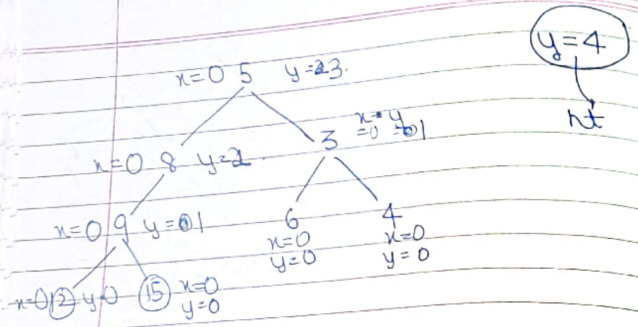
if(!p->lchild || !p->rchild);

→ deg 0 & 1.

if(p->lchild != NULL ^ p->rchild != NULL)

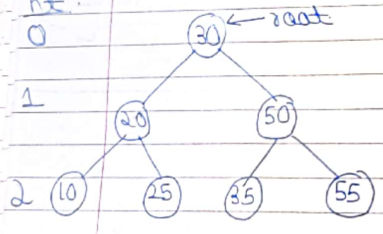
→ deg 1

Q
```
int fun(Node *p)
{
  int x, y;
  if(p)
  {
    x = fun(p->lchild);
    y = fun(p->rchild);
    if(x > y)
      return x+1;
    else
      return y+1;
  }
  return 0;
}
```
→ Postfix. [2n+1]



if want to start height from 0 : return -1

* BINARY SEARCH TREE



→ BT, with unique set of elements, where, value (elem) in greater than all elem in left ST & smaller than all elem in Right ST.

- Key = 25 (3 comparison — found).
  Key = 50 (2 comp.)
  Key = 40 (not found → 3 comp)

depends upon height of the tree.

```
node * Search(node *p, int Key)
{
  while (p)
  { if (Key == p→data).          → iterative.
      return p;
    else if (Key < p→data);
      p = p→lchild;
    else
      p = p→rchild;
  }
  return NULL;
}
```

```
node * Search (node *p, int Key).
{ if (p)
  {
    if (Key == p→data)
      return p;
    else if (Key < p→data).
      return Search (p→lchild, Key).
    else
      return search (p→rchild, Key);
  }
  return NULL;
}.
```

① 30, 50, 35. ✓
② 30, 20, 25, 50, 35 ✗. root = 30, where L.
③ 30, 50, 32, 60, 35 ✗.
Search 35.
60 won't get, bcoz 60 is in left of 35.

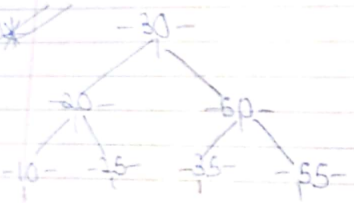
Q Tree: Keys → 1 to 100
key = 55
which of the following search seq are T/F?

① 35, 90, 40, 86, 45, 70, 65, 55 ✓

② 10, 80, 20, 25, 75, 35, 70, 38, 72, 42, 60, 55 ✗

③ 40, 70, 42, 69, 45, 48, 60, 46, 50, 55 ✗

10 - 100
10 - 100 10
10 - 90 90
20 - 90 20



10 20, 25, 30, 35, 50, 55

Inorder of BST = always Sorted order of elements

203

Q pre: 30, 20, 10, 5, 12, 15, 25, 24, 28, 50, 35, 31, 55, 52, 60

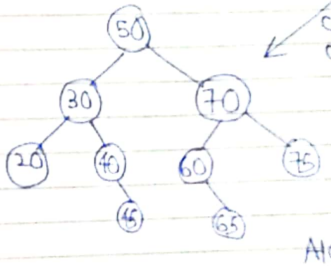
only 1 traversal sufficient to create BST (either preorder/postorder)

In: 5 10 12, 15 20 24, 25, 28, 30, 31, 35, 52, 55, 60

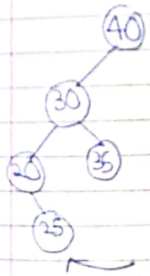
Preorder — can there be PreOrder of BST?

Q
1. 50, 30, 20, 40, 45, 70, 60, 65, 75 ✓
2. 40, 30, 20, 25, 35, 28, 60, 50, 70, 55 ✗

0 - 50
0 - 30
20



can direct create BST O(n)



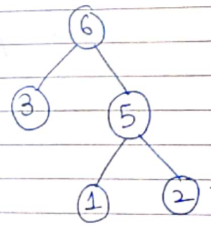
28 here ✗

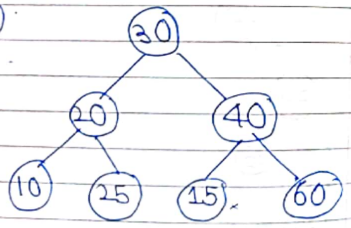
Algorithm to create BST from pre/post = O(n)

But, create tree = O(n²)

```
int fun(node *p)
{
  if(p)
  {
    if(p->lchild && p->data < p->lchild->data
       ||
       (p->rchild && p->data > p->rchild->data))
      return false;
    return fun(p->lchild) &&
           fun(p->rchild);
  }
  return true;
}
```

→ Preorder can't tell you whether its BST or not. [only inorder tells]





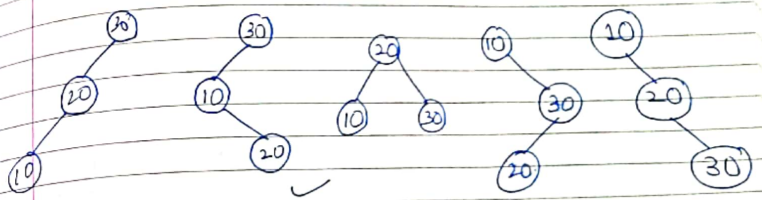
fun() return true. (but it's not BST).

Q. n=3

→ 5 BSTs are possible.

$T(n) = \frac{2nC_n}{n+1} * 1$.

∵ Ascending → 1 inorder = 1.

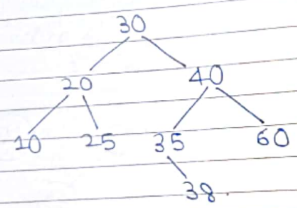


Q. In how many ways, an unlabeled Binary tree can be populated with ~~the~~ keys to form a BST. n
(to get specific inorder)

Inorder same.

* Insert Key = 38

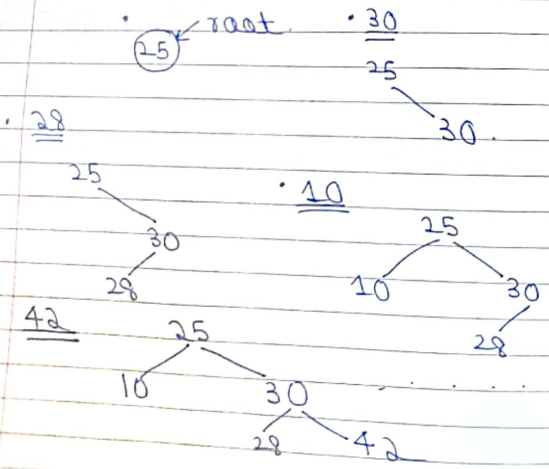
- first search 38 where it fails, insert key there.
  ↳ inserted as a leaf node.

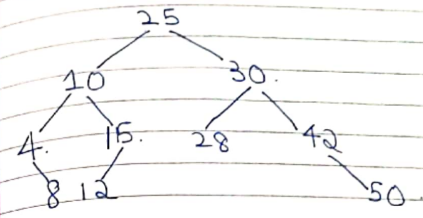


If found = don't insert it (duplicates)

* Creating a BST

Keys → 25, 30, 28, 10, 42, 50, 15, 12, 4, 8,



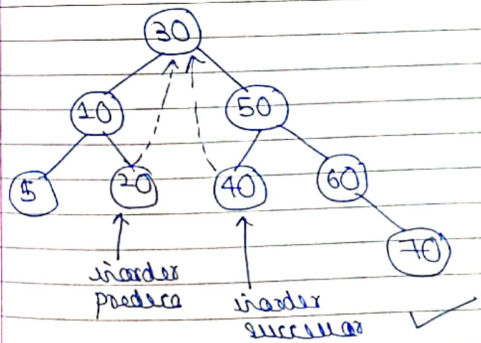


Q. Keys → K, n, d, r, l, h, a, e, i.
(Create BST) → H(W)

Q. preorder?
height?
search time?
which key → which level?
If key added: ht?
} based on tree drawn.

Q. Time taken to insert key, depends on height.
O(h) ; min height = O(log₂n)

* Deletion from BST



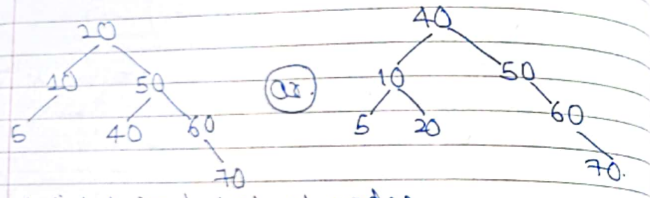
only 1 modification

- Case 1 : Key = 70 (leaf node).

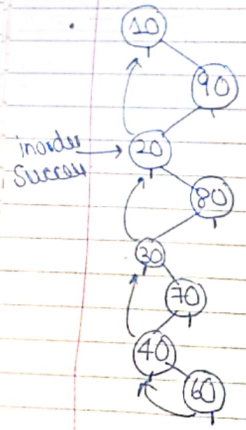
Case 2 : Key = 60; [child will take its value].

Node = deg 1

Case 3 : Search for Key,
delete it,
replace it with inorder pred / inorder successor.



∴ didn't disturb set of nodes.



delete Key = 10

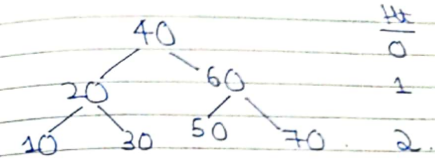
Worst case, in which multiple changes are required.

→ on Paper: we can make 90 as root.

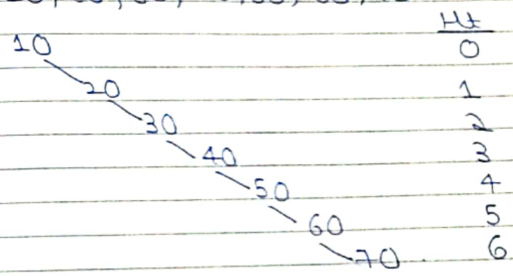
* Drawback of BST :

Insertion order

- Key → 40, 60, 50, 20, 10, 30, 70



- Key → 10, 20, 30, 40, 50, 60, 70.



Ht of BST depends upon order of Insertion

min h = log n
max h = n

Search time — expectation.

log n ≤ t ≤ n.

- Purpose of BST : Search time ↓ to log n.
if not log n, why use BST.

- BST doesn't guarantee search time to be $\log n$, it can be atmost n also.

$\text{Logarithmic} \leq h \leq \text{Linear}$

h not in control (depends on insertion)

Insertion, Deletion, Search — all depends on height

* **Height balanced BSTs**

→ 1. AVL Tree ✓
→ 2. Red-Black tree (not in Gate)

→ for every node, we calculate balance factor.

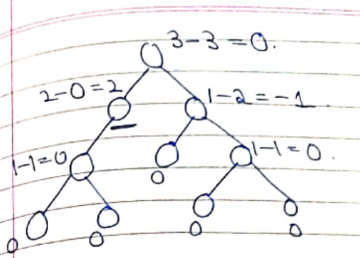
$BF = \text{ht of Left Sub tree} - \text{ht of Right Sub tree}$

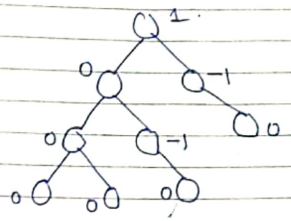
$bf = h_l - h_r = \{-1, 0, 1\}$

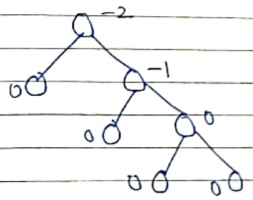
$|bf| = |h_l - h_r| \leq 1$ → node is balanced.

- if $|bf| > 1$, node is im-balanced.

NOTE: To balance a node, we perform rotations.

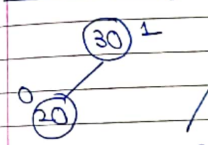




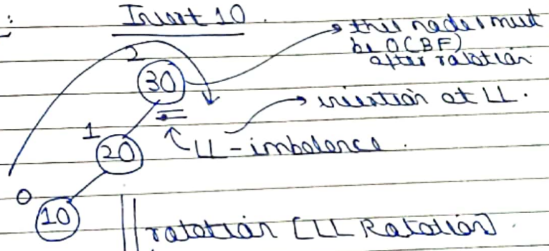


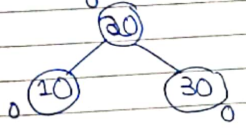
* **Rotations.**

Initially:



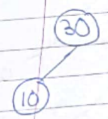
Insert 10.



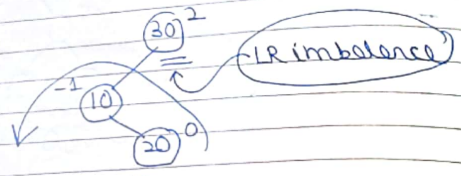


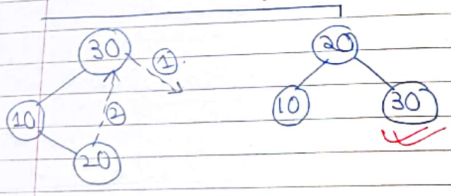
✓

Initially



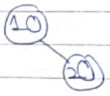
insert 20



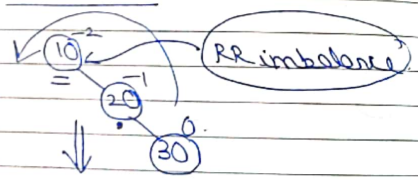


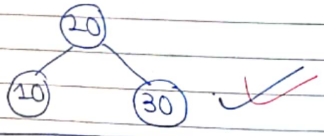
Left → right in place of root & move root to right.

Initially

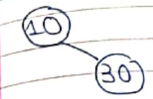


insert 30

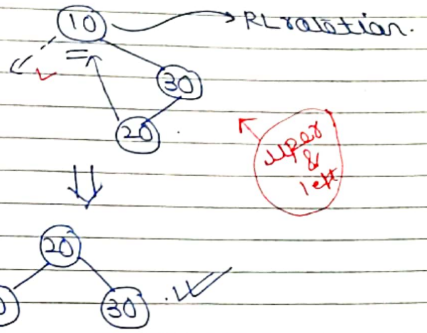




Initially

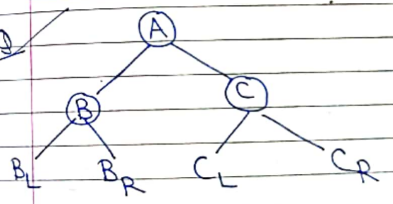


Insert 20



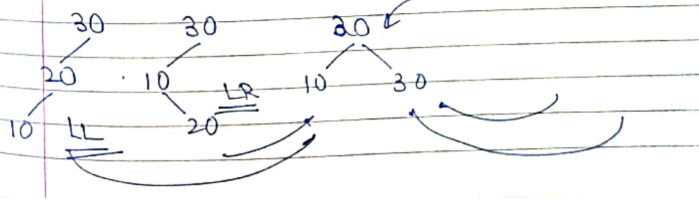
LL & RR – rotation: Single rotation → 1.
LR & RL – rotation: double rotation
→ count it as 1.
→ counted as 1 rotation only.

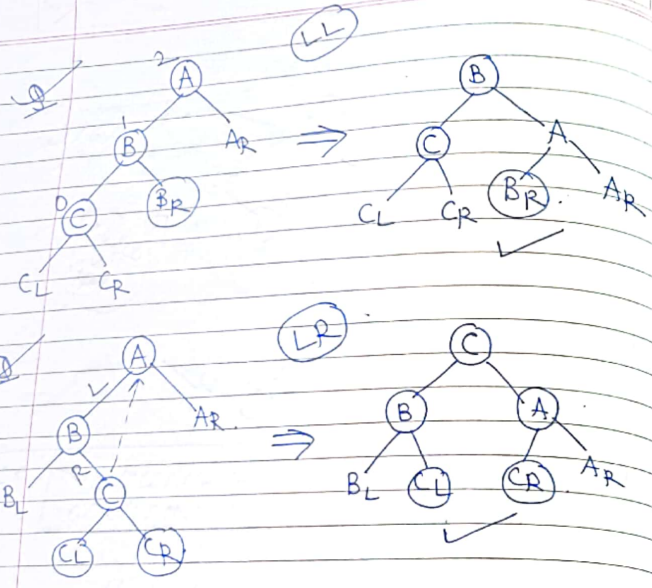
Q



(A) (B) (C).

[converted into] (rotate)



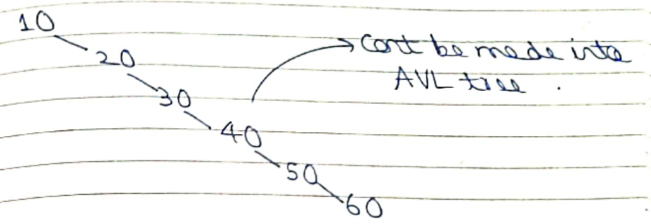


* ~~When~~ inserting, first balance, ~~the first~~ ancestor from the newly inserted node.

* AVL tree can be made during insertion, not a given ~~initial~~ BST (unbalanced) can be made into AVL.

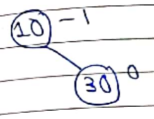
↳ BF ⟶ {−5, −4, −3, ⋯} ✗

BF can be greater than −2.

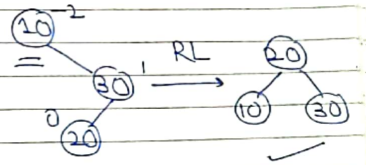


Q Keys ⟶ 10, 30, 20, 40, 50, 33 — 3 rotations

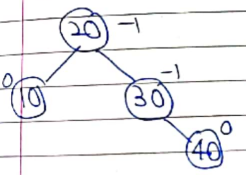
1) Insert 10, 30



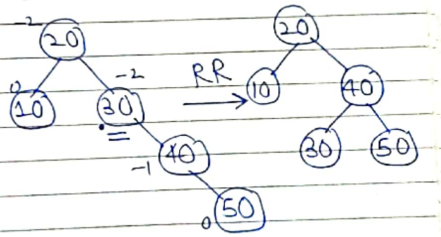
2) Insert 20



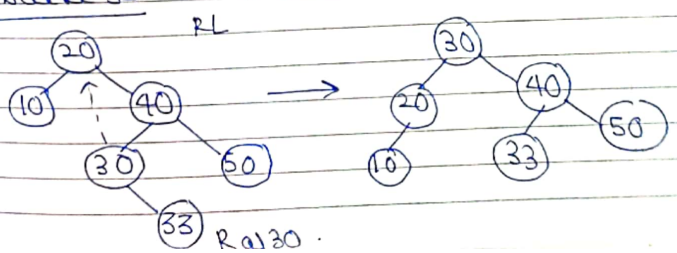
3) Insert 40



(4) Insert 50



(5) Insert 33



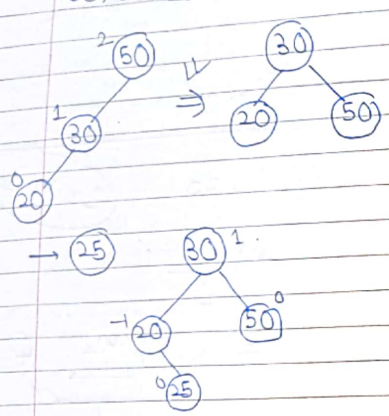
Ra) 30.

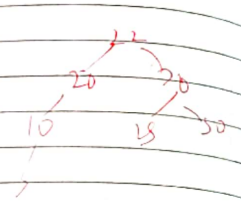
HW

Keys → 10, 20, 30, 40, 50, 60, 70

① Keys:

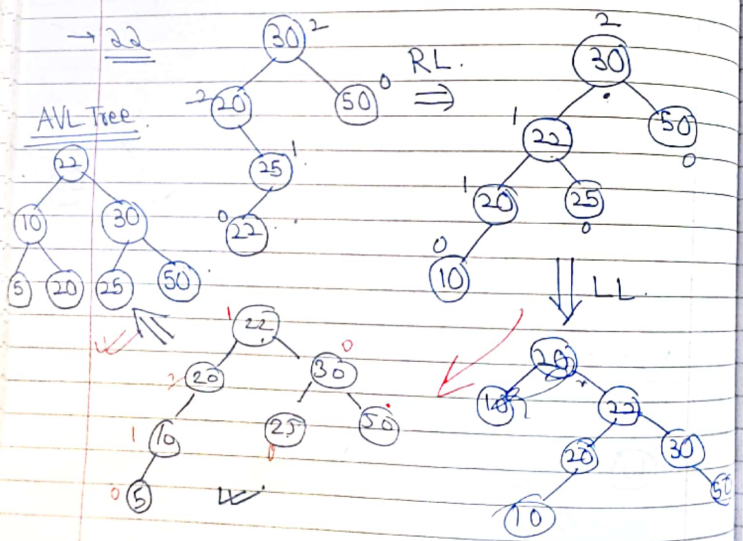
50, 30, 20, 25, 22, 10, 5.





→ 22

AVL Tree.



RL.

LL.

* Height V/S Nodes of AVL tree

if height 'h' is given:

min n = ?

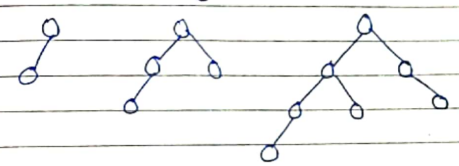
max n = $2^{h+1} - 1$.

if n nodes are given:

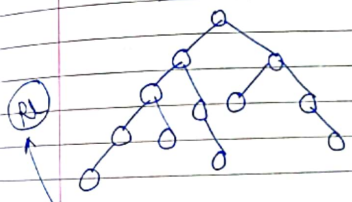
min h = $\lceil \log_2(n+1) - 1 \rceil$ ← height starting from 1

max h = ?

h=0 h=1 h=2 h=3 h=4



h=5



RL

$$N(H) = \begin{cases} 0 & h=0 \\ 1 & h=1. \\ N(H-2) + N(H-1) + 1 & h>1 \end{cases}$$

h	0	1	2	3	4	5	6	7	8	9	10
N(h)	0	1	2	4	7	12	20	33	54	88	143

5 height ke liye min. 12 nodes

↳ behaves like Fibbonacci. ↳ jaanaw due to balanced.

⇒ AL → balanced

Q. If height h is given:

min n: look in the table.
max $n = 2^h - 1$

If n nodes are given:

min $h = \log_2(n+1)$.

max h = look in the table ✓

Q. If height of AVL tree, h=5, then ($h \geq 1$), find minimum nodes & maximum nodes.

max : 31 ✓
min : 12. ✓

Q. If height of AVL tree, h=5; ($h \geq 0$) find minimum & max nodes.

min : 20
max : 63 → $2^{(h+1)} - 1$ ✓

Q2. If AVL tree is having n = 7 nodes; what is min & max height ($h \geq 1$).

min h = 3.
max h = 4 ✓

n=7

$h \geq 0$.

$h = 2^{h+1} - 1$
$\log(n-1) - 1$.

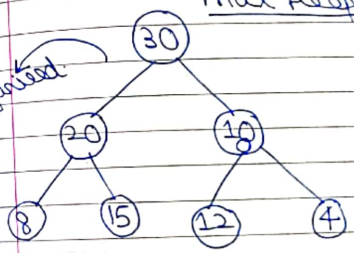
min h = 2.
max h = 3 → 4-1=3 ✓

Q. If AVL tree is having n = 10 nodes. Find max h. ($h \geq 1$).

max h = 4 ✓ 7-11 → max h = 4.

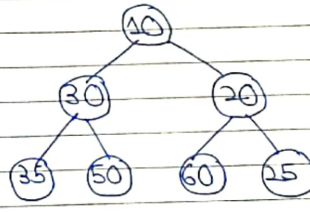
* HEAP → mostly implemented in Array (∵ no Gaps). ∴ complete.

max Heap



organised

min Heap



⇒ Heap is a complete BT; satisfying:
(i) every node is greater than or equal to all its descendents [Max Heap]
(ii) every node is lesser than / equal to all its descendents [Min Heap]

Q/ Find out which is heap! [max/min]

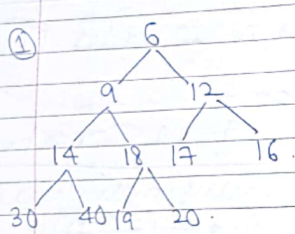
① 6, 9, 12, 14, 18, 17, 16, 30, 40, 19, 20.

② 50, 30, 20, 18, 4, 16, 12, (24), 21, 11.

✗ ③ 40, 35, 16, —, —, 12, 18

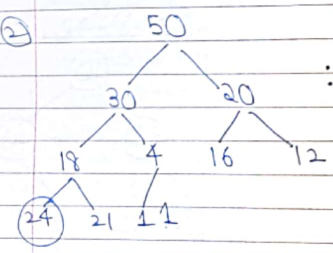
→ (24) → cracked problem from 24

①



: min Heap.

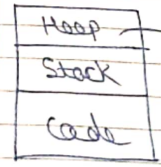
②



: min/max.

* height of heap always $\lceil \log_2 n \rceil$

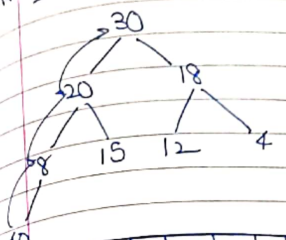
→ O(log n).

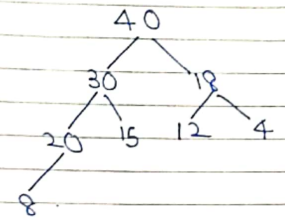


→ unorganised mem.

① should be complete

* INSERT: 40.





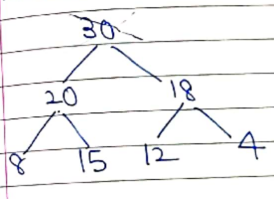
H	30	20	18	8	15	12	4	40

H	40	30	18	20	15	12	4	8

Insert: O(log n)

* ~~Delete~~ [can't delete of any choice]

(only delete root) [max element]



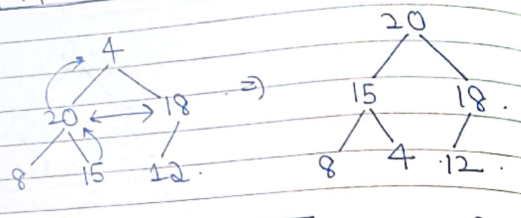
→ larger the no. → higher the priority.

- min heap: smaller the no., higher the priority.

- Heap useful for implementing priority queue.

~~30~~ 4	20	18	8	15	12	4

H | 4 | 20 | 18 | 8 | 15 | 12 |



Deletion = O(log n) (as till 4)

- inserting keys one by one, & form heap.

T = n log n (for n keys)

↳ insertion.

Heap Sort

- create Heap — n log n.
- Delete heap (one by one)
  - ↳ n log n
  - 2n log n.

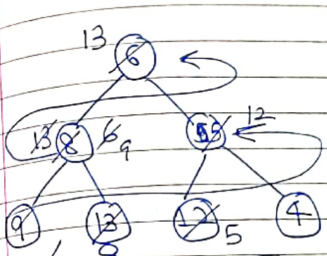
∴ O(n log n)

- First elements into heap & then delete one by one.

A | 16 | 8 | 5 | 9 | 13 | 12 | 4 |
1 2 3 4 5 6 7

→ insert & creating heap in single Array.
- delete also in same array (Keeping in both)

In place Sorting
↳ Heap Sort.
like bubble S, insertion S...



(insert: loop towards root).

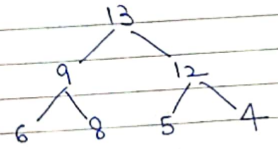
check from bottom

easy work: ~~insertion~~ Heapify (for creating Heap)

O(n).

Min time for creating heap.

Max time for creating heap. = O(n log n)
(inserting)



31/8/19 Saturday

Lecture 7

- Create heap = O(n)
- Insert element (one by one) = O(n log n)

* To delete min element from max heap

max → min : O(n)
delete : O(log n)
min → max : O(n)
f(n) = O(n)

* Search → O(n) (LS) in Heap

Q delete 7 elements from heap

1 — O(log n)
2 — O(2 log n)
⋮
7 — O(7 log n)

∴ O(log n)

Q $f(n) = n^2 + n$: $O(n^2)$ ← Asymptotic notation
$f(n) = 5n^2 + 8n$: $O(n^2)$
$f(n) = \log n + 5$: $O(\log n)$
$f(n) = 7\log n + 8$: $O(\log n)$

Q Time taken for sorting elements using Heap Sort is n log n. How many elements are there?

$\log n = x \log x$
$x = \frac{\log n}{\log x}$

(4) $\frac{\log n}{\log\log n}$
(3) n
(2) log n
(1) 1

- HS takes n log n.
for n elements ——— n log n
x ——— log n

$\frac{\log n}{\log\log n} \times \log_a \left[\frac{\log n}{\log\log n}\right]$

$\log n \longrightarrow \log n \log\log n$

$\frac{\log n}{\log\log n}\left[\log\log n - \log\log\log n\right]$ (negligible)

$\approx \log n$ (4)

* Tree → max ht = O(n) & O(log n)
Heap → O(log n) = ht
↳ always log n (can't be more/less).

* BST height

min h = log n.
max h = n.

AVL Tree

min h = log n.
max h = log
$1.44 \log_2(n+2)$
$\approx \log n$

* TIME & SPACE COMPLEXITY

- 2 Methods for Analysis:

- based on working.
- based on code written.

[Analysis done on algo not on program]

∴ Algorithm ⟷ program
↑ design. [Analysis]
↑ product. [testing].

Q. Algorithm My(x, y, z)
{
  r = 2 * x + 3 * y / z;
  return r;
}
[2 statements]

[in HLL]
every statement in algo takes 1 unit of time.

f(n) = 2 ∴ T = O(1)

- debugging is done, when required (while Analysing).

Space
x — 1 word, ∴ 4 words.
Y — "
Z — "
r — "
S = O(1)

- Every ~~word~~ var takes 1 word.

Q. Algorithm Add(A, n)
{
  s = 0; — 1
  for(i = 0; i < n; i++) — n + 1. [can ignore this one]
    s = s + A[i]; — n.
  return s; — 1
}

first see limit
for of repeation of statement (freq count)
cond^n will be checked n+1.

{i = 0 → 1, i++ → n, cond → n+1}

f(n) = 2n + 3 ⟶ T = O(n)

Space
A — n words. (Vector)
n — 1
s — 1
i — 1.

S = O(n) → approx / rough answer.

Q. Algorithm Add(A, B, n)
{
  for i = 1 to n do — n+1
  {
    for j = 1 to n do — n(n+1). → imp statement.
    {
      c[i, j] = A[i, j] + B[i, j] — n².
    }
  }
}

f(n) = 2n² + 2n + 1.

[Write n under for.

$T = O(n^2)$ $f(n) = 2n^2+2n+1$.

$S = O(n^2)$

Q2. Algorithm(A, B, n)
Begin:
n+1 — for i = 1 to n do
(n+1)n — for j = 1 to n do
n×n — c[i, j] = 0;

(n+1)n×n — for K = 1 to n do
n×n×n — c[i, j] = c[i, j] + A[i, K] * B[K, j];

end for.
end for
end for
end.

1. A — n^2 i — 1
B — n^2 j — 1
C — n^2. n — 1

$f(S) = 3n^2+3$, $S = O(n^2)$.

2. $f(n) = 2n^3+3n^2+2n+1$.

$T = O(n^3)$

$S = O(n^3)$

Q for (i=0; i<n; i++)
{ stmt; ——— n. O(n).
}

Q for (i=0; i<n; i++)
{ for (j=0; j<n; j++) O(n²)
{ stmt; [nested loop]
}
}

Q for (i=0; i<n; i++) — n+1
{ stm 1; ——— n
}
for (j=0; j<n; j++) — n+1
{ stm 2; ——— n
}. O(2n) = O(n).

↳ f(n) = 2n+2.
[Independent loops]

Q for (i=0; i<n; i++). — O(n).
for (i=n; i>0; i--); — O(n).
for (i=0; i<n; i=i+2); — $O(n/2) = O(n)$

for (i=0; i<n; i=i+500) —— $O(n/500) = O(n)$

```
count=0;
for(i=1; i<=n; i++)
{
  for(j=1; j<=i; j++)
  {
    count++;
  }
}
```

value of count?

Procedure (Approach)

$i=1 \longrightarrow 1$
$i=2 \longrightarrow 2$
$i=3 \longrightarrow 3$
$\vdots$
$i=n \longrightarrow n.$
$\therefore \Sigma n.$

i	j	count
1	1	1
1	2	X
2	1	1
2	2	2
3	1	X
3	1	1
3	2	2
3	3	3
3	4	X

1 — 1
2 — 2
3 — 3
4 — 4
n — n

$\therefore 1+2+3 \cdots n$

$\frac{n(n+1)}{2} \Longrightarrow O(n^2)$

$T=O(n^2)$

```
count=0;
for(i=1; i<=n; i=i*2)
{
  count++;
}
```


i	count
1	1
2	2
4	3
8	4
16	5 →
n^k	$\log_2(k)+1 = K$

$2^K = 1 = n.$

$K = \log_2(n)+1$

$\Rightarrow O(\log n).$

$T=O(\log n)$

```
count=0
for(i=n; i>0; i=i/2)
{
  count++;
}
```

	i	count
$n/2^0$	n	1
$n/2^1$	n/2	2
$n/2^2$	n/4	3
$n/2^3$	n/8	4
	⋮	K-1
$n/2^{K-1}$	1	K

$\frac{n}{2^{K-1}} = 1 \qquad \therefore 2^{K-1} = n.$

$\Rightarrow K = \log n + 1$

$\therefore T=O(\log n)$

→ Asympto = log n.

Q. `for(i=1; i<n; i=i*2)`
{ stmt; }

i	1	2
	1	2

How many times condⁿ will be checked?
⇒ log n + 1.

i	count
1	1
2	2
~~4~~ 3	~~4~~ 3
8	4
16	5
32	X. +1

i.e. n=20.

$\lfloor \log_2 20 \rfloor + 1$

⇒ $\lceil \log n \rceil + 1$

n=6.

i	stm
1	✓
2	✓
4	✓
8	X

$\log_2 6 = \lceil 2.2 \rceil = 3 + 1$

condⁿ checking !!

Q. `for(i=0; i<n; i++)` — O(n).
`for(i=n; i>0; i--)` — O(n).
`for(i=1; i<n; i=i*2)` — $O(\log_2 n)$
`for(i=n; i>0; i=i/2)` — $O(\log_2 n)$

+/- : n.
*/ / : log n

(Observation).

Q. `for(i=1; i<n; i=i*5)` — $O(\log_5 n)$.
`for(i=1; i<n; i=i*3)` — $O(\log_3 n)$

`for(i=n; i>1; i=i/3)` — $O(\log_3 n)$.

`for(i=0; i<n; i++)` — n — n+1
{
`for(j=1; j<n; j=j*2)` — log n
{ stmt; } → n log n.
}

`for(i=0; j=n-1; i<=j; i++, j--)`
{ }. → O(n/2) = O(n)

`for(i=1, j=n; i<=j; i=i*2; j=j/2)`
→ $O(\log_2 n)$.
[From both sides, behaviour is log n]

Q for(i=1; i<logn; i++) - O(log n)

Q for(i=1; i<sqrt(n); i++) — O(√n)

Q for(i=1; i*i<n; i++) — O(√n)

Q for(i=1; i<loglogn; i++) — O(loglogn)

Q for(i=1; i<logn; i=i*2) — O(loglogn)

Q for(i=1, j=n; i<=j; i=i*3, j=j/2)

→ O(log₂ n) ∵ log₂ n > log₃ n

→ Upper Bound.
[can give max this time]

Q
```
k=0;
for(i=0; k<n; i++)
{
  k=k+i;
} Stmt;
```

n=6 4 6.

i	k
0	0
1	1
2	3
3	6 X

n=12.

i	k
0	0
1	1
2	3
3	6
4	10
5	15 X

Appr: (condn to be checked.)

k	i
0	0
0+1	2
0+1+2	3
0+1+2+3	4
0+1+2+3+4	5
⋮	⋮
1+2+3+····p = n.	p p+1.

→ Code will execute for p times.

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

$T = O(\sqrt{n})$ [Upper Bound]

* Asymptotic Notations

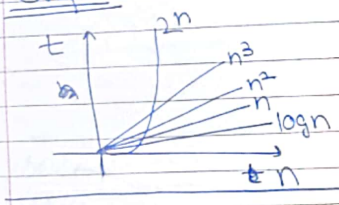
[1 < logn < n < nlogn < n^2 < n^3 < ··· < 2^n < 3^n < ···· n^n]

→ Possible time complexities.

* 1 — constant.
  log n — logarithmic
  n — linear
  n^2 — quadratic
  n^3 — cubic
  2^n — Exponential.

Growth of function.

Graph:



n^3	2^n
125	32

n=5.

n	n^3	2^n
3	27	8
5	125	32
9	721	512
10	1000	1024
11	1331	2048

$2^n >= n^3 , \forall n >= 10$

only put bigger values.

$n^3 > 2^n$ $n^3 < 2^n$

0 1 ∞

2^n always greater than n^3

Q n^{100} & 2^n.

⇒ 2^n always greater.

$n^K < 2^n$

K-degree exponential

- $f(n) = 3n+5$: Linear —— $O(n)$.
- $f(n) = 500n+9000$: Linear — $O(n)$.
- $f(n) = 3n^3+4n^2+5n$ — Cubic $O(n^3)$.
- $f(n) = 5\sqrt{n}+3$ ——

b/w log n & n.

$\Omega(\log n)$ lower Bound. $O(n)$. upper bound.

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

Asymptotic.

- $f(n) = n! + 4$

lower bound $\Omega(1)$. ✓

upper bound $O(n^n)$.

known As lower.

$\Omega(n)$ ✓

[till all functn, till now Θ came]. (even $\sqrt{n}$).

Q $f(n) = \log n!$

- $\Omega(1)$ [Lower]
- $O(n \log n)$ [upper bound] . n^2

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

- $\Omega(n)$ [lower than/ eq to]
- $\Theta(n)$ [when UB = LB].
- $O(n)$ [greater than/ eq to].

Q $f(n) = 2n^2 + 2n + 1$

- $\Omega(n^2)$
- $\Theta(n^2)$.
- $O(n^2)$

$f(n) = 2n^3 + 3n^2 + 2n + 1$.

- $\Omega(n^3)$ [LB]
- $\Theta(n^3)$ [Avg Bound] [tight Bound].
- $O(n^3)$ [UB].

* We come across 'big O' notation most [max the time]

Imp

Q $f(n) = 2n+3$	UB	LB
	$O(n)$ ✓	$\Omega(n)$ ✓
	$O(n^2)$ ✓	$\Omega(\log n)$ ✓
	$O(n^3)$ ✓	$\Omega(1)$ ✓
	$O(\log n)$ ✗	$\Omega(n^2)$ ✗

∴ $O(n)$ → Tighest upper Bound
$\Omega(n)$ → Tightest lower Bound
→ useful.

* Time taken by Linear Search is f(n) and: (GATE) → f(n)

1. $f(n) = O(2^n)$
2. ✗ $f(n) = O(\log n)$. → false.
3. $f(n) = \Omega(1)$.
4. $f(n) = O(n^2)$.

① Big O → upper Bound.

$f(n) = O(g(n))$ if & only if, there exists +ve constants c & n_0, such that:

$f(n) \leq c * g(n) \ \forall n \geq n_0$ → SV. (g(n) → some functn.)

eg: $f(n) = 2n+3$.

$2n+3 \leq 5n^3 / 1n^2 / 4n^8 \ldots$

$2n+3 \leq 2n + 3n$.

$2n+3 \leq 5n \ \forall n \geq 1$.

$f(n) \leq c\,g(n)$ → shud be only 1 term.

$f(n) = O(n)$

eg: $f(n) = 2n+3$

$2n+3 \leq 5n^2 \quad \forall n \geq 1$.

$f(n)$ c $g(n)$.

$f(n) = 2n+3 = O(n^2)$ [not tightest]

Q $f(n) = n!$

$1 * 2 * 3 * \cdots * n \leq 1 * n^n \quad \forall n \geq 1$.

$n! \leq 1 \cdot n^n \quad \forall n \geq 1$.

$O(n!) = n^n$

② Omega

$f(n) = \Omega(g(n))$ iff there exists c & n_0, such that $f(n) \geq c * g(n) \ \forall n > n_0$.

eg: $f(n) = 2n+3$

$2n+3 \geq 1 * n \quad \forall n \geq 0$.

$\therefore f(n) = 2n+3 = \Omega(n)$.

Q $f(n) = n!$ → Subs. all 1 [Highest degree term].

$1 * 2 * 3 \cdots n \geq 1 * 1 * 1 \cdots 1$

$n! \geq 1$.

$n! = \Omega(1)$

③ Average

$f(n) = \Theta(g(n))$ iff there exists 3 constants c_1, c_2, n_0 such that:

$c_1 * g(n) \leq f(n) \leq c_2 * g(n) \ \forall n > n_0$

eg: $f(n) = 2n+3$.

$1 * n \leq 2n+3 \leq 5 * n$.

c_1 $g(n)$ $f(n)$ c_2 $g(n)$

$\therefore f(n) = 2n+3 = \Theta(n)$

like,

$1 \leq n! \leq n^n$.

$g(n)$ $\times$ $g(n)\times$.

$\therefore$ avg. bound one bar !

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

$n \le f(n) \le n^2$ ✗

[write TUB & TLB to find θ]

* Properties of Asymptotic N

1. $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$
   ↳ UB
2. if $f(n) = O(n)$ then: a = constant
   $a * f(n) = O(n)$.
3. if $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$
   then $f(n) = \theta(g(n))$
4. if $f(n) = O(g(n))$ & $g(n) = O(h(n))$
   then: $f(n) = O(h(n))$.

Old Gate Q:

5 if $f(n) = O(g(n))$ & $d(n) = O(e(n))$,
then:
(i) $f(n) + d(n) = O(\max(g(n), e(n)))$
(ii) $f(n) * d(n) = O(g(n) * e(n))$

* Comparison of functions

$f(n) = n^2$; $g(n) = n^3$ $g(n) > f(n)$

↳ 2 methods:

(1) Sampling (min 3 samples).

(2) Applying log on both sides.

$\log(n^2)$ $\log(n^3)$

$2\log n + 5 < 3\log n$

↑ const term can be ignored not const coefficient.

Q $f(n) = 2^{\sqrt{n}}$ $g(n) = n^{\log n}$ $f(n) > g(n)$

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

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

$\log(\sqrt{n}\log 2)$

$\log 2 \cdot \frac{1}{2}\log n$ $2\log(\log n)$

$\Rightarrow \log 2^{\sqrt{n}}$ $\log n^{\log n}$

$\Rightarrow \sqrt{n}$ $\log n \log n$

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

$\Rightarrow \log\sqrt{n}$ $\log(\log n)^2$

$= \frac{1}{2}\log n > 2\log(\log n)$

① $f(n) = n^{\sqrt{n}}$, $g(n) = 2^{\log n}$

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

$\Rightarrow \sqrt{n}(\log n) > \log n$

$f(n) > g(n)$

② $f(n) = n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log n}$

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

$f(n) = g(n)$

③ $f(n) = n^2 \log n$, $g(n) = n \log^{10} n$

$\log(n^2 \log n)$ $\log(n \cdot (\log n)^{10})$

$\log(n^2 \log n)$ $\Rightarrow \log n + 10 \cdot \log(\log n)$

$2\log n + \log(\log n)$ $\log n + 10 \log(\log n)$

$f(n) > g(n)$

④ $g_1(n) = \begin{cases} n^3, & n \le 10{,}000 \\ n^2, & n > 10{,}000 \end{cases}$

$g_2(n) = \begin{cases} n, & n \le 100 \\ n^3 & n > 100 \end{cases}$

$g_2 > g_1$

① $\log ab = b \log a$

② $a^{\log_c b} = b^{\log_c a}$

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

[∵ b4 log, dant da exact comparison]

Q $f(n) = 3n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log n}$

[Same]

after 10,000; g_2 is always greater.

* Imp Topic :

best case — Ω → LB of function

worst case — O → UB of function

no relation b/w case & notation

worst: $f(n) = n$ — Linear Search

$\Omega(n)$ $\Theta(n)$ $O(n)$

function ka Bounds

best $f(n) = 1$.

$\Omega(1)$ $\Theta(1)$ $O(1)$

* Binary Search tree

- Best case : element found in root

  Best case Time : $O(1)$.

- Worst case : element found in leaf Node

  Worst case Time : min $O(\log n)$, max $O(n)$.

  → can have 2 times .

  $O(h)$

Q What is the min worst case time for Searching in BST ?

→ $O(\log n)$.

Q What is max worst case time for LS in Array ?

→ $O(n)$.

Lecture 8

* Time Complexity of recursive funct

```
void fun(int n) — T(n)
{
  if(n>0) — 1
  {
    pf(n); — 1
    fun(n-1); — T(n-1).
  }
}
```

1) decreasing funct
2) dividing funct.

Q

```
void fun(A, n, Key)
{
  return Linear Search(A, n, Key); — n
}
```

Time taken by LS.

$O(n) = T$

- for recursive funct, we use T(n) [not f(n)]

$T(n) = T(n-1) + 1$ → Time funct become recursive not Polynomial.

→ Recurrence relation : asymptotically.

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

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

asymp.

- Recursion must have a stopping condition else stack overflow.

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

1) Tree Method:

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

$T(n-2) + 1$

$T(n-3) + 1$

$T(n-K) + 1$

Assume: $n-K=0$ $\quad T(0) + K$

$n=K$.

$\boxed{T(n) = 1+n}$

2) Back Substitution Method (Induction)

$T(n) = T(n-1) + 1$ —— ①

Substitute $T(n-1)$ in ①.

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

$\Rightarrow T(n) = T(n-2) + 2$ —— ②

Substitute $T(n-2)$ in ②.

better write 1+2 instead of adding.

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

$\Rightarrow T(n) = T(n-3) + 3$ —— ③

Continue for K times.

$T(n) = T(n-K) + K$ —— ④

Assume, $T(n-K) = T(0)$ $\quad \therefore n-K=0$

$n=K$.

$\boxed{T(n) = 1+n}$ $\longrightarrow \Theta(n)$

Q

```
void fun(int n)
{
  if(n>0)                    ——— 1.
  {
    for(i=0; i<n; i++)       — n+1
    {
      pf(i);                 — n
    }
    fun(n-1);                — T(n-1).
  }
}
```

$T(n) = n + T(n-1)$ $\longrightarrow$ RR.

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

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

1) Tree

$T(n-1)+n$

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

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

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

$\Rightarrow n-K+1+n-K+2+\cdots n-2+n-1+n.$

Assume $n-K=0$ & $n=K$.

$1+2+\cdots n$

$\Rightarrow T(n) = n(n+1)/2.$

$\Theta(n^2)$.

2) Induction

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

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

Assume that $T(n-K) = 0$; $n=K$,

$T(n) = T(0)+[1+2+3+\cdots n-1+n]$

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

$\therefore \boxed{T(n) = \Theta(n^2)}$

ii) SC: $n=10, n>0$.

Assume; $n-K=10$, $n=K+10$

Substitute in next step:
[can't be really solved].
→ we will still assume 0.
[But, in the end, use 0 notation].
(approx).

```
1) void fun(int n)
{
  if(n>0)
  {
   for(i=0; i<n; i++)
   {for(j=0; j<n; j++)
    {
     pf(i,j);
    }}
   fun(n-1);
}
```

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

$\Theta(n^3)$.

Q
```
void fun(int n)
{
  if(n>0)
  {
    for(i=1; i<n; i=i*2)
      pf(i);
    fun(n-1);
  }
}
```

→ $T(n) = T(n-1) + \log n$

$\Theta(n \log n)$

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

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

$\Rightarrow T(n-2) + \log(n-1) + \log n$

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

$\Rightarrow T(n-3) + \log(n-2) + \log(n-1) + \log n$

$n-(k-1)$

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

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

$\Rightarrow 1 + \log n^n$

$\Rightarrow 1 + n \log n$

$T(n) = O(n \log n)$

Gate

Q $T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + n^2 \longrightarrow O(n^3)$

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

$T(n) = T(n-5) + \sqrt{n} \longrightarrow O(n\sqrt{n})$

$T(n) = T(n-50) + n \log n \longrightarrow O(n^2 \log n)$

$T(n) = T(n-15) + 5n \longrightarrow O(n^2)$

Q
```
void fun(int n)
{
  if(n>0)
  {
    pf(n);
    fun(n-1);
    fun(n-1);
  }
}
```

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

$$T(n) = \begin{cases} 1 & \text{, if } n=0 \\ 2T(n-1)+1 & \text{, if } n>0 \end{cases}$$

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

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

$\Rightarrow 4T(n-2) + 2 + 1$

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

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

$\Rightarrow 2^2 T(n-2)+2+1$

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

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

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

$n-K=0$ $\qquad 1(2^K-1)$

$\Rightarrow 2^n \times 1 + 2^K - 1$ $\qquad \frac{a(r^n-1)}{r-1}$

$\Rightarrow 2^n + 2^n - 1$

$\Rightarrow 2^{n+1}-1$

$\boxed{\theta(2^n)}$

```
if(n>0)
{
  pf(n);
  fun(n-1);
  fun(n-1);
  fun(n-1);
}
```

$\theta(3^n)$

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

- $T(n) = 2T(n-1)+1$ —— $\theta(2^n)$
- $T(n) = 3T(n-1)+1$ —— $\theta(3^n)$
- $T(n) = 2T(n-1)+n$ —— $\theta(n2^n)$
- $T(n) = T(n-1)+T(n-2)+1$ —— $O(2^n)$

↳ Fibonacci Series.

Gen. form of recurrence eqⁿ, we write:

⇒ Decreasing functn ka Master Theorem:

↳ $T(n) = aT(n-b)+O(f(n))$

✓ Case 1: if $a=1$, then $O(n*f(n))$

* Case 2: if $a>1$, then $O(a^n * f(n))$.

↳ Substitute to check, better. (∵ classroom result).

* DIVIDING FUNCTIONS

```
void fun(int n)
{
  if(n>1)
  {
    pf(n);
    fun(n/2);
  }
}
```

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

$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1, & n > 1 \end{cases}$ ← write 1 to solve easily

[Assumption will change]

$T(n) = T(n/2) + 1$ ✓

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

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

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

$\vdots$

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

Assume: $T\left(\frac{n}{2^K}\right) = T(1)$

$\therefore n = 2^K$

$K = \log n$

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

$\therefore \theta(\log n)$

Page No. 257

Q.
```
void fun(int n)
{
  if (n > 1)
  { for (i = 0; i < n; i++)
    {
      pf(i);
    }
    fun(n/2);
  }
}
```

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

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

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

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

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

$\vdots$

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

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

$T(n) = 1 + n\left[\underbrace{1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{K-1}}}_{\text{Infinite GP}}\right]$

$\frac{a}{1-r}$ ← $\frac{1}{1-\frac{1}{2}} = 1$

$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \infty = 1$

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

$\Rightarrow 1 + 2n$

$\Theta(n)$

Q. Void fun(int n)
{
if(n>1)
{
pf(n);
fun(n/2);
fun(n/2);
}
}

1. $2^K - 1$

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

$\Rightarrow 2[2 \cdot T(n/2^2) + 1] + 1$

$T(n) = 2^2 T(n/2^2) + 2 + 1$

$\Rightarrow 2^2[2 \cdot T(n/2^3) + 1] + 2 + 1$

$\Rightarrow 2^3 T(n/2^3) + 2^2 + 2 + 1$

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

$\Rightarrow 2^K T(n/2^K) + 2^K - 1$

$\Rightarrow$

$n/2^K = 1 \quad (\log n) = K$

$2^n + 2^n - 1$

$\Rightarrow 2 \cdot 2^n$

$\Rightarrow 2^K \cdot T(1) + 2^K - 1$

$\Rightarrow n + n - 1 \quad O(n)$

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

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

$T(n) = 2^K T(n/2^K) + Kn$

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

$\boxed{T(n) = \Theta(n \log n)}$

Q. $T(n) = 2T(n/2) + n^2$

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

$\Rightarrow 2^3 T(n/2^3) + \frac{n^2}{2^2} + \frac{n^2}{2} + n^2$

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

$\Rightarrow n * 1 + n^2\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{K-1}}\right]$

$f(n) = n^K \log^p n$

$n + 2n^2 \Rightarrow \Theta(n^2)$ ✓

* MASTER'S THEOREM (for Dividing func)

General form of recurrence relation:

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

Case 1: if $\log_b a > K$, then $O(n^{\log_b a})$

eg: $T(n) = 2T(n/2) + 1$

$a=2, b=2, f(n)=1 \quad \therefore K=0$.

$\log_2 2 > 0$

$\therefore T(n) = O(n^{\log_2 2}) = \underline{O(n)}$

H/W:
- $T(n) = 4T(n/2) + n$ — $O(n^2)$
- $T(n) = 8T(n/2) + n$ — $O(n^3)$
- $T(n) = 8T(n/2) + n^2$ — $O(n^3)$
- $T(n) = 5T(n/2) + n$ — $O(n^{\log_2 5})$
- $T(n) = 7T(n/2) + n^2$ — $O(n^{\log_2 7})$ ($K=2$)

Case 2: if $\log_b a = K$, then:

$O(f(n) * \log n)$.

eg: $T(n) = 2T(n/2) + n$.

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

$\Rightarrow \log_2 2 = 1 = K$.

$\therefore \boxed{O(n \log n)}$

Case 3: if $\log_b a < K$, then $O(f(n))$.

eg: $T(n) = 2T(n/2) + n^2$.

$a=2, b=2, f(n)=n^2, K=2$.

$\log_2 2 = 1 < 2$.

$\therefore \underline{O(n^2)}$.

Case 2:

if $\log_b a = K$, then:

① if $p \geq 0$ then $T(n) = O(f(n) * \log n)$

② if $p = -1$, then $T(n) = O(n^{\log_b a} * \log\log n)$

③ if $p < -1$, then $T(n) = O(n^{\log_b a})$

9.
10. nlogn.

PRACTISE

① $T(n) = 2T(n/2) + n^2 \log n$ — $O(n^2)$

② $T(n) = 2T(n/2) + n \log n$

③ $T(n) = 9T(n/2) + n^3$

④ $T(n) = 4T(n/2) + n^3$

⑤ $T(n) = 2T(n/2) + \frac{n}{\log n}$ — $O(n \log\log n)$

⑥ $T(n) = 2T(n/2) + \frac{n}{\log^2 n}$ — $O(n)$.

⑦ $T(n) = 4T(n/2) + \frac{n^2}{\log^2 n}$

⑧ $T(n) = T(n/2) + T(n/2) + 1$

⑨ $T(n) = T(n/2) + T(n/3) + 1$

⑩ $T(n) = T(n/2) + T(n/3) + n$ → nlogn.

⑪ $T(n) = T(n/3) + T(2n/3) + n$
→ $O(n \log n)$.

```
void fun(int n)
{
  if (n>2)
  { pf(n);
    fun(√n);
  }
}
```


$$T(n) = \begin{cases} 1 & ; \text{if } n=2 \\ T(\sqrt{n}) + 1 & ; \text{if } n>2 \end{cases}$$ ← can be solved easily.

$\Rightarrow T(n) = T(\sqrt{n}) + 1$.

$T(n) \Rightarrow T(n^{1/2}) + 1$.

$\Rightarrow T(n^{1/2^2}) + 2$.

$\Rightarrow T(n) = T(n^{1/2^3}) + 3$

$T(n) = T(n^{1/2^K}) + K$. $\quad \frac{1}{2^K} = 2$.

Assume $n = 2^m$.

$T(n) = T\left[2^{\frac{m}{2^K}}\right] + K$.

Assume $T\left(2^{\frac{m}{2^K}}\right) = T(2)$.

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

$\therefore m = 2^K \quad \therefore K = \log m$

$T(n) = T(2) + K$.

$T(n) \Rightarrow 1 + \log m$. $\quad \because n = 2^m$.

$m = \log_2 n$.

$\therefore T(n) = \theta(\log\log n)$.

Q
```
void fun(int n)
{
  if(n>2)
  {pf(n);
  fun(√n)+n;
  }}
```
→ $T(n)=T(\sqrt{n})+1$
$O(\log\log n)$.

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

* SORTINGS

Bubble, Selection, Heap, Merge, Quick
Insert, Shell, Count, Bin, Radix.
↳ index based Sorting.

⇒ Rest → Comparison Based Sorting.

• Algorithm Selection based on:

1. Comparisons
2. Swaps.
3. Adaptive
4. Stable.
5. Extra Memory

← Criteria for Analysing Sorting Algorithm

→ If list is already sorted & also taking less time : adaptive.

4. Stable.

duplicate elements in list.

5 8 2 4 8 7 9 10.

Stable: 2 4 5 7 8 8 9 10 ← the original order of the duplicate elem must be preserved.

Unstable: 2 4 5 7 8 8 9 10.

• name →	Anil	Ajay	Bunty	Chandu	Nitin	
• marks →	10	15	12	10	5	
	Sorted +10					

↳ sorted according to name.

• name	R	N	A	C	S	B	A
marks	4	5	10	10	10	12	15

name.

↳ sorted accor to marks.

↳ Stable Sort used.

orderby(marks, name);
↳ first sort by name, then marks.

→ Comparison based Sorting.

Sortings	Compari.	Swaps	Adaptive	Stable	Ext Mem
Bubble	n^2	n^2	✓	✓	✗
Insertion	n^2	n^2	✓	✓	✗
Selection	n^2	n	✗	✗	✗
Heap	$n\log n$	$n\log n$	✗	✗	✗
Merge	$n\log n$	$n\log n$	✗	✓	✓
Quick	$n\log n$	$n\log n$	✗	✗	✗
Shell	$n^{3/2}$ or $n^{5/2}$	$n^{3/2}$	✓	✗	✗
Count	$O(n)$	–	✗	✓	✓
Bin	$O(n)$	–	✗	✓	✓
Radix	$O(dn)$.	–	✗	✓	✓
Index Sort [not used]	No. of digits.				

- Based on comp, HMQ faster.
- Based on Swaps, Selection much faster.
- Only 2 Algo Adaptive: Bubble, Ins.
- [Most used Algorithm: Merge]

MS → works on Array/LL Both.
QS → works only on Array.

Inplace → | 5 | 8 | 6 | 9 | 3 | 2 |

[Array itself be Sorted, only in that array] ⇒ in Place Sorting. (no extra Array).

Merge → not in Place Sorting

* Bubble Sort.

List → 8 5 9

1 Pass.

List					II Pass		
8]	5]	5	5	5]	5	5	5
5	8]	8]	8	8]	8]	4	4
9	9]	9]	4	4	4	8]	3
4	4	4	9]	3	3	3	8
3.	3	3	3]	9	9	9	9

if 1st is greater than interch [i.e. → Worst interchange].

comp = n−2
max = n−2
swaps

Part I

no. of comparisons = n−1

max. Swaps = n−1 [if every element get interchanged].

Total n−1 Passes [in Bubble Sort]

III Pass

5]	4	4
4]	5]	3
3	3	5
8	8	8
9	9	9

II Pass

4]	3
3]	4
5	5
8	8
9	9

← Sorted.

- Why Bubble Sort? → stone in Pond.

→ Heavier element gets set down & bubble will up.

- comp = $(n-1)+(n-2)+\cdots 3+2+1$ (Pass 1, Pass 2).
  $\Rightarrow \frac{n(n-1)}{2} = O(n^2)$.
  Swaps = $O(n^2)$

Code:

```
for(i=0; i<n-1; i++)        → total n-1 Passes.
{
  for(j=0; j<n-i-1; j++)    → comparison in each Pass
  {
    if A[j] > A[j+1]
    {
      swap(A[j], A[j+1]);
    }
  }
}
```

5 7 8 12 16 20 → already Sorted.

adaptive BS. takes O(n) - stop after 1st Pass.

```
for(i=0; i<n-1; i++)
{ flag=0;
  for(j=0; j<n-i-1; j++)
  {
    if (A[j] > A[j+1])
    {
      swap(A[j], A[j+1]);
      flag=1;
    }
  }
  if(flag==0)
    break;
}
```


- will take less time (may take same time) Sorted / Partially Sorted → fast.

* Result of each Pass = largest element at end.

Q largest 3 no. from a list.
→ Perform BS upto 3 Pass.
$3n = O(n)$.

Q 3 largest distinct no. from Array.
→ max n-1 Pass.
$\Rightarrow (n-1)*n = O(n^2)$.

② INSERTION SORT

A	3	7	10	14	18	22	25

- insert an elem in Sorted list in Sorted Pos.

x = 12

(if found an element less/equal, place 1 pass after that).

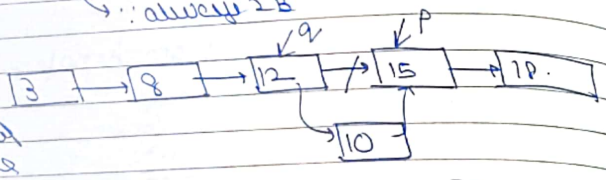
→ Shifting Process tells us the position of element, dont need to find out the position before Hand.

[Shift all the elem, which are greater].

A | 22 | 14 | 7 | 18 | 3 | 10 | 12 |

I, II, III → Insertion Process [till last]

NOTE * data movement is much costlier than pointer movement.
→ depends on DT.
∵ always 2B



[matter of storing info]

Much Faster in LL, than Array.
[Insertion Sort]
→ most Suitable for LL.

⇒ Insertion Sort

A | 8 | 6 | 3 | 5 | 2 |

I^st Pass: insert 6

| 8 | | 3 | 5 | 2 |

6 ↓

6 8 3 5 2.

II Pass: insert 3

6 8 — 5 2

3

3 6 8 5 2

II Pass: insert 5

3 6 8 — 2

5

3 5 6 8 2

IV Pass: insert 2.

3 5 6 8 —

2

2 | 3 | 5 | 6 | 8 | ✓ Observation.

∴ n-1 Pass for n elements,

No. of comparison

Pass 1 — 1.
Pass 2 — 2.
⋮
Pass n-1 — n-1.

∴ $\frac{n(n-1)}{2}$

Time depends on comparison in Comparison based Sorting.

Max swaps = $O(n^2)$ → data movement
Max comparison = $O(n^2)$

Q | 2 | 4 | 7 | 10 | 15 | 19 |

take out 4 compare with 2. & put back.

comp with 5. put back.

∴ 1+1+1+1+1 = 5 comp.

∴ n-1 comparisons → O(n)

if already Sorted.

→ This Sort is adoptive ∵ if already Sorted list → O(n) [less time] (By nature)

Bubble Sort → not adaptive by Nature
Insert Sort → adaptive by nature

Q If IS after 4 passes.

→ we get any smallest/largest.

⇒ we get 1st 4 elements in the Sorted order.

Q Max time by IS = $O(n^2)$

Min time by IS = O(n) [already Sorted]

BS used in IS: Time?

→ Position can be decided pre

- If BS used for finding out position of element in IS. How much time IS taken? → $O(n^2)$.

∴ Shifting takes time.

- IS & BS can replace one another (in Array)
- IS suitable for LL, not Array.

↳ not Strongly dependent on index.

[You can sort LL using IS, without extra mem, just change links]

(3) Selection Sort

1	6 ← i
2	8
3	5
4	4
5	2 ← j
6	7

1. Select i = 1.
2. Find min. element from 1 to 6 as j.
3. Swap a[i] & a[j].

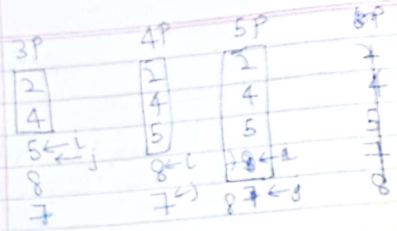
1	2 → smallest element Sorted.
2	8 ← i
3	5
4	4 ← j.
5	6
6	7

P1

Pass 2

1. Select i = 2
2. Find min element from i to 6 as j.
3. Swap a[i] & a[j].

1	2
2	4
3	5
4	8
5	6
6	7



→ Selecting a position & finding an element [fix a pos, & finding the min after that index]

→ n elements → n-1 Passes

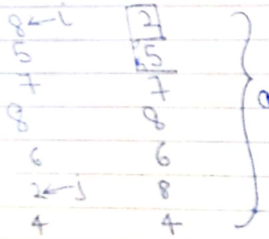
→ Comp → $n-1+n-2+\cdots 3+2+1$

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

→ 1 Swap in each Pass

$\therefore O(n)$

H/W Find out, if SS can't be made adaptive?



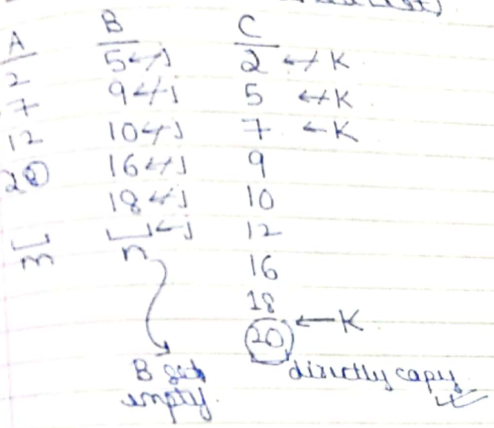
2. If we perform K passes of Selection Sort

→ K min elements in list

Merge Sort

working depend on Merging

Merging (combing 2 Sorted lists into single Sorted List)




```
void Merge(A,B,m,n)
{
  i=1, j=1, k=1;
  while (i<=m && j<=n)
  {
    if(A[i] < B[j])
      C[k++] = A[i++];
    else
      C[k++] = B[j++];
  }
  for(; i<=m; i++)
    C[k++] = A[i];
  for(; j<=n; j++)
    C[k++] = B[j];
}
```

relation for merging.

- Time taken for Merging → $\theta(m+n)$
  - copying [main process in merging]
- Time based on copying

$\theta(n)$ ✓ → Combining 2 SL into single SL

	A	2	4	12	16	3	5	9	15
		1	2	3	4	5	6	7	8
		l			m				h

→ 2 Sorted lists.

B [auxiliary array] (another array must [for merge])

[Merging 2 lists in Array] array.

```
Void Merge(A, l, mid, h)
{
  i=l, j=mid+1, K=l;
  while(i<=mid && j<=h)
  { if(A[i] < A[j])
      B[K++] = A[i++];
    else
      B[K++] = A[j++];
  }
  for(; i<=mid; i++)
    B[K++] = A[i];
  for(; j<=h; j++)
    B[K++] = A[j];
```

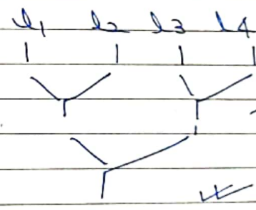
```
for(i=l; i<h; i++)
  A[i] = B[i];
}
```

	l_1	l_2	l_3	l_4
a →	2	3	7	4
	8	5	12	13
	20	9	16	16

→ 4-way Merge.

l_5:
2
3
4
:

* m-way merge – m lists can be merged together.



* Iterative Merge Sort [2-way Merge Sort]

	1	2	3	4	5	6	7	8
A	7	4	9	3	6	5	8	2
I A	4	7	3	9	5	6	2	8
II A	3	4	7	9	2	5	6	8
III A	2	3	4	5	6	7	8	9

(every element is a list) [sorted] ✓

→ auxiliary Array, copied back to A

2 way Merge Sort

[further sorting]

→ light weighted Sort.
→ If add elements, 1 more pass level increases.

∴ log n passes

→ For 1 Pass: n elements (equal) are merged.

∴ MS = n log n

Q 20 elements. After 2 passes, result?
[2-way merging → default.
call funcⁿ (merge) log n times.

MERGE SORT (recursive) → follows divide & conquer Strategy [divide into subproblems]

	1	2	3	4	5	6	7	8
A	8	4	7	5	9	3	6	2

break by MS

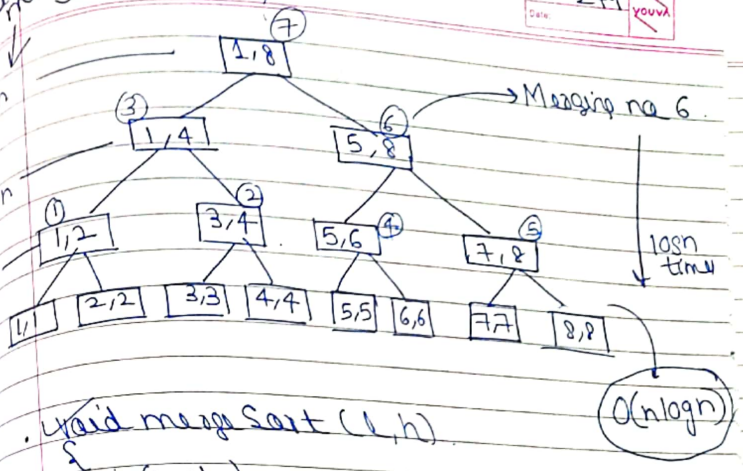
1st merge → ① 4 8 | ② 5 7 | ④ 3 9 | ⑤ 2 6

③ 4 5 7 8 | ⑥ 2 3 6 9 → last order changed.

⑦ 2 3 4 5 6 7 8 9

no. of passes = no. of levels

last merging




```
void mergeSort(l,h)
{
  if(l<h)
  {
    mid = ⌊(l+h)/2⌋;   ——— 1
    mergeSort(l, mid);  ——— T(n/2)
    mergeSort(mid+1, h); ——— T(n/2)
    merge(A, l, mid, h); ——— n
  }
```

→ Pg 276

→ Total 15 funcⁿ calls.
→ Merge (5,6) done in 10th call [10th recursive call]

Q Size of Stack (required for recursion)
→ log n. [ht of tree]

Recurrence Relation

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

$T(n) = 2T(n/2) + n$ ① ①

case 2:

$n \log n$.

NOTE

① for small size lists, MS slow.

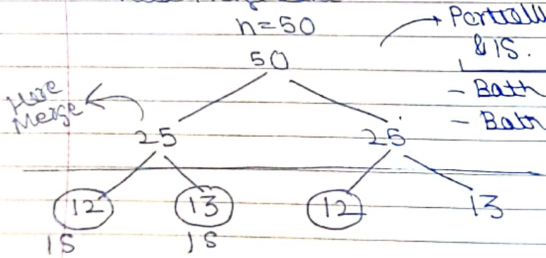
- uses recursion.
- stack usage takes lot of time [Stack creat/dest]

If $n <= 15$, MS is slow → in runtime [than IS, BS, SS].

→ MS best for lot of elements

→ Void MergeSort

n=50



→ Partially Merge & IS.

- Both suitable for it
- Both stable.

→ IS used for making MS better.
∵ Both are stable, ∴ IS used.

```
void MergeSort (l, h)
{
  if (h-l <= 15)
      IS(A, l, h).
  else
  {
    mid = ⌊(l+h)/2⌋;
    MS(l, mid);
    MS(mid+1, h);
    Merge(A, l, m, h);
  }
}
```

② Space used by MS.

Extra Space → $n + \log n$. (Auxiliary Array)

$S(n) = 2n + \log n$.

$O(n)$.

⑤ QUICK SORT

- If an element is at a place, when LHS <=, or RHS >= value, element → Sorted position.

→ works on idea that, element is said to be in sorted position if all the element on LHS is smaller than the element & all the

[In doubly LL → (Height)]
[not restart]

Page No. 282
Date: YOUVA

elements on RHS is greater than element.

40, 30, 10, 20, (50), 70, 90, 80, 60

(50) 40 70 30 20 60 80 10 90
i → j

(50) 40 10 30 20 60 80 70 90
j i
crossed.

[20 40 10 30 (50) 60 80 70 90]
in sorted Position.

pivot: find its sorted Posn.

(50) 60 70 90 80 20 10 30 40 ∞
i
(at pivot) i → in search of greater : [∞]
(at ∞) j → lesser than or equal — in search of.

50 60 70 90 80 20 10 30 40 ∞
i → j

50 40 70 90 80 20 10 30 60 ∞
i j

50 40 30 90 80 20 10 40 60 ∞
i j

50 40 30 10 80 20 90 70 60 ∞
i j

50 40 30 10 20 80 90 70 60 ∞
p
i j i [i crosses].

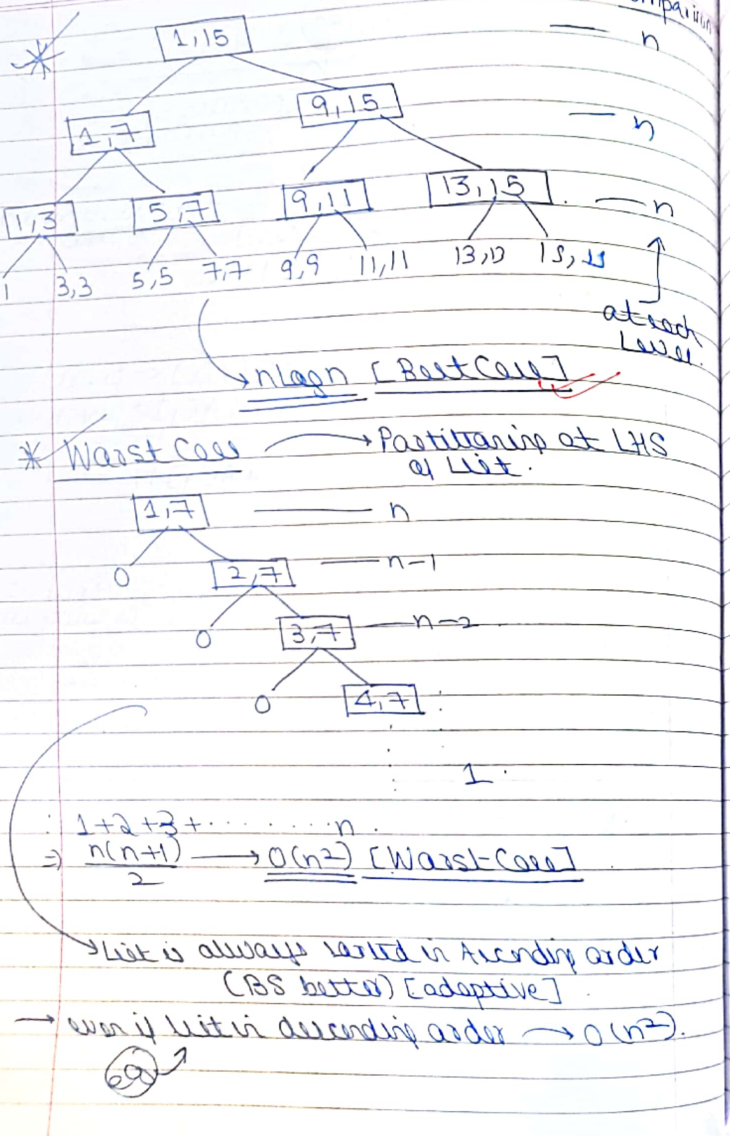
sorted Position.

(20 40 30 10) (50) (80 90 70 60 ∞)
l j [partitioning position] h

```
void QuickSort
{ if(l<h)
  { swap(A[l], A[(l+h)/2]);   → mid element as pivot.
    pivot = A[l]; i=l; j=h;
    do
    {
      do { i++; } while(A[i] <= pivot);
      do { j--; } while(A[j] > pivot);
      if(i<j)
        swap(A[i], A[j]);
    } while(i<j);

    swap(A[l], A[j]);
    QuickSort(l, j);   → j acts as ∞ in lower list.
    QuickSort(j+1, h);
  }
}
```

* Best case of QS

→ Partitioning always done in the middle.



Q. Best case for QS? → eg?
→ when List is partitioned in middle.
Time? → O(n logn)

Q. Worst case for QS? [sorted not a case]
→ when List partitioned [aspiring] ending of List.

10 20 30 40 50 60 70 → when sorted, as acts as slow.

40 20 ~~20~~ 30 30 50 60 70
3
pivot
(Selecting middle when as pivot)
worst → best case.

Q. Selecting middle as pivot ← imp for Gate.

Best Case: List is Partitioned as middle
or T = O(nlogn).
eg: ascending/descending.

Worst Case: List is part. at end.
T = O(n²).
eg: X [exists, but don't know]

* Randomized Quick Sort

- Random element selected as Pivot.
- faster than QS (general).

```
if (l<h)
{
    r = random(l, h);
    swap(A[l], A[r]);
}
```

Case & time → variable
eg → don't know.

* If pivot is always the median, then the partitioning will be in the middle. & thats the best case.

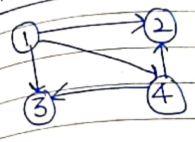
If algo finds median in O(n) time. time taken by QS?

Algo → (n+n) · log n · (Gate Q)
comp. levels.
2n log n
O(n log n)

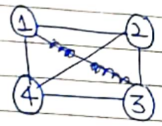
* GRAPH

G = (V, E)

Directed Graph ①



Graph ②



n = |V| = 4, V = {1, 2, 3, 4}
e = |E| = 5, E = {(1,2), (1,4), (2,3), (2,4), (3,4)}

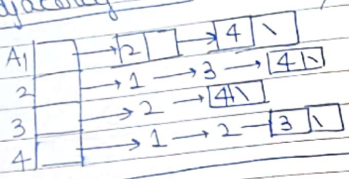
* Graph → Adjacency Matrix.

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array} n \times n$$

→ $O(n^2)$
$O(|V|^2)$

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$ ②

* Adjacency list



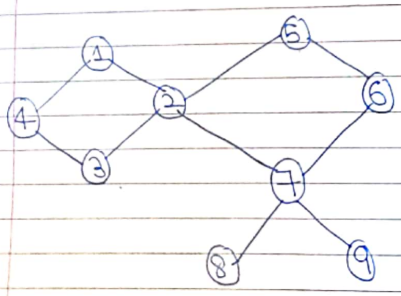
→ O(|V|+|E|)

O(n)

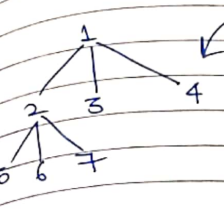
- Algorithm using Adjacency Matrix :
  - max → O(n²).
  - (usually).
  - Time complexity mentioned using matrix.
- Algorithm using Adjacency list.
  - O(n). ✓

* GRAPH TRAVERSAL METHODS

- BFS
- DFS



1) Breadth First Search



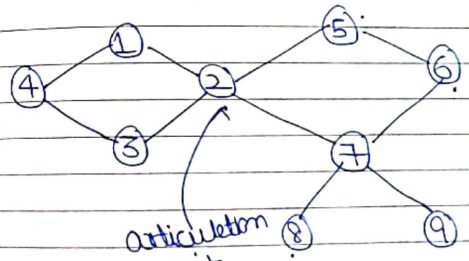
Using tree traversal.
1. Visit Vertex
2. Explore vertex
3. dead Vertex → explore done.
4. Live Vertex → exploring left [after visit 4, 1 dead]. ✓

after 1
↓
4 → 3 → 2 ✓ (order not imp)

- BFS

- if you visit a node, explore it completely, b4 switching to next node.

Q.

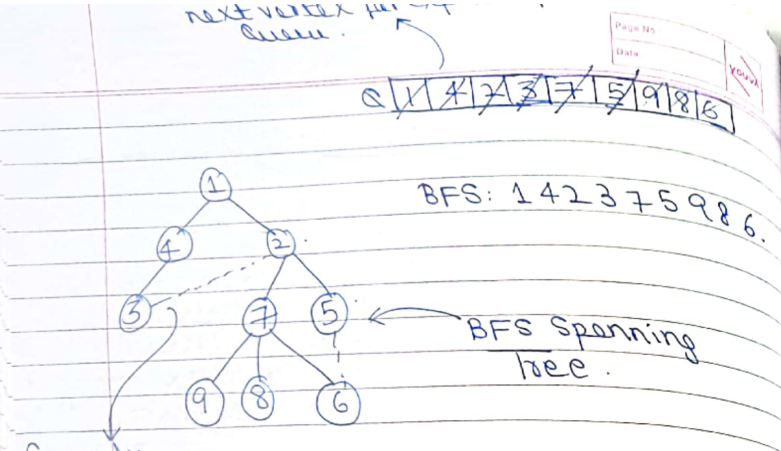


BFS: 14. (if we remove this, Graph divided into pieces).

Q.

next vertex for exploration

Queue:



BFS: 1 4 2 3 7 5 9 8 6.

BFS Spanning tree.

Cross edge [only from n to n+1 level] (not too much). [can be in neighbour level too]

BFS:

→ We can start traversal from any vertex (V).
→ Once a vertex V is visited, insert it in the Queue.
→ Start exploring vertex V by visiting all its adjacent vertices W.
- They can be visited in any order and must be inserted into the Queue.
- Once, V is completely explored, select the next vertex from the queue & start exploring it.

→ Continue, till all the vertices are completely explored.

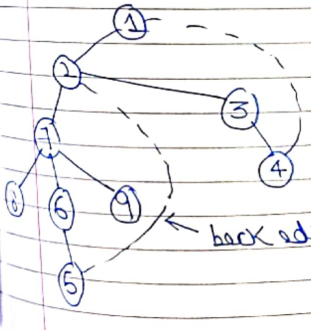
IMP POINTS:
- Next vertex for exploration must be selected from Queue.
- If a vertex is selected for exploration, then it must be completely explored before selecting next vertex.

Page No. 291

1. 1, 2, 4, 5, 7, 3, 6, 8, 9 ✓
2. 2, 5, 1, 7, 3, 6, 4, 9, 8 ✓
3. 5, 6, 2, 7, 3, 1, 8, 4, 9 ✗
4. 2, 7, 3, 5, 6, 8, 9, 1, 4. ✗
5. 7, 6, 2, 8, 9, 5, 3, 1, 4. ✓

DEPTH FIRST SEARCH (DFS)

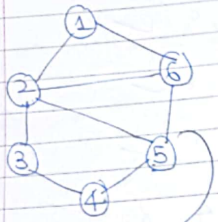
→ Start DFS from any vertex (V).
→ Start exploring V, & visit its adjacent vertices W. (into stack)
→ Once, a new vertex W is visited, suspend the exploration of V, & start exploring W.
- Exploration of V will continue, when W has become dead node (pop).

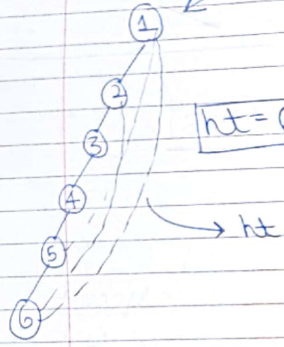


DFS: 1, 2, 7, 8, 6, 5, 9, 3, 4.

6	
7	7 7 3
2	2
1	

Q



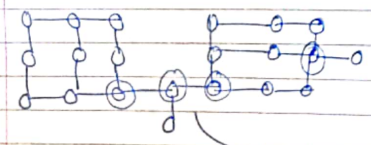


ht = 6 (no. of vertices).

→ ht is less (articulation points ?)

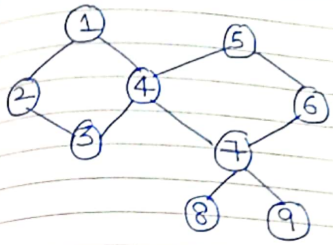
→ ht of DFS tree must be equal to no. of nodes (if less → articulation point present). (multi children due to AP) ~~up 1~~

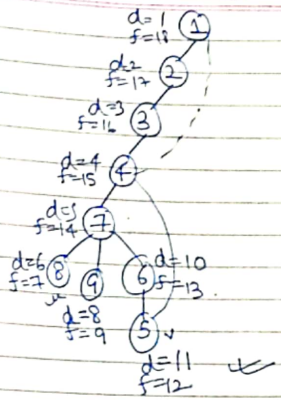
Q



→ 4 Articulation Points.

Q





d() : discovery time.
f() : finishing time.

→ If 2 vertices u & v such
d[u] < d[v] < f[u]
→ discovered after u.
→ discovery of v, before finish of u.
→ v is descendant of u.

Q : Gate :
d[v] > f[u]
→ discovery of v, after finish of u.
v is discovered after finish of u.
→ 2 diff connected component.
8 & 5 / 9 & 6 (imp)

TOPICS :

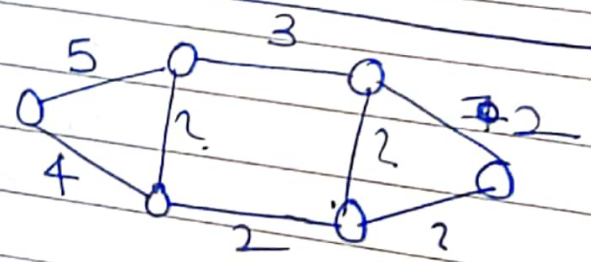
Spanning tree : tree covering (n−1) edges. [sub Graph].
→ prims
Kruskals

→ Select min edge [min Heap]

- follow Kruskal to find
  - Prim → (n^2)
  - Kruskal → $O(n \log n)$.

→ Red-Black Tree.

1.



2.

* Dijkstra
  Bellman Ford } (many Gate Problem)
  → -ve edge ✓
  But not, -ve cycle.