

Structures & Unionsdefining structure:

```
struct book {
    char title[20];
    int pages;
    float price;
};
```

variables:

struct A

{ v1, v2; }

struct A { v1, v2; }

typedef struct:

```
typedef struct {
}; struct-name;
```

Accessing

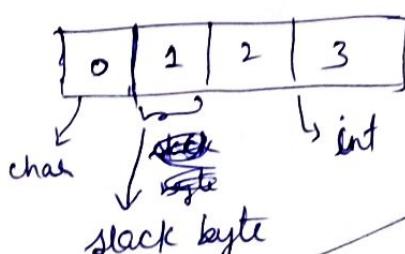
- 1) •
- 2) (\*ptr) • → member
- 3) ptr → q

Copying & Comparing

struct Person P1, P2;

P1 = P2 ✓

P1 == P2 ✗

Word Boundaries & Slack Bytes:-

- \* the last value assigned will be assigned to all the members irrespective of data types

\* UNIONS

→ union item &amp;

```
int m;
float x;
char c;
```

- the compiler allocates a piece of storage that is large enough to hold the largest variable type in the union

`code.m = 379`  
`code.c = 'a';`  
(cout << code.m)  
 gives error because we should  
 make sure that we are accessing the member  
 who value is currently stored.  
 $\rightarrow$  union item       $abc = \{100\}; \checkmark$   
 union item       $abc = \{10.00\}; \times$   
 because during initialisation, we have to use only  
 first member of structure.

### BIT Fields:

struct personal {

  unsigned s : 1

  unsigned age : 7

  unsigned m : 1

  unsigned e : 3

  unsigned : 4

several data items  
packed in a word  
of memory

16 bits

### Structure Sizer:

$\rightarrow$  struct A {

  int a;

  struct A a1;  $\rightarrow X$  (invalid)

```

struct A {
    int a;
    float b;
};

struct A *a;
} b1;

```

Solve → F1

$\rightarrow \text{size of } (*a) = 8 \text{ bytes}$   
 $\rightarrow \text{size of } (b1) = 16 \text{ bytes}$ .

## Pointers

→ derived data type

→ contains memory addresses as their values

~~\* Imp~~ pointers can be used to access and manipulate data stored in the memory

\*) return multiple values from a function

\*) references to functions & thereby facilitating

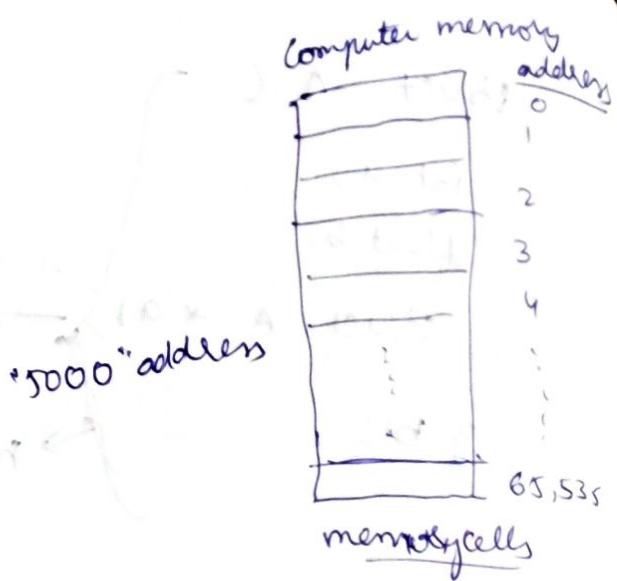
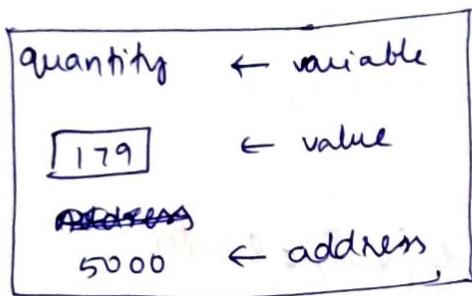
\*) references to functions & thereby facilitating passing of functions as arguments to other functions

\*) support dynamic memory management

\*) efficient tool for manipulating data structures such as structures, linked lists, queues, stacks, trees.

\*) Reduce length & complexity of programs and thus reduce execution time

\*) increase the execution speed and thus reduce the program execution time



- associates "quantity" with "5000" address like "house" & "house no"
- can access "179" either with the name 'quantity'

(\*) memory address.

↓  
assigned to variables

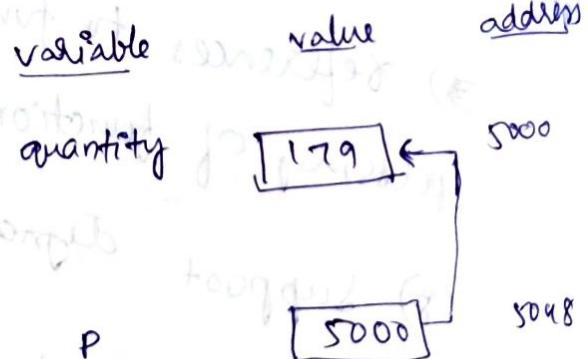
↓  
such, variables that hold memory addresses are

called pointer variables.

int \*p = & quantity;

"p" points "quantity"

↓  
pointer



Pointer constants

Pointer values

Pointer variables

→ memory address  
→ will not change  
we should use  
them

used, storing

value obtained by  
using '==' operator

Accessing the address of available:

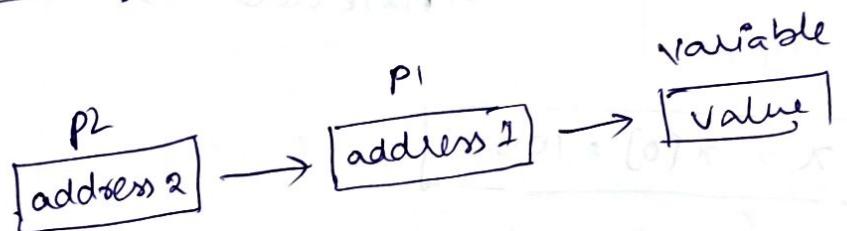
\*& variable name

Declaring Pointer Variables:

datatype \*ptrname;

e.g. int \*a;

Chain of Pointers:



"multiple indirections"

main() {

int x, \*p1, \*\*p2;

x = 100;

p1 = &x;

p2 = &p1;

printf("%d", \*\*p2); // output 100

Pointer Increments and Scale Factor :-

p1 = 2800

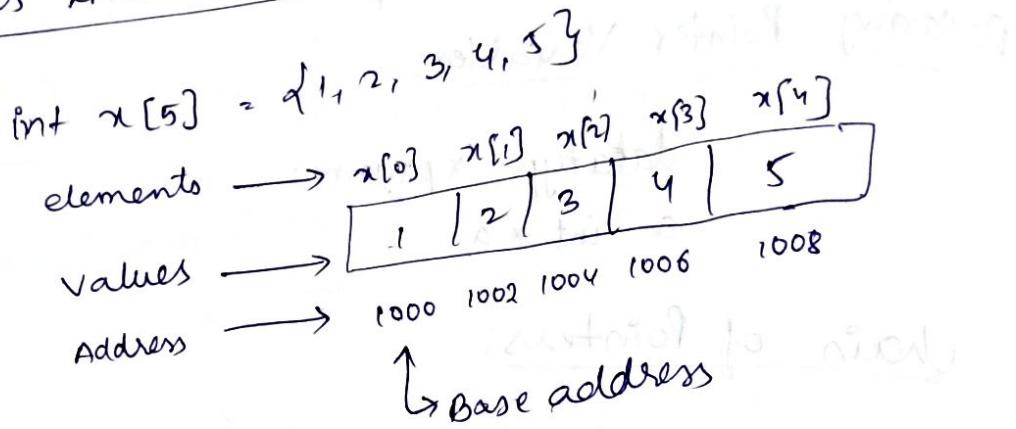
(p1 + 1)  $\Rightarrow$  2804  
not 2801



p1++  $\Rightarrow$  2804

p1 + 2  $\Rightarrow$  2808 not 2802

## Pointers And Arrays



\*  $x = &x[0] = 1000$

Ex:-

int \*p;

$p = &x; \equiv p = &x[0]$

Now,

$p = &x[0] (= 1000)$

$p+1 = &x[1] (= 1002)$

$p+2 = &x[2] (= 1004)$

$p+3 = &x[3] (= 1006)$

$p+4 = &x[4] (= 1008)$

address of  $x[3] = \text{base address} + (3 \times \text{scale factor of int})$

$$1000 + (3 \times 2) = 1006$$

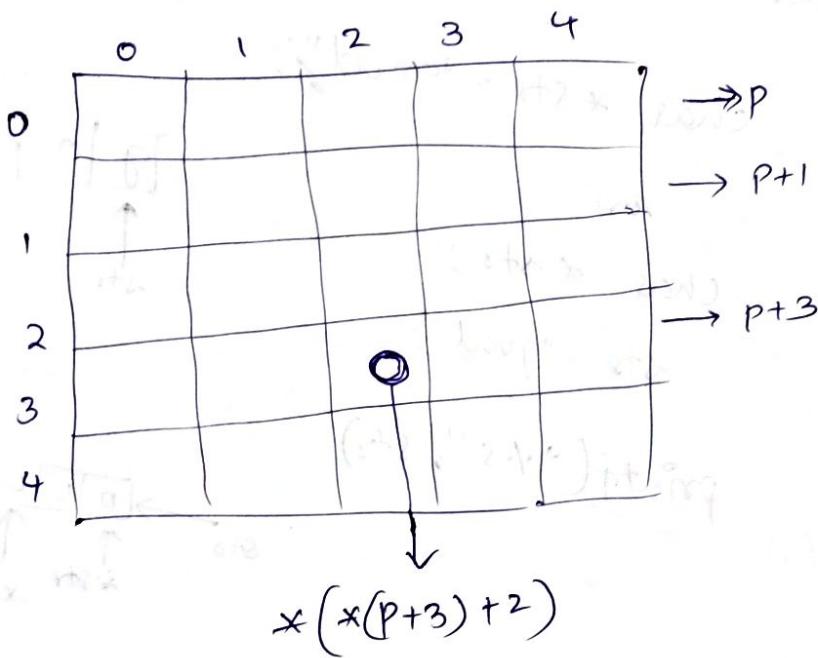
⇒  $\boxed{* (p+3)}$

## Two Dimensional Array

for 1D - array =  $\&(x+i)$  or  $\&(px+i)$

for 2D - array =  $\&(*(*x+i)+j)$  or

$\&(*(*px+i)+j)$



$p \rightarrow$  pointer to first row

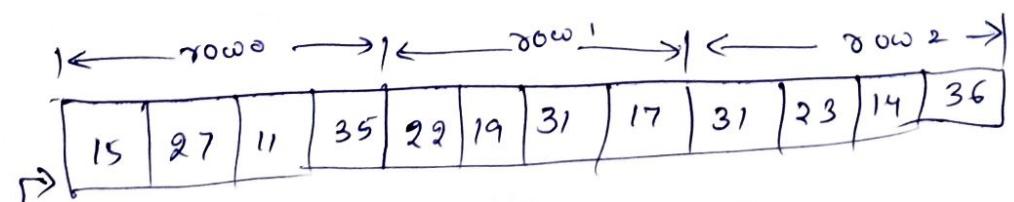
$p+i \rightarrow$  pointer to  $i^{th}$  row

$\&(p+i) \rightarrow$  pointer to first element in the  $i^{th}$  row

$\&(*p+i)+j \rightarrow$  pointer to  $j^{th}$  element in the  $i^{th}$  row

$\&(*(*p+i)+j) \rightarrow$  value stored in the cell  $(i,j)$

```
int a[3][4] = { {15, 27, 11, 35}, {22, 19, 31, 17},
                {31, 23, 14, 36} };
```



address =  $4a[0][0]$

→ compiler does appropriate mapping

## Pointers and character strings

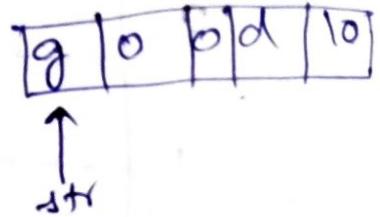
char str[5] = "good";  
 automatically ~~10~~ is inserted at the end of string  
 by compiler

char \*str = "good";

(or)

char \*str;

str = "good";



printf("%s", str)



main() {

char \*str;

str = "delhi";

while (str != '\0') {

printf("%c is stored at %d", \*str, str);

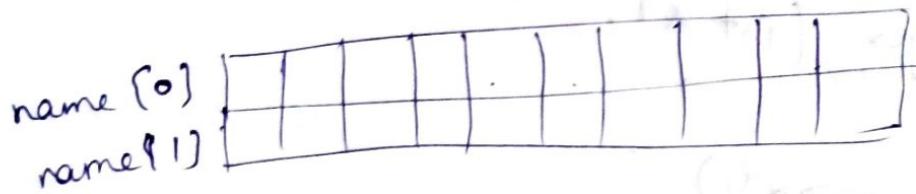
str++;

}

}

## Array of Pointers:-

char name [2][10];



char \*name[2] = { "India", "Australia" };

name[0] → India

name[1] → Australia

name[0] → [I | n | d | i | a]

name[1] → [A | u | s | t | r | a | l | i | a]

```
for(i=0; i<=2; i++)  
printf ("%s\n", name[i]);
```

\* (name[i]+j)

\* p[3]  
↳ an array of 3 pointers

(xp)[3]  
↳ pointer to an array of three elements  
because      '\*' has low precedence

## Functions Returning Pointers

(int \*larger (int \*x, int \*y)) {

if (\*x > \*y)

return (x);

else

return (y);

}

## Pointers To Functions

→ constant.

$$(*\text{ptr}) \cdot \text{age} = 63$$

(or)

$$\underline{\text{ptr} \rightarrow \text{age} = 63}$$

---

int \*a = ~~int~~ (int \*) malloc (10 \* sizeof(int));  
cin >> \*a + i;

int \*\*a = (int \*\*) malloc (10 \* sizeof(int \*));

for (i = 0; i < 5; i++)  
 \*(\*a + i) = (int \*) malloc (10 \* sizeof(int));  
 for (j = 0; j < 2; j++) { cin >> \*(\*a + i) + j; } }.

# DYNAMIC MEMORY ALLOCATION AND

## LINKED LISTS

→ dynamic data structures

with dynamic memory management techniques

→ Process of allocating memory at run time is

known as "Dynamic Memory Allocation"

→ Four library routines known as

Memory Management Functions

↳ allocating & freeing memory during

program execution

### Function

malloc

Allocates request size of bytes and returns a pointer to the first byte of the allocated space

calloc

Allocates space for an array of elements, initialises them to zero and then returns a pointer to the memory

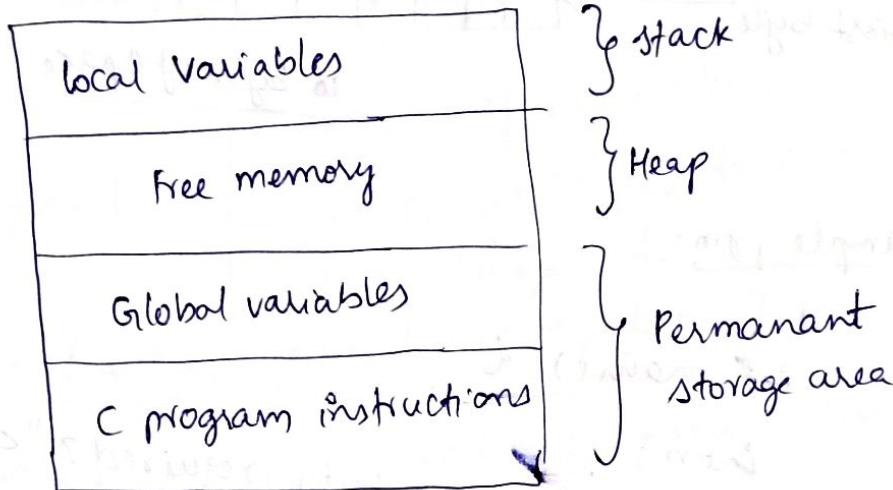
free

frees previously allocated space

### Task

`realloc` modifies the size of previously allocated space

### Memory Allocation Process:



→ size of heap keeps on changing when program is executed due to creation and deletion of variables that are local to functions & blocks

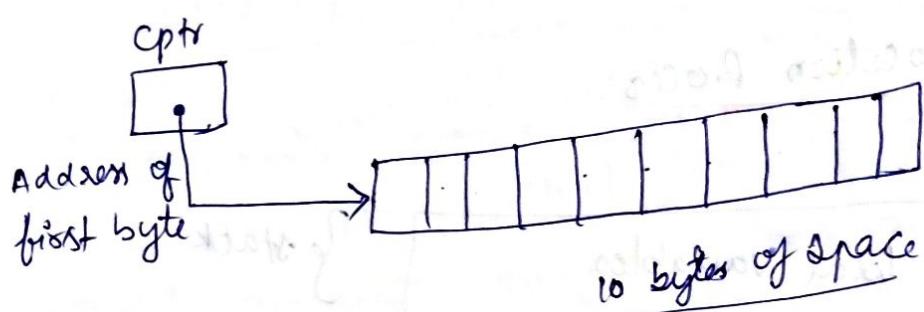
→ possibly encounter "memory overflow" during dynamic allocation. In such situation, function returns a "NULL" pointer

Allocating a Block of Memory: (malloc)

`ptr = (cast-type *) malloc (byte-size);`

`x = (int *) malloc (100 * sizeof(int));`

`cptr = (char *) malloc(10);`



### Sample program:

```
int main()
```

```
{ int n; cout << "no of bytes required?:" << n;
```

```
cin >> n; int * p = (int *) malloc (sizeof (int) * n);
```

```
int * i = p;
```

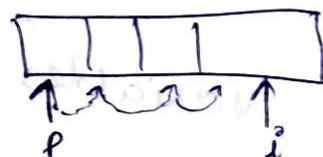
```
for (p=i; p<i+n; p++)
```

```
cin >> *p;
```

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

```
cout << *p;
```

```
}
```

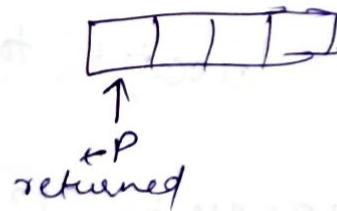


### ALLOCATING MULTIPLE BLOCKS OF MEMORY:

#### CALLOC:

calloc - generally for storing derived data types  
such as arrays and structures

malloc	calloc
→ single block of storage space	→ multiple blocks of storage
→ one argument	→ each of same size
	→ sets all bytes to zero
	→ two arguments
	ptr = (cast-type *) malloc( n, elem-size );
→ contiguous space for 'n' blocks, each of size, elem. size bytes	→ all bytes initialised to zero"
	→ pointer to the first byte of the allocated region is returned



e.g.: struct student {

    };  
    student \* st = (student \*) malloc ( 30, sizeof(st) );

## Releasing the Used Space:

FREE:

`free(ptr);`

- \* It is not the pointer that is being released
  - \*) but rather what it points to
- Ex: `(int *) i = (int *) malloc(sizeof(int)*4);`
- `free(i);`
- i  $\rightarrow$  remains & can be used but, the memory is released
- and no other pointers should point to releasing memory else it shows runtime error
- \*) just release the pointer, to release memory allocated to array of elements

## ALTERING SIZE OF A BLOCK : REALLOC :

`ptr = malloc(size)`

`ptr = realloc(ptr, newsize)`

- ) if additional space needed (or) space to be reduced realloc is used
- ) "Reallocation" is done means, ~~create~~ create the same in an entirely new region and move the contents of old block into the new block

## How to Practice a Question

1. Try to solve the problem on your own

2. Write the code for the algo on paper

3. Test code

4. Type code in comp.

Five steps to a technical question :- (five step approach)

1. Ask your interviewer qstns to resolve ambiguity

2. Design an algorithm

3. Pseudocode first

4. Write your code at moderate space

5. Test code and carefully fix any mistakes

## Five Algorithm Approaches:-

### 1. Exemplify:

Take an example and try to solve using example.

Ex:- Given a time, calculate the angle between

the hour and minute hands.

## 2. Pattern matching :

Under this approach, we consider what problems the algorithm is similar to and try to modify the solution to the related problem to develop an algorithm.

## 3 Simplify & Generalise

### 4. Base Case & Build:

$n=1, n=2, n=3$

### 5 Data Structure Brainstorm

→ check with each ~~base~~ every data structure whether it suits to the problem or not.

### What good code looks like:

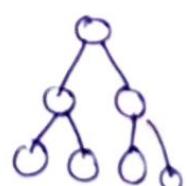
- 1) Appropriate data structure
- 2) Appropriate code reuse
- 3) Modular
- 4) Flexible & Robust
- 5) Error checking [Exceptions]

## Introduction to Heaps

	insert	search	find min	Delete min
unsorted array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Sorted array (ascending)	$O(n)$	$O(\log n)$	$O(1)$	$O(n)$
unsorted linked list	$O(1)$	$O(n)$	$O(n)$	$O(n+1) \approx O(n)$
Heap	$O(\log n)$		$O(1)$	$O(\log n)$

Heap is a binary tree ( $O(1)$ ) 3-ary ... ( $O(r)$ )  $r$ -ary tree

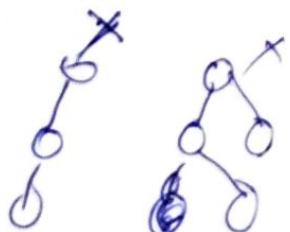
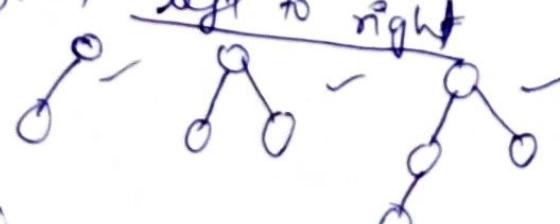
complete binary tree -



Heap (Almost Complete B-T).

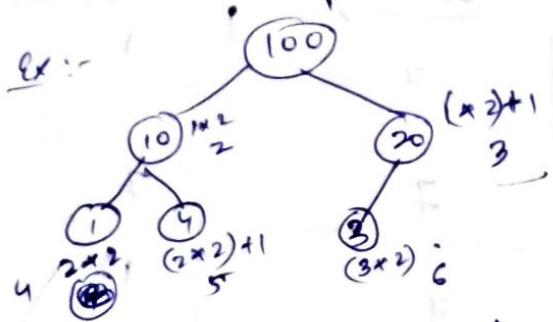
leaves should present only in last level

from last left to right but one level & filled

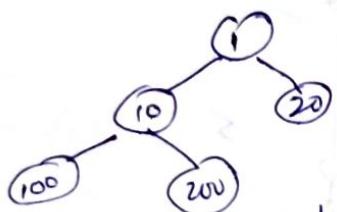


Max Heap:

All the elements in left & <sup>right</sup> <sub>subtree</sub> <sup>of node</sup> should be less than root for every node

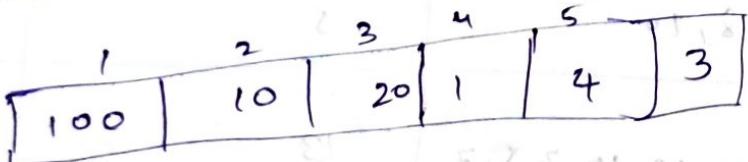


Min heap: ~~parent parent~~ its  
~~root root~~ Should be greater than ~~its~~  
~~right & left subtree~~ elements ~~children~~  
~~"root" contains min. element~~  
~~of entire tree.~~



All the elements in left & right  
 subtree should greater than root  
 for every node

### Implementation



left shift  
 - left child -  $2 \times (\text{parent index})^0 = 2 \times 0 = 2 \times 1$   
 - Right child -  $2 \times (\text{parent index}) + 1 = 2 \times 1 + 1 = 2 \times 2 + 1$   
~~left shift + 1~~  
 parent =  $\frac{(\text{child index})}{2}$  = Right shift

Example :-

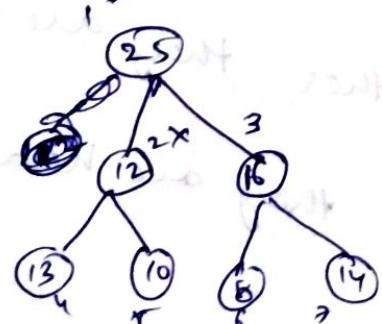
→ check below examples, whether they are heaps,

if not upto what length they are heaps

	<u>A. length</u>	<u>A. heap size</u>
① 25, 12, 16, 13, 10, 8, 14	7	1
② 25, 14, 16, 13, 10, 8, 12	7	7
3. 25, 14, 13, 16, 10, 8, 12	7	1
4. 25, 14, 12, 13, 10, 8, 16	7	2
5. <u>(14, 13, 12, 10, 8)</u> → descending order → max heap	5	5
6. <u>14, 12, 13, 8, 10</u>	5	5
7. <u>14, 13, 8, 12, 10</u> → max heap	5	5
8. <u>14, 13, 12, 8, 10</u>	5	5
9. 89, 19, 40, 17, 12, 10, 2, 5, 7,	13	2
10. 11, 6, 9, 70	5	5

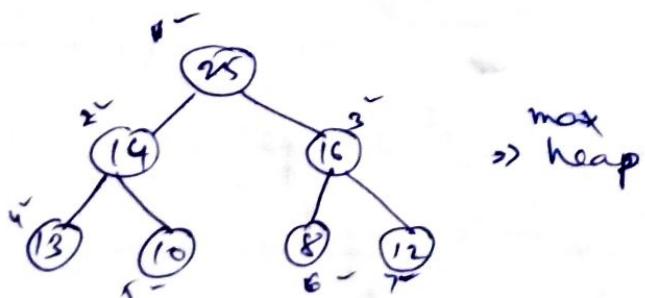
### A. heap size

The no. of elements following heap property

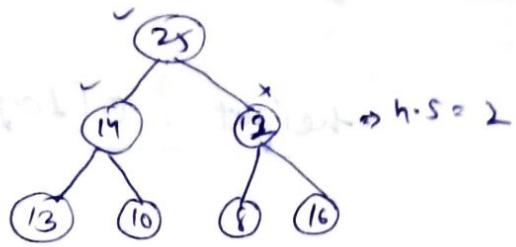
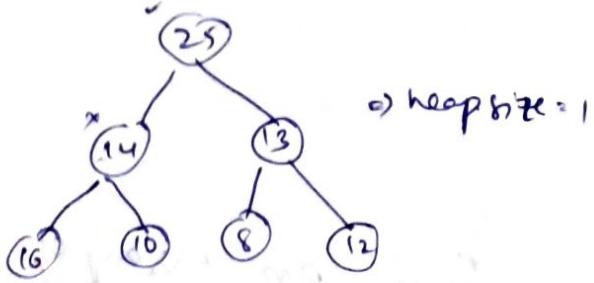


⇒ heap size = 1 [∴ only 1 element follows heap property i.e., parent ~~>~~ children]

②

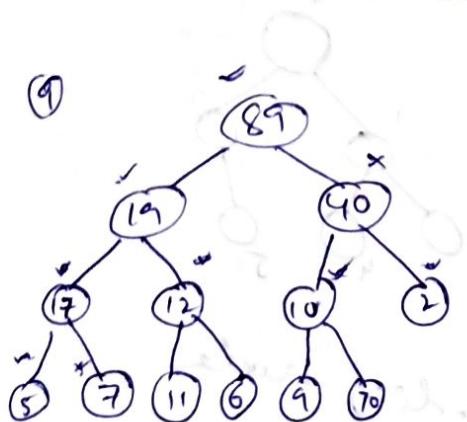


⇒ max heap

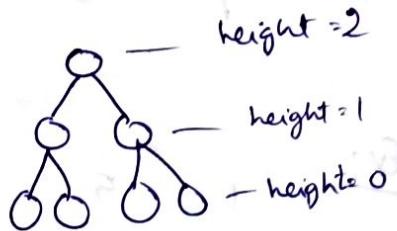
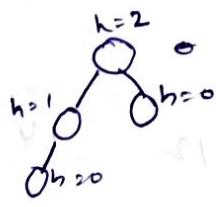


→ Array → Ascending order  $\Rightarrow$  ~~min~~ heap

$\Rightarrow$  Descending order  $\Rightarrow$  max heap



## Max Heapify Algorithm



Height of C.B.T	1	2	3	4
max no. of nodes	3	7	15	31

$$\Rightarrow \text{max. no. of nodes} = (2^{h+1} - 1)$$

## complete ternary tree

$$\text{max no. of nodes} = 3^{h+1} - 1$$

$$\left. \begin{array}{l} \text{complete n-ary tree} \\ \text{max. nodes} = n^{h+1} - 1 \end{array} \right\}$$

Relation b/w weight & no. of nodes

$$h = \lfloor \log n \rfloor$$

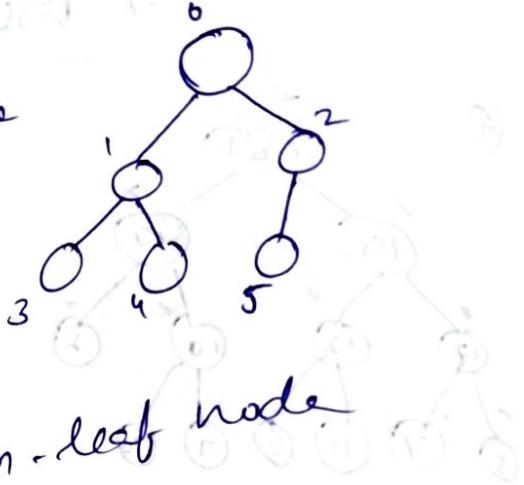
height  $\Theta(\log n)$

Construction of Max-heap from an array of elements

\*\* In a complete binary tree, the

elements starting from

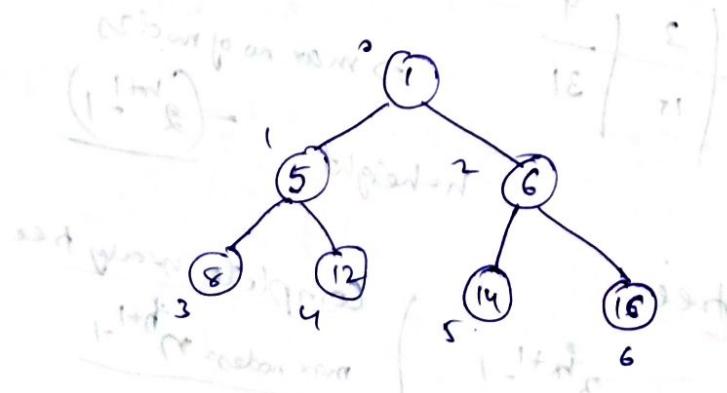
$(\lceil \frac{n}{2} \rceil + 1, \dots)$  are leaves  
is the largest non-leaf node



\* Every leaf is a heap.

Ex:-

Step 1: write in binary tree

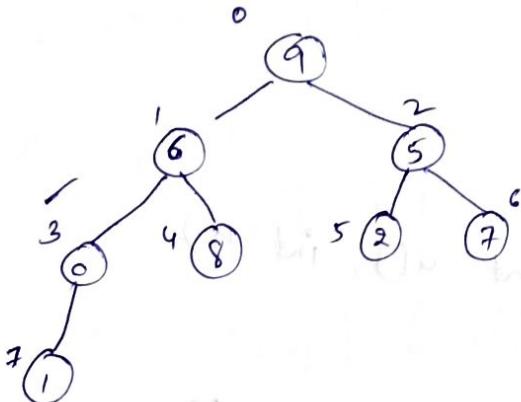


Here '2' is the largest non-leaf node

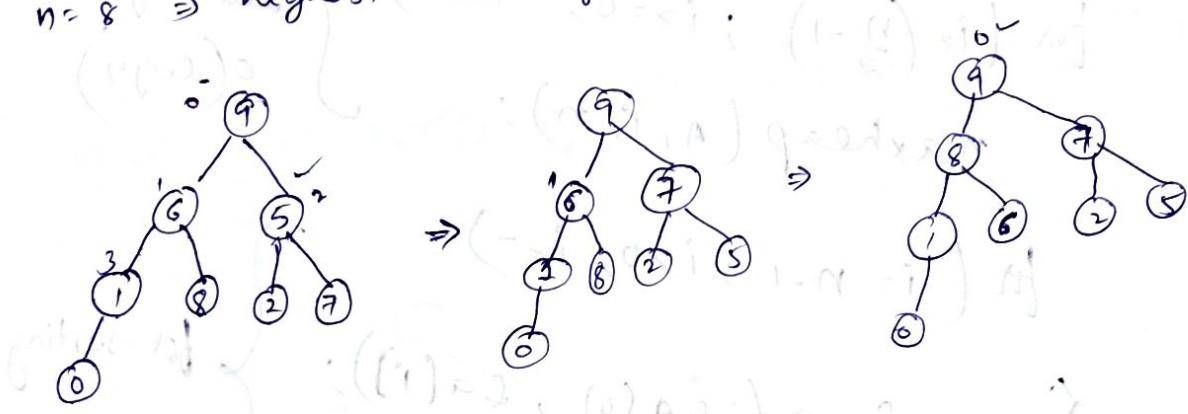
→ start from the largest non-leaf node

and start changing tree to form max-heap by moving towards up in tree

② 9 6 5 0 8 2 7 1



$n = 8 \Rightarrow$  highest non-leaf node  $\Rightarrow \frac{n-1}{2} = \frac{8-1}{2} = 3$



Refer Nagamani Notes (Pictures)

void maxheap (int A[], int i, int n)

1      int child;  
      int temp;

for (temp = a[i]; leftchild(i) < n; i=child)

2      child = leftchild(i); // first compare leftchild, if  
          if (a[i] < a[child]) then move  
          to rightchild

if (child != n-1 && a[child+1] > a[child])

    child++

if (temp < a[child]) // compare right child + root

    a[i] = a[child];

else break;

}

a[i], temp;

}

void heapSort ( int a[], int n)

{ int i;

for (i = (n - 1) / 2; i >= 0; i--) } building max heap

maxheap(a, i, n);

for (i = n - 1; i >= 0; i--) } o(log n)

{ Swap (&a[0], &a[i]); } for sorting

maxheap(a, 0, i); } o(n)

}

$\Rightarrow$  Total for sorting  $O(n \log n)$

After sorting, just print array, to see the output

## Algorithm for heapifying

Build-max-heap ( $A$ )

i)  $A.\text{heapSize} = A.\text{length}$

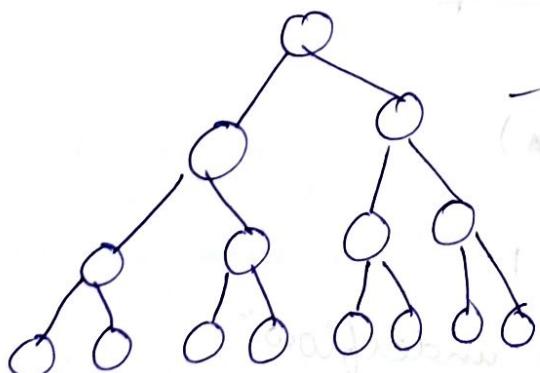
for ( $i = \lceil \frac{A.\text{length}}{2} \rceil$  down to 1)

max-heapify ( $A, i$ )

3

→ Number of nodes of different heights in a complete binary tree

binary tree



→ if no. of nodes =  $n$ ; height =  $h$

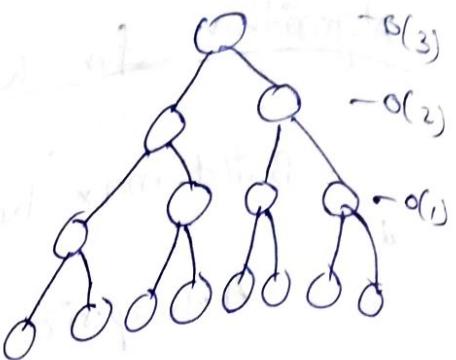
Then, no. of nodes at height,  $h$  (atmost)  $= \left\lceil \frac{n}{2^{h+1}} \right\rceil$

Ex: height = 0,  $\left\lceil \frac{15}{2^{0+1}} \right\rceil = 8$  nodes (atmost)

$n=2$ ;  $\frac{15}{2^{2+1}} = \frac{15}{2^3} = \frac{15}{8} = 2$  nodes

height = 2 ⇒ 2 nodes

$$\sum_{n=0}^{\infty} \left[ \frac{n}{2^{n+2}} \right] (c \cdot h)$$



$$= \frac{cn}{2} \sum_{h=0}^{\log n} \left( \frac{h}{2^h} \right)$$

-  $\frac{cn}{2} \cdot 2$

$\tilde{O}(n)$  will fit for above for result  
not proved

Extracting -max -in heap

Heap-extract-max (A)

2 if A.heapSize < 1  
error "heap underflow"

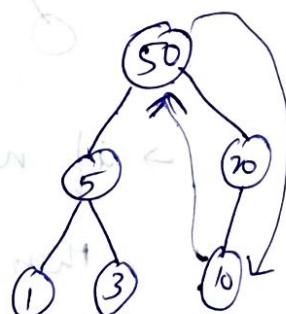
max = a[1]

$a[i] = a[a.\text{heap\_size}]$

a. heapsize  $\Rightarrow$  a. heapsize - 1;

~~(from) max heapify ( $a, i$ )~~

return max;



and reduce heap size and ~~needed~~ apply max heapify on  $\approx 10^{\text{th}}$  and then if it forms heap & then again if you want max element you can delete

## Training

### Applications of linked list:-

1. Media player

1. volume - doubly linked list

2. forward button -

3. Reverse button -

4. play, pause -

### Stacks

1. browsing history (queue)

2. Inbox

### Queue:-

- customer care

- Online Reservation

## Dynamic Programming

→ Intermediate soln's are stored and used to find the solution.

### Generic form of printf

printf ("");

printf (

variable no. of arguments  
(list)

printf (const char \*, ...)

↳ dynamically allocated string

↳ syntax for making a ~~function~~  
built in function

Q. Pgm → compiler → Assembler → loader → Run

C

b) ex: construction of house

- 1) foundation
- 2) pillar
- 3) roof    4) walls

d) Top-down

c++  
1. Integration of parts

2. Bottom up

class

User defined data type

object - instance of class  
- object has life (memory allocated)

Encapsulation:

class ~~calc~~ calc

2. - int a;  
    int b;

public:

int add(int a, int b) {

    int c; int d;

c = ~~this.~~ this.a + this.b;

return c;

int sub (int a, int b) {

    int c;

~~c = this.a - this.b;~~

return c;

```
int main() {
    calc a calc (a,b);
    cout << "Enter operation";
    cin >> op;
    cout << "Enter a+b"; cin >> a;
    cout << "Enter a-b"; cin >> b;
```

switch (op) {

case 1: add(a,b);  
int c = add(a,b);  
cout << "a+b" << c;  
break;

case 2: sub(a,b);  
int c = sub(a,b);

cout << "a-b" << c;

break;

default:  
cout << "Enter correct operation";

```
return 0;
} catch (Exception e)
```

}

(Exception e)

try {

```
if (b == 0)
    throw b;
else
    throw
```

testing along

→ try, catch block also used for  
with exception handling

14/7/15

# → highest priority

< → gives specific file → (search for file made directly where 'c' is installed)

> → gives just the paths to a file

" " → gives just the paths to a file → (search first made in current folder and then where 'c' is installed)

hello.cs → c → two

1. Preprocessing

~~compilation~~ → hello.i

2. Compiling

→ hello.s (assembly code)

3. Assembling

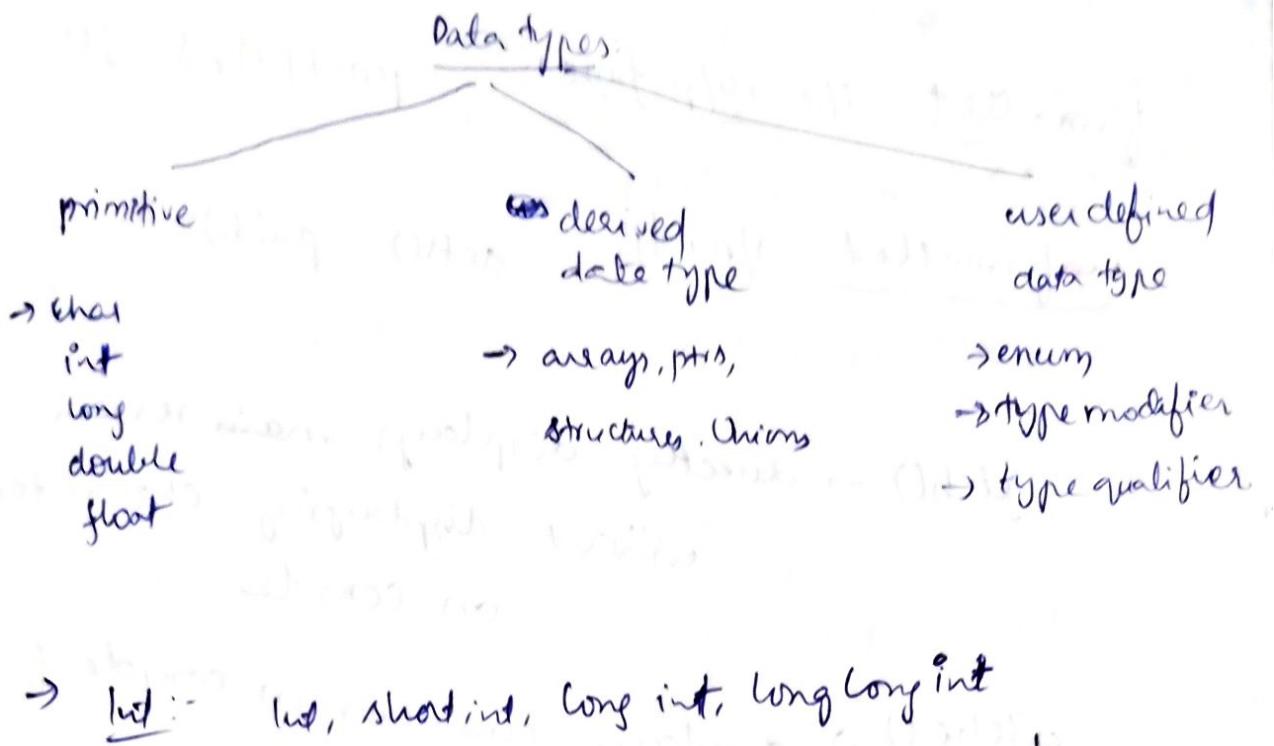
→ hello.o (object file)

4. Linking

→ hello.exe (links & gives .exe file)

Loading

→ hello



<u>Raw</u>	<u>Date type</u>	<u>Size</u>	<u>Range</u>
	int	2	$-2^{16}$ to $2^{16}-1$
	char	1	$-2^8$ to $2^8-1$

- Type definition
- implicit casting
  - explicit casting

typedef int avenum;

int no; avenum no;

enum week { sun, Mon, Tue, Wed, Thu, Fri, Sat };

↳ we cannot change enum values

void main() {

enum week today = Fri;

today = Tue;

printf("%d", today+1);

{ i=10, j=20, k=50 }

'k' cannot be changed

formatted i/p o/p func : printf(), scanf()

unformatted i/p o/p : gets(), puts()

getch() → directly displays main screen without displaying character on console

getche() → displays character on console + direct to main screen

a = - - 2; (only for increment)  
a = 20 (decrement)

$$\begin{aligned} a &= (a++) + (++a) \\ \underline{a = 20} &\quad \begin{matrix} 2 \\ 2 \end{matrix} \\ \underline{a = 43} &\quad \underline{43} \end{aligned}$$

main() {

int i = 10, j = 2, k = 0, m;

m = ++i || ++j + ++k;

printf("%d.%d.%d.%d", i, j, k, m);

3.

Ans:

11, 2, 0, 1

Sol: when handling with '11' ~~(or)~~ 'ft'  
 if we set first '1' in '11', then it will not compute  
 remaining statement. or if we false '4' - if <sup>0</sup>.

Q  
1011

Printf ("%d %d", ++i, ++j);

Q

int a = (1, 2, 3, 4, 5)

int n = 1, 2, 3, 4, 5

printf ("%d, %d", a, n)

Q  
sp. program  
int a = 1, 2, 3, 5;  
~~78112~~  
error

a = 5

\* = 1

Right shift operations of negative no. <sup>110</sup>  
<sup>(or) 10000</sup>

~~10~~ (hexadecimal)  
~~0000 0000 0000 1010~~

$\cong^{10}$   
~~1111 1111 1111 1010~~

after right shift vacant bits will be  
 filled by '1's.

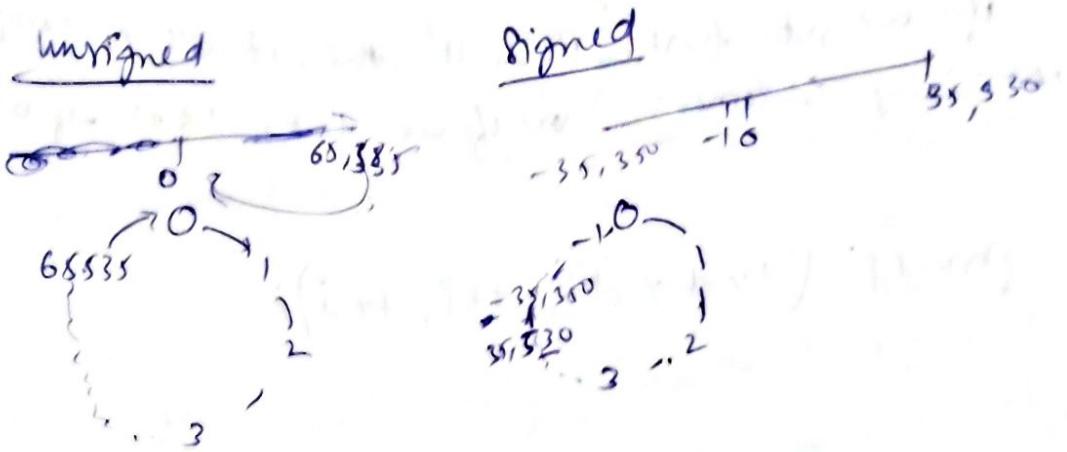
~~101000~~  
~~111111~~  
~~100000~~

$10 \frac{10}{2-8}$

~~00000~~

110 

① 1.  $\rightarrow$



~~unsigned~~  $a = -1;$

$$\Rightarrow a = \underline{65,535}$$

Q. of (1, 2, 3, 6, 0)

print ("hai");

else

print ("hello");

~~a = 97~~ 2 - 32  
A - 65

printf ("%f.%f.%f.%f.%f");

10 - carriage return

c/r : %f.%f

1b - backspace

Q. Bitwise Operator

7a - alarm

Q. print ("m ks"); ks

print ("1b mi la"); ksmi off:-

print ("1r ha ly"); Rhi hai

~~printf ("1.d", printf("FACE"));~~ → O/P FACE 4  
O/P: FACE → outer & inner printf:  
 ↗ + ↘  
 ↗ ↘  
 ↗ ↘

~~printf ("1d", FACE)~~  
~~printf ("FACE" + 2);~~ → only upto '7'  
ignore 1st two characters  
 ↗ ↘

2.155  
./2.3f  
 2.100

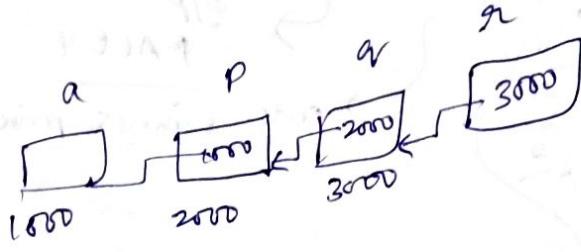
"FACE"  
ptr + 2  
 ↓  
FACE  
PA  
(ptr + 2)  
 ↗ (ptr + 2)

④ Arrays

int num[10];  
 int num[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

Pointers

int \*p;



$$P = 1000$$

$$\&P = 10$$

$$q = 2000$$

$$\&q = 1000$$

$$\&\&q = 10$$

$$r = 3000$$

$$\&r = 2000$$

$$\&\&r = 1000$$

$$\&\&\&r = 10$$

113

(x a) + 1

a + 1

int \*arr[10] → ptr to array

int (\*arr)[10] → array of pointers

array of pointers

(int (\*) func)()



int n = 20;

int a[n] = {1, 2, 3, 4} X error

only we have to use

macros

#define n 20;

Q)

a

a(2), 2(a)

To do

Ques 1)  $a[0] = 2^3$  now what would you call it?

$$2^{(0+1)+1+1} = 2^4$$

Ques 2)  $\{int a[3][2]\} = \{1, 2, 3, 4, 5, 6\}$

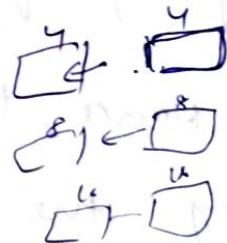
printf("%d, %d, %d", a[2][1], x^(a[2]+1), x^(x^(a+2)+1))

printfs of banners took around

Ques 3)  $\{int arr2D[3][3]\} = \{1, 2, 3, 4, 5, 6\}$

printf("%d, %d, %d, %d, %d, %d", arr2D, arr2D, arr2D)

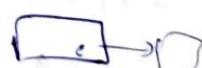
Ques 4)  $\{int arr2D[3][3]\} = \{1, 2, 3, 4, 5, 6\}$



11. 2

2 + 3 + 2

2 + 3 + 2



Ans)  $\{int arr2D[3][3]\} = \{1, 2, 3, 4, 5, 6\}$

## Bleak Number

The following numbers that are bleak, comes in a series of numbers

1, 4, 6, 13, 15, 18, 21, 23, 30, 32, 37, 39, 45.

This particular series are called Binary Self

## Colombian Series Numbers

Numbers that cannot be expressed as

the sum of distinct terms of the form  $(2^k + 1)$  ( $k \geq 0$ )  
 (or) equivalently, numbers not of form  $(m + \text{sum of binary digits of } m)$

Hence, this problem can be solved either by

→ checking the  $2^k + 1$  or m + sum of binary digits of m

$$\underline{2^k + 1}$$

$$2^0 + 1 = 2$$

$$2^1 + 1 = 3$$

$$2^2 + 1 = 5$$

$$2^3 + 1 = 9$$

4 [cannot be formed]  
 6 [cannot be formed] } bleak

static bool IsBlock(int m)

2  
int len = 0;

while ( $(2^{\text{len}} + \text{len}) < m$ )

len++;

for (int i = len; i >= 0; i--)

if ( $m \geq (2^i + 1)$ )

$m = m - (2^i + 1)$

return  $m != 0$ ;

}

## Dynamic Programming

→ Using the solution which is already solved  
by not solving it again

Ex:- fibonacci series

$$f(n) = f(n-1) + f(n-2) ; \text{ otherwise}$$

$$= 1 ; n = 1$$

$$= 0 ; n = 0$$

2  $f(n)$

if ( $n == 0$ )

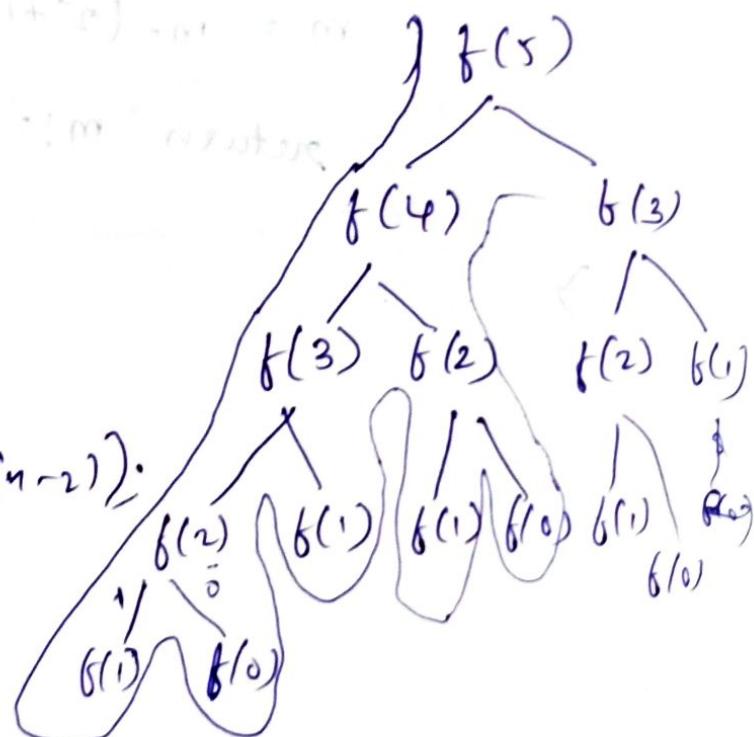
return 0;

if ( $n == 1$ )

return 1;

return ( $f(n-1) + f(n-2)$ );

3



check the table

for  $T(n)$

if  $T(n)$  is empty

$f(n)$

if  $T(n)$  is empty

$T(n) = f(n)$

2	3	5	8
2	3	4	5

## Introduction to LCS

(longest common subsequence)

substring ≠ sub sequence

Ex: Substring      RAVINDRA  
                        1 2 3 4 5 6 7 8      Subsequence

1) should be contiguous like raw, vin, dra, ra, av...	2) <del>can</del> must not be contiguous but indices should be in ascending order
---	---

like Rav, Vin, dra,  
ta, av..

2) ~~to~~ must not be contiguous  
but indices should be in  
ascending order

Ex:  $\{1, 3, 5\}$ ,  $\{1, 4, 6\}$

21, 8.53 X  
RAN

Q) Given a string, then no of substrings =  $2^m$

## DNA Example:-

→ longest common subsequence, applied in DNA components  
finding similarities b/w DNA's  
(A, G, C, T)

Ex: if DI = AGCCCTCAGT

$$P_2 = GCT$$

If  $D_1 = D_2$  then they are of same person  
 if  $(D_1 \neq D_2)$  then they are of very much similar person

$D_2$  is substring of  $D_1$  they are like  
much similar

if  $D_2$  not a substring of  $D_1$  then we try to  
find longest common subsequence (LCS)

to measure similarity.

Ex:  $S_1$ : Ravindra

$S_2$ : Ajay

Subsequences = { $d_3, A, AA$ }

longest common subsequence =  $\{AA\}$   
of length = 2

$A = RAVINDRA$

$B = AJAY$

Brute force Approach to find subsequences

time complexity:

$O(2^m)$  - Find all subsequences of 'A'

$O(n \cdot 2^m)$  - Find for each subsequence whether it is a subsequence of 'B'

$O(2^m)$  - 3. Find the LCS

## Optimal Substructure:

If the main problem is split into small problem, which can be part of main problem

$$X = [x_1, x_2, x_3, \dots, x_n] \quad x_1, x_2, \dots, x_n$$

$$Y = [y_1, y_2, y_3, \dots, y_m] \quad y_1, y_2, \dots, y_m$$

$$x_1, x_2, x_3, \dots, x_i$$

$$y_1, y_2, y_3, y_4, \dots, y_j$$

$$c[i, j] = \begin{cases} 0 & ; i=0 \text{ or } j=0 \text{ (if any of the string length is 0)} \\ 1 + c[i-1, j-1] & ; i, j > 0 \text{ and } x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & ; i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

## Recursion Tree & Unique

### Sub Problems

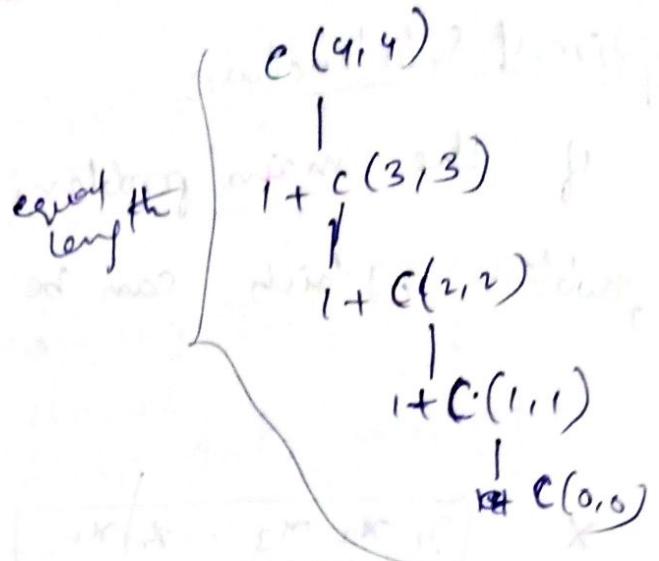
case i) if input DNAs of same lengths: (Best Case)  
& match found

$$X = \{A, A, A, A\}$$

$$Y = \{A, A, A, A\}$$

Time compl:  
 $O(n)$

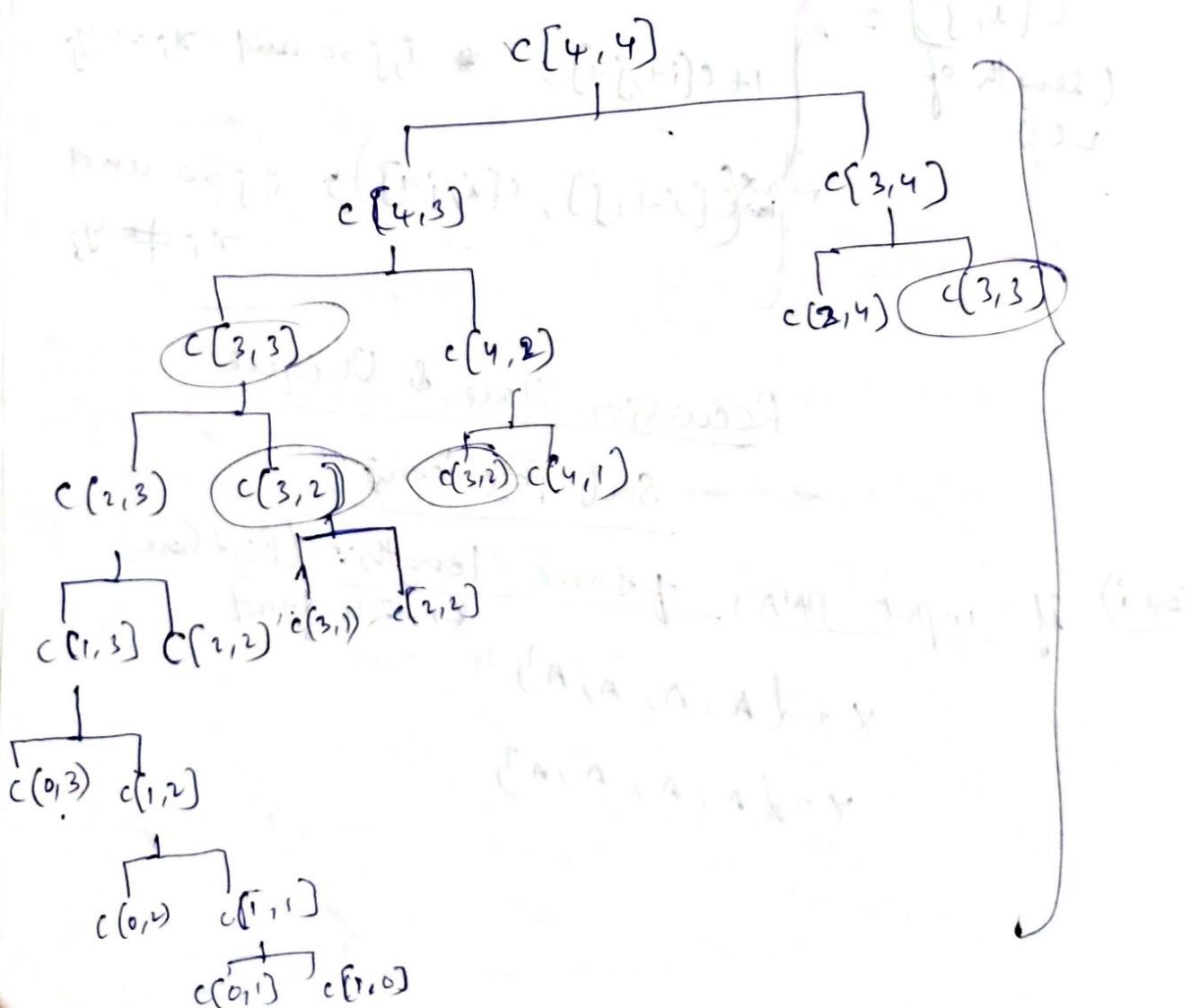
Space comp:  
 $O(n)$



Case ii:- DNA's of diff length (n). Same No match found:-

$$x = \{ \overset{1}{A}, \overset{2}{A}, \overset{3}{A}, \overset{4}{A} \}$$

$$y = \{ \overset{1}{B}, \overset{2}{B}, \overset{3}{B}, \overset{4}{B} \}$$



$c(n, m)$

1

$c[n-1, m]$

1

$c[n-1, m-1]$

1

$c[n-2, m-1]$

1

$c[n-2, m-2]$

1

$c[0, 1]$

$c[1, 0]$

depth of tree

$O(n+m)$

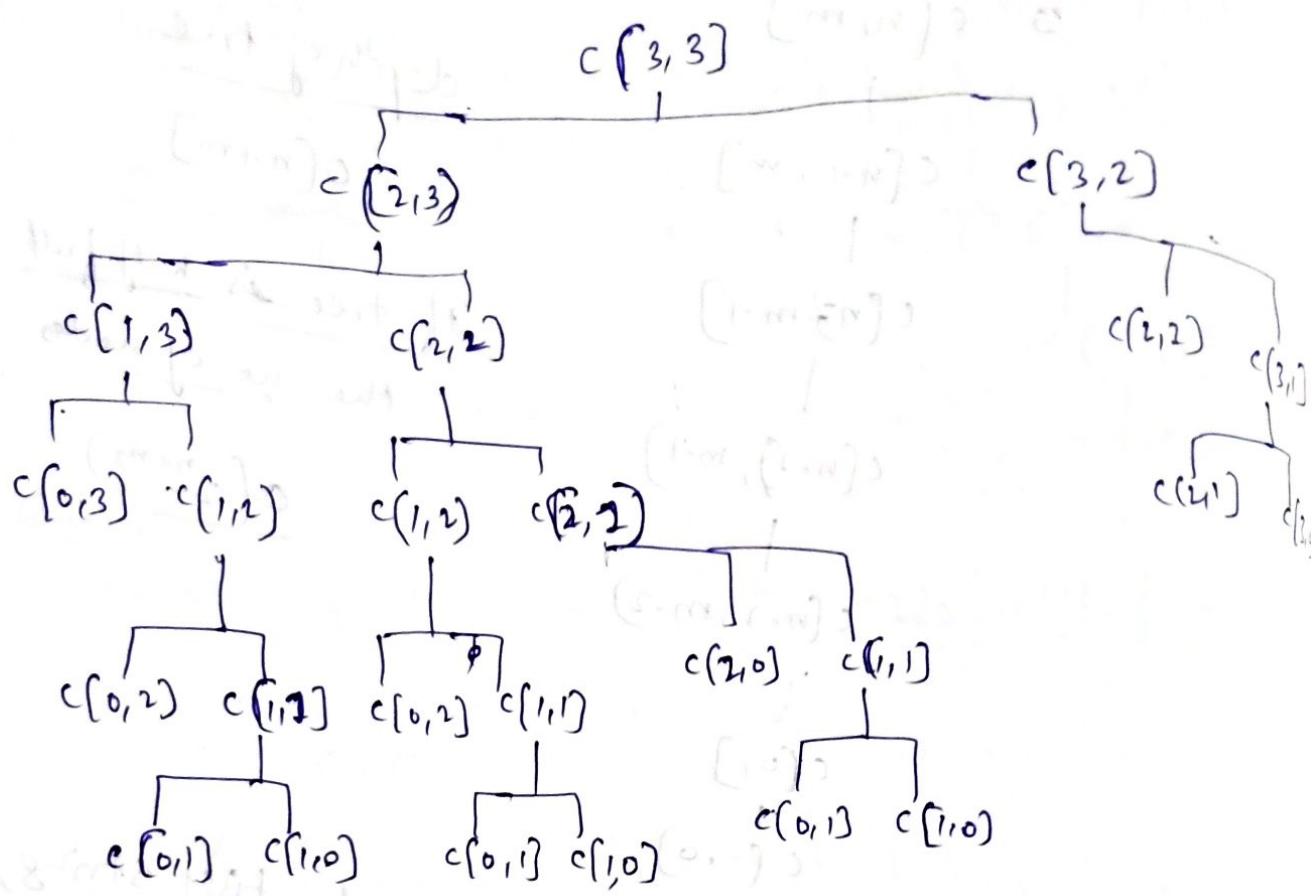
If tree is half full  
the no. of nodes

$O(2^{n+m})$

\* Tree no. of \* if  $n = m$  length of first string,  
 n length of second string, then no. of unique  
 solutions =  $(n+m)$ . [if we discard  $(0,1)(0,0), (1,0)$   
 $(0,0)$  else  $(n+1) * (m+1)$ ]

ex.: (3, 3) - unique solutions  $\Rightarrow \frac{3 \times 3}{(0,1)} = 9$

$4 \times 4 = 16$



unique nodes

(3,3), (3,2), (3,1)

(2,3), (2,2), (2,1)

(1,3), (1,2), (1,1)

(1,0), (2,0), (3,0)

(0,1), (0,2), (0,3)

(0,0)

Cells required

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

Bottom-Up Approach *X this is best*  
 compute smaller problems first & then  
 larger problems

Top-Down Approach:  
 compute larger problem first & then  
 smaller

→ Top-Down Memoization

→ same as D.P

Examples:

$$x = (A A B)$$

$$y = (A C A)$$

		A	C	A
		0	1	2
		0	0	0
x	0	0	1	1
A	1	0	1	2
A	2	0	1	1
B	3	0	1	2

if  $x_i = y_j - 1 + c(i-1, j-1)$   
 else  $\max(c(i-1, j), c(i, j-1))$

Ex 2:

$$X = \{A_1^1, B_1^2, C_1^3, B_2^4, D_1^5, A_2^6, B_2^7\}$$

$$Y = \{B_1, P_1, C_1, A_1, B_2, A_2\}$$

	X	A	B	C	B	D	A	B
A	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Q B C B A

② B D A B

$$\textcircled{3} \quad B \subset A \subset B$$

# Matrix Chain Multiplication

## Dynamic Programming

A  $(3 \times 2)$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

$3 \times 2$   
 $p \times q$

B  $(2 \times 3)$

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

$2 \times 3$   
 $q \times r$

If The no. of multiplications required

$$= \boxed{(p \times q) \times r}$$

$$\Rightarrow (3 \times 2) \times 3$$

no. of elements  $\rightarrow$  col index

## Chain Multiplication

①

$$\left( \left( \underbrace{\begin{matrix} A \\ 2 \times 1 \end{matrix}}_{\substack{\text{multiplications} \\ (1 \times 2 \times 1)}} , \underbrace{\begin{matrix} B \\ 1 \times 2 \end{matrix}}_{\substack{\text{multiplications} \\ (2 \times 2 \times 1)}} \right) \underbrace{\begin{matrix} C \\ 2 \times 4 \end{matrix}}_{\substack{\text{multiplications} \\ (2 \times 4 \times 2)}} \right)$$

$\Rightarrow$   
 $(4 \times 1) + 1 = 16$   
go multiplication

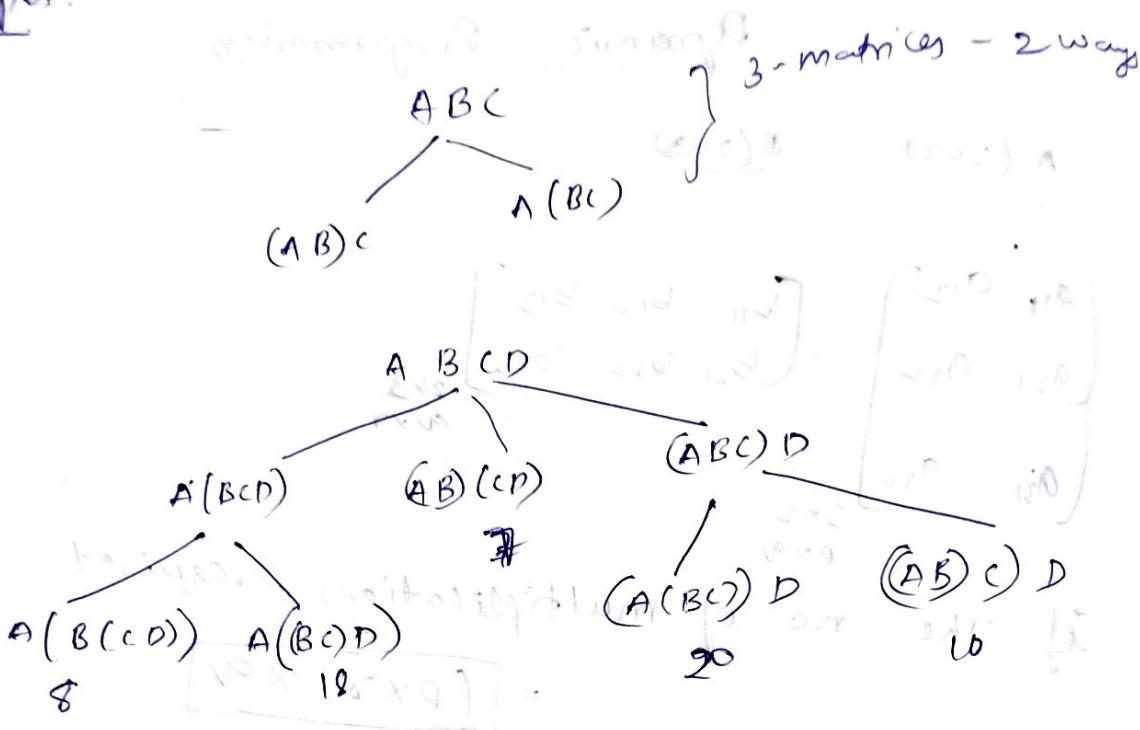
②

$$\left( \underbrace{\begin{matrix} A \\ 2 \times 1 \end{matrix}}_{\substack{\text{multiplications} \\ (2 \times 4 \times 1)}} \underbrace{\left( \begin{matrix} B \\ 1 \times 2 \end{matrix} = \begin{matrix} C \\ 2 \times 4 \end{matrix} \right)}_{\substack{\text{multiplications} \\ (1 \times 4 \times 2)}} \right)$$

$\downarrow$   
 $(2 \times 4 \times 1) + (1 \times 4 \times 2) = 16$   
16 multiplications

In ② approach, no. of multiplication are reduced.  
so, if we take substantially large dimensions,  
it will be more effect in operations.

Examples:



Ex-.

$$A_{1 \times 2} \quad B_{2 \times 1} \quad C_{1 \times 4} \quad D_{4 \times 1}$$

$$A(B(C)D)$$

$$\begin{array}{rcl} 2x+1=4x \\ 1 \times 2 - 11 & 2x+4=4x+1 \\ 2 \times 4x & & \Downarrow \\ 8 & + & 8 \\ 1 \times 1 \times 2 & & \\ 2 & + & \end{array}$$

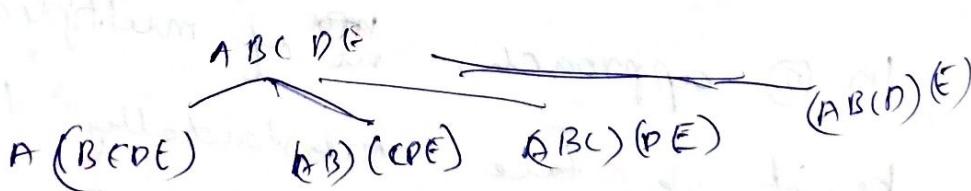
18

$$A \begin{pmatrix} B(CD) \\ \frac{1}{2 \times 1} (1 \times 1) \\ \frac{1}{2 \times 1} (2 \times 1) \end{pmatrix}$$

$$\begin{array}{c} \text{Diagram showing } A(B+C) = AB + AC \\ \text{Left side: } A \downarrow \text{ (with a checkmark)} \quad B+C \\ \text{Right side: } AB + AC \end{array}$$

4 matrices  $\rightarrow$  5 ways

5 matrices  $\rightarrow$  14 ways



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

if 4 matrices  
 $n = 3$ ,

## Optimal Substructure & Recursive Equations

Aim:- To parenthesise in such a way that minimum no of multiplications are required

matrices :-  $A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ \dots \ A_n$   
 dimensions,  $P_0 \times P_1 \ P_1 \times P_2 \ P_2 \times P_3 \ P_3 \times P_4 \ P_4 \times P_5 \ \dots$   
 of matrices

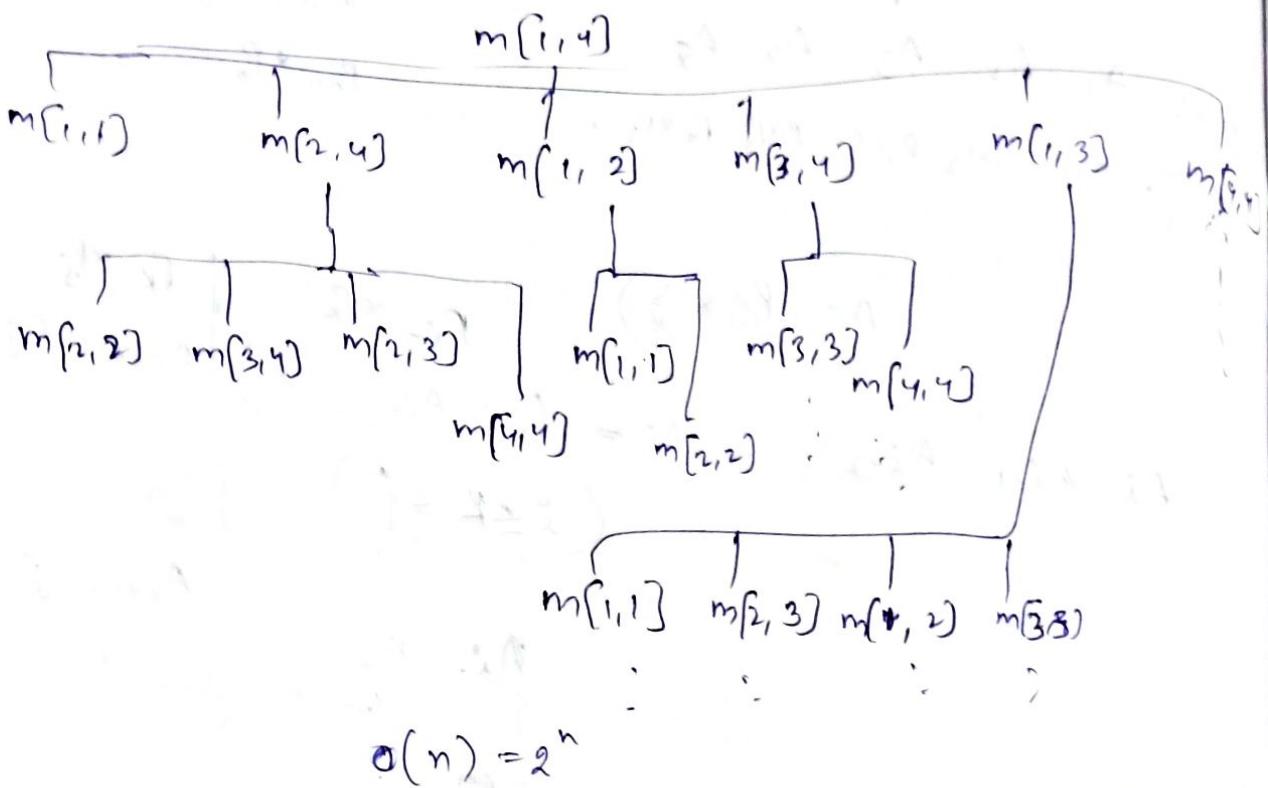
$$A_i \circ (P_{i-1} \times P_k) \quad P_{i-1} \times P_k \quad | \quad P_k \times P_j \\ A_i \circ A_{i+1} \circ A_{i+2} \circ \dots \circ A_j = (A_i \circ A_{i+1} \circ \dots \circ A_k) / (A_{k+1} \circ \dots \circ A_j) \\ (i \leq k \leq j)$$

$A_{i \dots k}$

## Recursive Equation

$$m[i, j] = \begin{cases} 0 & ; i = j \\ \min_{i \leq k < j} \left( m[i, k] + m[k+1, j] + (P_{i-1} \times P_k \times P_j) \right) & \end{cases}$$

## Recursive Tree



The problems are repeating, so no need to compute repeatedly

## D-P Bottom Up Implementation:

				subproblems	
(1,1)	(2,2)	(3,3)	(4,4)	- 4	1
(1,2)	(2,3)	(3,4)		- 3	2
(1,3)	(2,4)			- 2	3
(1,4)				- 1	4

$$(1,3) = \min \left\{ \begin{array}{c} (1,1) + (2,3) + P_0 P_1 P_3 = 16 \\ \quad 0 \quad 8 \quad 8 \\ (1,2) + (3,3) + P_0 P_2 P_3 = 14 \\ \quad 2 \quad 0 \quad 8 \quad 2 \end{array} \right.$$

$$O(n^2) O(n) = O(n^3)$$

te

zu

$$\begin{array}{c} n \\ (n-1) \\ (n-2) \\ \vdots \\ 1 \\ \hline n + (n-1) + (n-2) + \dots + 1 \end{array}$$

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

# Bottom-up Dynamic Programming Algorithm

LCS(x, y)

1.  $m = x.length;$

$n = y.length;$

Let  $c[0..m, 0..n]$  be a new table

for  $i=0$  to  $m$

$c[i, 0] = 0$

for  $j=0$  to  $n$

$c[0, j] = 0$

for  $i=1$  to  $m$

for  $j=1$  to  $n$

if  $x_i == y_j$

$c[i, j] = c[i-1, j-1] + 1$

else

$c[i, j] = \max(c[i-1, j], c[i, j-1])$

}

}

return c;

}

Time Comp:  $O(m \times n)$

Space Comp:  $O(n^2)$