1.	Week1 Day1 11/8	Introduction to Programming Concept Data Types and Bit Operations:	1 - 12
Week1.	Day2 12/8	Bit operations, and Complexity analysis.	13-26
2.	Day1 18/8	Recursion	27-40
Week2	Day2 19/8	Sorting and searching	41-55
3	Day1	Searching	56-63
Week3	Day2	Lab	—
4	Day1	Searching and Hashing	64-75
Week4	Day2	Hashing	76-87
5	Day1	Strings	
Week5	Day2	Mixed Bag Packing	
week6	Day1	Prime numbers & Game theory.	
	Day2	Lab	

Steps of problem solving

step①:- Understand the problem constraints inputs and outputs

step②:- come up with a solution.

step③:- validate the correctness of solution (on edge cases)

step④:- Time and space complexities

step⑤:- $T \times M \leq 10^8$, $S \leq 10^{6-7}$

step⑥:- Figure out the different parts of code

① data-types

② code-outline (i/p & o/p for function)

③ critical conditions ($j = \emptyset 1$, i or $i+1 < n, n+1, n-i$)

step⑦:- Then write the code in one shot.

Week1

11/8/2018

→ While writing a program we declare variables.

→ Variables store the data in memory.

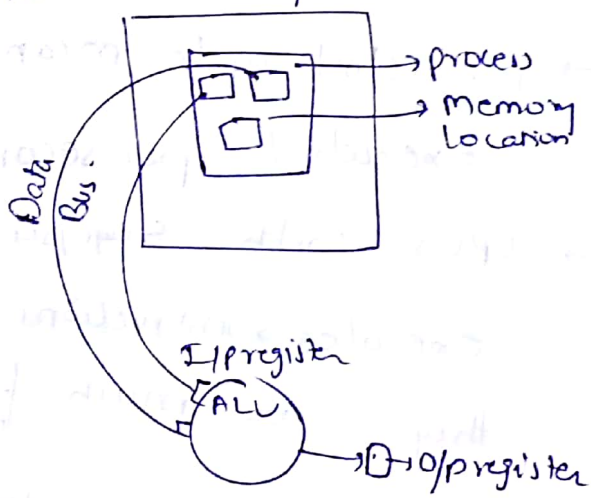
→ When we perform an arithmetic operation then the data is sent through data bus to ALU

↳ ALU consists of input registers to store the i/p data temporarily.

→ The address of the variable are received by the address bus to ALU.

→ The operations are performed in ALU and then the result is stored in o/p register.

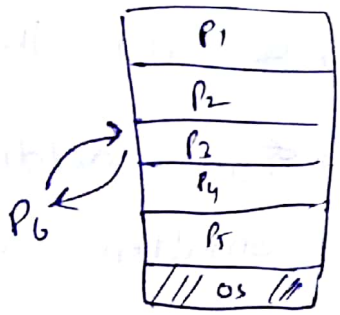
→ Then it is again fetched to the memory and displayed.



→ Now consider the processes P_1, P_2, P_3, P_4, P_5. They are executing in the system.

→ If another process P_6 is to be added then a process has to be swapped out

→ Here we are swapping out P_3.



→ Swapping out a large no. of processes will make the system to halt this is known as thrashing.

→ Performance of the system depends on:

① Ramsize
② Speed (Technology DDR3, DDR4)
③ Disk (SSD vs HDD)
   ↓ costly, speed
   ↓ low cost, low speed
④ Processor:
   1GHz
   1×10^9 GHz.
   10^9 cc/sec.
   10^9 instructions/sec.

→ 1 (cc) clock cycle means 1 instruction will be executed per second.

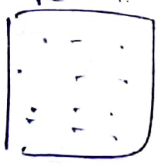
→ CPUs with 5 cyc/sec means 5 ins/sec can executes 5 instructions per second which means they are much faster.

⑤ Architecture - 32bit or 64bit.

→ to fetch the data address is required

→ Each address is unique.

→ Smallest amount of data which can be fetched is 1byte.



→ Which one of the systems would be faster the one having a RAM of 4GB or 8GB in a 32-bit system? A 4GB B 8GB.

→ Both of them would have the same speed.

1KB = 1024 B = 2^{10}B

1MB = ~~120~~ 1024 KB = 2^{10}B

1GB = 1024 MB = 2^{30}B

4GB = $2^2 \times 2^{30}$ B

= 2^{32} Bytes. → for 2^{32} bytes 32 different lines are required.

8GB means $2 \times 2^{32} = 2^{33}$ → There 32 lines are the size of RAM.

→ So even if the RAM is of size 2^{33} bytes then in a 32-bit machine it would only accept 32 bytes so, both will have the same speed.

MMU:- MMU is Memory Management Unit responsible for generating addresses.

→ MMU is controlled by the Operating System.

→ If OS is 32 bit then it will generate addresses of 32bits.

→ So, even if we use 8GB RAM or 4GB RAM for a 32bit machine it will take only 32bits. so, as shown in the diagram

Remaining 4GB will not be used.

→ An OS of certain bit can be run on H/W of certain bits

OS	H/W	
32 Bit	→ 32bit	✓
32 bit	→ 64 bit	✗
64bit	→ 32bit	✓
64 bit	→ 64bit	✓

→ Binary representation of 13.

13 → 00001101

-13 → 10001101 ⊗

So using 2's compliment we can represent -13 as

1111 0010
+1
-13 ⇒ 1111 0011 ✓

Data-Types:

Data-type	size
char	1B
int	4B
float	4B
double	8B
long	8B
bool	1B
short	2B

signed / unsigned are 2-bit.

→ find the lowest and highest values of signed and unsigned numbers with different bit size.

for 2bit → 0 ← 00 → 0 (signed)
1 ← 01 → 1
2 ← 10 → -2
3 ← 11 → -1

#bits	signed	unsigned
1	-1, 1	0, 1
2	-2, 1	0, 3
4	-8, 7	0, 15
8	-128, 127	0, 255
n	$-2^{n-1}, 2^{n-1}-1$	$0, 2^n - 1$

For 1bits → 0 = 0 (unsigned) = 0 (signed)
1 = 1 = -1

For 4 bits → 0000 → 0 (unsigned)
0111
1111 → 15
signed: 1000 → -8
-1 ← 1111
7 ← 0111

So, in a 32 bit machine
for char it takes 1Byte = 8bits, n = 8
signed $-2^{8-1}, 2^{8-1}-1$
signed $= -2^7, 2^7-1$
unsigned $= 0, 2^8-1$

data-type	Byte	bits	signed	unsigned
char	1 Byte	8bit	$-2^7, 2^7-1$	$0, 2^8-1$
int	4 Bytes	4x8=32	$-2^{32-1}, 2^{32-1}-1$	$0, 2^{32}-1$
float	4 Bytes	4x8=32	$-2^{32-1}, 2^{32-1}-1$	$0, 2^{32}-1$
double/long	8 Bytes	8x8=64	$-2^{64-1}, 2^{64-1}-1$	$0, 2^{64}-1$

$\Rightarrow \quad 2^{31} \rightarrow 2^{30} \times 2$

$2^{10} = 10^3$

$2^{30} = 10^9 \times 4$

→ So for int the maximum bits is 4×10^9 if we assign more than this to int then overflow will occur and we will not understand the actual problem.

→ We will get the constraints as
* sum of n elements in an array.

$1 \le N \le 10^7$
$-10^6 \le arr(i) \le 10^6$
$-10^{13} \le sum \le 10^{13}$

→ This indicates the number of elements in an array are from 1 to 10^7.

→ The possible elements can be in a range of -10^6 to 10^6.

→ So here an array of type int is okay for arr; but what if for sum.

→ If we declare sum as integer then it will overflow as int can hold only 10^9 and the sum has range in between -10^{13} to 10^{13}

→ Hence here we will get an error & we will not understand.

→ So, we must declare if(sum) as long.

→ Now again on huge data

```
int a = 10^6 , b = 10^7
int c = a * b;
print c;
```

$a * b = 10^6 * 10^7$

$c = 10^{13}$

↳ 'c' is of type integer so it cannot store 10^{12} hence there is a chance that overflow will occur we need to declare c as long and even type cast during multiplication.

```
int a = 10^6, b = 10^7
~~int c = a*b;~~
long c = long(a*b);
p|( c);
```

Operators :

→ Arithmetic operator :- +, −, *, ÷, %

→ Relational operator :- >, <, >=, <=, ==, !=

→ Logical operator :- &&, ||, !

→ Bitwise operator :- &, |, ~, ^, <<, >>.

a	b	a|b	a+b	a∧b	~a
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

int a=25; → 0011001
b=12; → 0001100

a|b → 0011101 (29)

a+b → 0001000 (8)

a∧b → 0010101 (21)

~a → 1100110 (this is not defined)
-128+64+4+2

→ Here in bit wise operations

a+b+c = c+b+a

a|b|c = c|b|a = b|c|a

a∧b∧a = a∧a∧b
= 0∧b
b ↔ b

∴ So the different one will be caught.

Performing left shift operations

a → 00000110 (6)

a<<1 → 00001100 (12)

a<<2 → 00011000 (24)

a<<3 → 00110000 (48)

a<<i → (a*2^i)

Performing right shift operation on a

a : 00010011 (19)

a>>1 : 00001001 (9)

a>>2 : 00000100 (4)

a>>3 : 00000010 (2)

a>>i : a*/2^i

→ Given a number 'n' as i/p parameter the function should return the result as 2^N.

// General way

```
pow2(int n)
{
  long int c=1;
  for(i=1;i<=N;i++)
    c=c*2;
  return c;
}
```

to reduce the ~~time~~ no. of steps using ~~right~~ shift operator.

```
pow2(int n)
{
  return 1<<n;
}
```

// Here the value which is returned should be type casted to long

so →

```
pow2(int n)
{
  return (long)1 <<n;
}
```

→ If I want to work with 2^63 then place 1 should be unsigned

so

1u <<n;

→ In general we have to declare it as unsigned long int.

→ 1u <<n is performed in a single step because as it is done

directly in the hardware.

/* For a given number n to check whether the i th bit is set or not */

Sample →

5, 2 → T	⇒ (1)01 → T
7, 1 → T	1(1)1 → T
7, 4 → F	(0)111 → F
10, 1 → T	1(0)10 → T
10, 5 → F	(0)1010 → F
10, 2 → F	

Code →

```
bool CheckBit(N,i){
    return ((n >> i)&1);
}
(or)
return  n>>i%2;
(or)
return  N&(1<<i)
```

/* For a given number n find the number of setbits */

It means no. of 1's in the binary.

Sample:

5 → 101 → 2

22 → 10110 → 3

15 → 1111 → 4

Code:

```
setBits(int n)
{
int c=0;
while(n!=0)
{
  if((n>>i)&1)
    c++;
}
S.O.P(c);
}
```

// At the positions 'x' and 'y' place '1' and remaining as 0's

Sample: x = 3, y = 2

3 2 1 0

1 1 0 0

Code:

```
int create(int x, int y)
{
  return (1<<x)|(1<<y);
}
(or)
  return 2^x+2^y;
```

// Starting from left place the no. of x's as 1's and no. of y's as 0's

Sample:

x y

3 2 → 1110 0

1 3 → 1000

4 2 → 111100

```
int create(int x, int y)
{
    return ((1<<x)-1)<<y;
}
```

(9)

(or)

```
return
for(i=y; i<=x+y-1; i++)
    {
      2^i;
    }
```

(4,2)

$2^4 = 10000 - 1$

$11111 << 2$

11100

n	n-1	n & (n-1)
25	24	24
18	17	16
24	23	16
26	25	24

n → 25 → 11001

n-1 → 24 → 11000

11000 = 24

```
int countSetBits(int N)
{
    while(N!=0)
    {
      n=n & (n-1)
       c++;
    }
}
```

/* If i want to extract bit at even or odd positions */

Sample: n = 25 → 00011001
00010001

```
int extr(int n)
{
  x = hexaDec(5);
return n & x ;
}
```

Week-1
Day-2

12/8/2018

Bitwise operations and complexity analysis:

/* Given an array of numbers from 1 to N+1 we need to find the missing number from 1 to N+1. */

Sample: [1, N+1] ← $ar_{N=6}$: 3 7 1 2 6 4 → 5

$ar_{N=5}$: 1, 5 4 2 3 → 6

```
arrayFind(int n)
{
  int arr[n+1];
  if(
  for(i=1;i<=n+1;i++)
  { if(i==arr[i])
      continue;
    else
      printf(i);
  }
}
```

Sol(2) $\sum_{i=1}^{N+1} i - \sum_{i=0}^{N} arr[i]$

Sol(3) $a \wedge b \wedge a = b$

```
int exOr(int n)
{
  int arr[n+1];brr[n+1];
    for(i=1; i<=n+1;i++)
     {
      for(j=1; j<=n+1;j++)
       {
       c=arr[i] ^ brr[j];
       }
      }
    return c;
}
```

→ Compute and return a^n $1e9 \rightarrow 1 \times 10^9$.

```
int compute(int a, int n)
{
  long int ans=1, m=1e9+7;
   for(int i=0;i<n;i++)
     ans=(ans*a)%m;
     return ans;
}
```

→ if you want to give size of an array as arr[1e6] then it will give an error as 1e6 is double type.

and size of an array cannot be a double so we have to typecast it.

arr[(int)leg]

leg is of type double.

Modulus arithmetic :

$(3*5)\%4 = 3 \Rightarrow 3\%4 * 5\%4 = 3$

$(7*6)\%4 = 2 \Rightarrow (7\%4 * 6\%4)\%4 = 2$

so, $(a*b)\%m = a\%m * b\%m$

$\rightarrow (a+b)\%m = (a\%m + b\%m)\%m$

$(3+5)\%4 = (3\%4 + 5\%4)\%4 = 0$

$(7+6)\%4 = (7\%4 + 6\%4)\%4 = 1$

$(a-b)\%m = (a\%m - b\%m + m)\%m$

$(10-3)\%4 = (10\%4 - 3\%4 + 4)\%4$

$= 2 - 3 + 4$

$3\%4 = 3$

** $(a/b)\%m = (15/3)\%4 = 15\%4/3\%4 = 3/3 = 1$

$a\%m / b\%m$

$(17/7)\%4 = 17\%4/7\%4 = 1/3$ (NO)

So $(a/b)\%m$ is not applicable.

Complexity Analysis of an algorithm:

prob — soln1
— soln2
— soln3

// For a no to implement it perfect square
no of divison.

```
int div(int n)
{
  int cnt=0;
  for(i=1; i<=sqrt(n); i++)
  {
    if(n%i==0)
      cnt++;
  }
  printf(cnt)
}
```

→ if I do not use the sqrt function then I would have used 'n' so it 'n' were 10^9 then

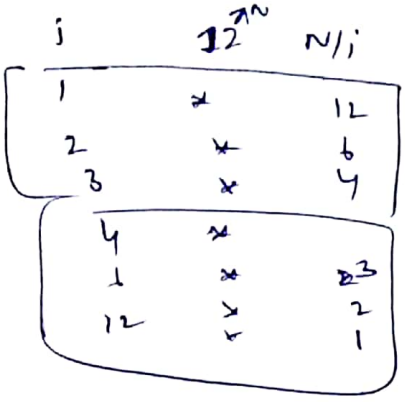
10^9 iterations

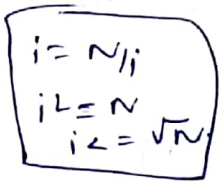
∴ 1GHz: 10^9 cycles
$= 10^9$ ins/sec

1 iter = 1 ins

running the code on 1GHz processor if there were 10^{18}

then for executing 10^9 instruction it would take 10^9 seconds
$= 31.7$ yrs.





Here you can see that we are not exceeding sq root of 12.

→ How to find complexity of codes theoretically.

```
f(n)
{
  for(i=0; i<=n; i++)  → O(n)
  {
    x++;               → O(1)
    y--;               → O(1)
    z = x+y+x*y;
  }
}
```

→ So in this the loop will iterate n times so we can consider the time complexity → O(n)

→ But if there is no for loop the time complexity will be O(1).

→ Now if there are two for loops what will be the time complexity.

① →

```
for(i=0;i<=n;i++)
{
 for(j=0;j<=n;j++)
 {
    O(1)
 }
}
```

→ Here the first loop will iterate n times. The second one will iterate n times when at every point of the first loop.

i	j	total
1	n+1	
2	n+1	
3	n+1	
:	:	
n	n	n^2

∴ The time complexity $= O(n^2)$

② →

```
for(i=1;i<n/2;i++)
{ O(1)
}
```

$O(n/2)$
$= O(n)$

Combining ① & ② in a single problem

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

→
```
for(i=1; i<N; i++)
{
  for(j=1; j<N; j++)
  {
    O(1)
  }
}
for(i=1; i<n/2; i++)
{
  O(1)
}
```

i	j	total
1	1-N	N
2	2-N	+N-1
3	3-N	+N-2

Neglect the coefficient
take only the highest power

$$\frac{N(N+1)}{2} + N/2$$

$$= O(N^2).$$

→
```
for(i=1; i<=n; i=i*2)
{
  O(1);
}
```

i increment 2
4
8
16
.
$2^k = n$
$k = \log_2 n$.

∴ The time complexity is

$$O(\log_2 n)$$

→ Consider the following.
$N = 10^6$, for a machine of speed 1GHz, 1 iteration = 1 instruction

$$\log_2 10^3 = \log_2 10 = 10.$$

Big O → N^3 $N^2 \log N$ N^2 $N \log N$ N $\log N$ 1

#Iterations → 10^8 $10^{12} \log_2 10^3$ 10^{12} $10^6 \log_2 10^6$ 10^6 $\log_2 10^6$ $N=30$ 2^{30}

$10^{12} \times (10)$

2×10^{13}

$\therefore$ 16 hr

$\frac{2 \times 10^{13}}{10^9} = 2 \times 10^4$ $\frac{10^{12}}{10^9} = 1000$ $\frac{10^6}{10^9}$ $\frac{2(10)}{10^9}$ $\frac{2^{30}}{10^9}$ $\frac{2^{6}}{10^9}$

time → $\frac{2 \times 10^4}{60 \times 60} = 5.5$ hrs $\frac{1000}{60 \times 60} = 16.6$ min 1ms . 20ns 1sec 31.7

→ Suppose there are m 3 arrays

each of certain size

having certain elements

3

2

1 2

3

1 5 6

5

1 2 3 4 5

→ So if we run a loop of size t which takes an array as input and prints it out then we get the time complexity as

$T \times$ N^2, N^3, $N \log N$ } m

$T \times m$

→ Assume that we are running these instructions on a 1GHz machine

10^9 instructions can be executed per sec

$T \times m \leq 1$ sec

$10^9 \leq 1$ sec

→ each iteration may take 5 to 10 steps

so $5 \times 10^9 \leq 1$ sec (not possible)

$T \times m \leq 1$ sec

So definitely $T \times m \leq 10^8$

→ Consider the following examples.

$1 \leq T \leq 10^2$

$1 \leq N \leq 10^3$

N^3

$T \times N^3$

$= 10^2 \times (10^3)^3$

$= 10^2 \times 10^9$

$= 10^{11}$

$\frac{10^{11}}{10^9} > 100$ sec

→ We need to pass the test case in 1 sec but the problem is taking 100 seconds hence it will fail.

→ $1 \leq T \leq 10^4$

$1 \leq N \leq 10^3$

$T \times N \log_2 N = 10^4 \times 10^3 \log_2 10^3$

$= 10^4 \times 10^2 \times 10$

$= 10^8 / 10^9 = 0.1$ sec

Hence this will pass.

$1 \le T \le 10^6$

$1 \le N \le 10^6$

$T \times \sqrt{N}$

$= 10^6 \times \sqrt{10^6}$

$= 10^6 \times 10^3$

$= 10^9 / 10^9 = 0.1$ sec

$N \le 10^{18}$

$10^6 \times \sqrt{10^{18}}$

$10^6 \times 10^9$

$= 10^{15} / 10^9 = 10^6$ sec for sure this will not pass.

→ Now we shall represent the entire complexity

Soln 1:- $3n^3 + 4n^2 + 1$ $O(n^3)$ put $n = 5$

$3 \times 125 + 4 \times 25 + 1$

$375 + 100 + 1$,

$= 476$

Soln 2:- $n^2 + 10000$ $O(n^2)$

$5^2 + 10000$

$= 10025$

} here 1 is better than 2

but if $n = 100$

$3(100)^3 + 4(100)^2 + 1 = 3 * 1000000 + \cdots$

$(100)^2 + 10000 = 20000$

then here ② is better than ①

→ This is the drawback of Big O notation.

→ As we know there are 3 cases in the complexity of an algorithm

Best case	Worst case	Avg case
1	N	N/2

Big-O: Puts an upper bound on the complexity of algorithm based on input size.

→ Consider a linear search where the no. to be searched will be entered by the user.

→ So in the worst case it may happen that he enters a no. which is at the last or not in the array.

→ So here all the elements will be searched hence the time complexity will be O(n) in the worst case.

```
fun (N, arr[N]){
 int a, b, arr[N], brr[N], c[N];
 double d1, d2;
 bool f1, f2;
  int mat1[N][N], mat2[N][N]
  }
```

$N^2 + 3N + 25$

→ So here the Θ time complexity will be $O(N^2)$.

$1 \le T \le 10^3$

$1 \le N \le 10^5$

```
loop(T)
{
    int mat[N][N]
}
```

$4 \times (10^5)^2 B = 4 \times 10^{10} B$

$= 40 \times 10^9$

$= 40 GB$

```
long int mat[N][N]
```

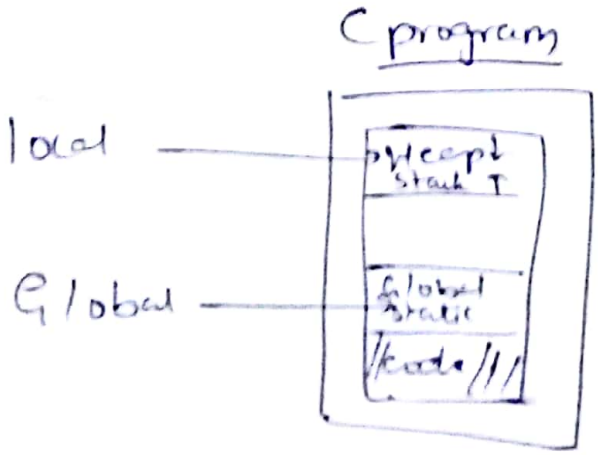
$8 \times 10^{10} = 80 GB$

bool mat[10^8] = 100 mb

$(x, y) \le 10^{6.7}$

→ Every time the loop starts executing the local variables are destroyed so we can take the space from 10^{1-7}.

and time limit should be 10^1



→ So basically the local variable, are stored in stack or heap & for global variable, are stored in Global static section.

→ Global variable can be given more space than local variables.

→ Find the missing elements in the array.

arr_N: 1 7 2 7 6 2 15 1 15

⟶ take ^ of all, common elements will get cancelled left over will be 6.

~~$x_i (= 2^k = [1,2,4,8,16...])$~~

~~$x_1 + x_2 + x_3 \dots x_m = 13$~~

$a^{13} = a^{x_1} a^{x_2} a^{x_3} = a^N = a^{x_1} a^{x_2} a^{x_3}$

$x_1 + x_2 + x_3 = N$

```
missingNum(arr[])
{
for(i=1; i≤n; i++)
 {
 missingNum ^= i;
 }
for(i=1; i<a.length; i++)
 {
 missingNum ^= a[i];
 }
 return missingNum;
}
```

#bit	x	ans
0	a^1	a^1
1	a^2	a^1
2	a^4	a^5
3	a^8	a^{13}

ans = ans * x
x = x * x

```
pow(a,N){
x = a;
ans = ans * x;
ans = 1;
while(N!=0) for(i = 2; i <= to Math.log(n)/Math.log(2); i++)
 {
  if((N>>i)&1)
   {
    ans = ans * x
   }
  x = x * x;
 }
```

time complexity for this will be

$\log_2 N$

$O(\log_2 N)$

(or,)

$O(N) = 10^9$ set

$N = 10^9$

$\log_2 10^9$

$= 3 \log_2 10^3$

$= 3 \times 0$ ns.

If $N = 10^8$

$= 60$ ns

→ In built power function works on $O(\log_2 N)$

$\log a * b$

$= \log a + \log b$.

→ Given that array when an input is given then the sum of the elements in an array should give the output as equal to the no.

ar: −5 6 3 15 −4

k = 17

15 + 6 − 4

= 17

	0	1	2
ar	7	10	5

[if 1 is in 0th position then 7 will be selected]

N=3

[if 1 is in the 1st position then 10 will be selected]

[if 1 is present in the 2nd position then 5 will be selected]

0	0	0	→ ϕ
0	0	1	→ 7
0	1	0	→ 10
0	1	1	→ 10,7
1	0	0	→ 5
1	0	1	→ 5,7
1	1	0	→ 5,10
1	1	1	→ 7,5,10

```
array fun (int, ar[10])
{
  int a[3]={7,10,15};
  int i=0,j=0;
     while (b[i]!=0)
     {
       for (i=0;i<=7;i++)
       {
          if ((b[i]>>j)&1)
            { ar[i]=1;
              }
            sum =sum+ar[i];
       }
     }
}
bool subset (ar[], N, k){                    → (?)
     {  s=0
     { for (int i=0; i<(1<<N); i++)
        if (cb(i,j)==1)
            s = s+ar[j]
            if(s==k)
              return True;
            } else return False;
```

Note:- Exponential soln. space works for small values.

2^n.

$= O(n*2^n)$

RECURSION

Recursion: It is a programming paradigm using which which takes different inputs within the same function.

```
int sum/fib(int n)
{
  int c=0
  for(i=0; i<n; i++)
  {
    c = c+i;
  }
 return c;
}
```

```
int fact(int n)
{
  if(n>=1)
   return n*fact(n-1);
  else
   return 1;
}
```

```
int sum(int n)
{
  int c=0;
  if(n>0)
    ret 0
   c = n+sum(n-1);
  else
  return c;
}
```

```
int fib(int n)      1,1,2,3,5,8,13
{
   if(n==1||n==2)
     return 1;
return fib(n-1)+fib(n-2)
```

```
int APsum(int a, int d, int n)
{
```

APsum(

To find the complexity of recursive code, we write recursive relations.

$T(0) = 1$

Time complexity of sum

$T(N) = T(N-1) + 1$

$= (T(N-2)+1) + 1$

$= (T(N-3)+1+1) + 1$

$= \vdots \quad \vdots$

$= T(N-(N-1))$

$= T(N-4)+4$

$= T(N-k)+k$

$0 = N-K.$

Time complexity of factorial

$K = N$

$O(N) = N$

$T(0) = 1$

$T(N) = T(N-1) * 1$

$= T(N-2) * 2$

$= T(N-K) * K$

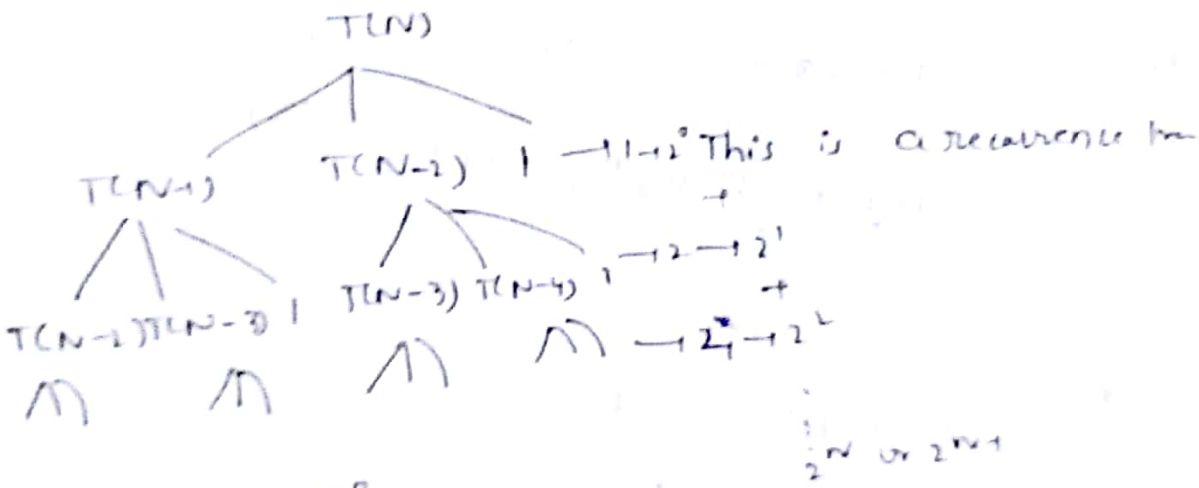
$0 = N - K$

$K = N$

$O(N) = N$

Time complexity for fibonacci series.

$T(N) = T(N-1) + T(N-2) + 1$
$= T(N-3) + T(N-4) + 1$ } so such method doesn't work



2^0

$$T(N) = 2^0 + 2^1 + 2^2 \ldots 2^{N-1}$$

$= (a = 1) \quad (r = 2)$

Time complexity for fibo :

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

$= O(2^n)$

Solve the recurrence relations.

1) $T(N) = 2T(N/2) + N$ $\boxed{T(1)=1}$

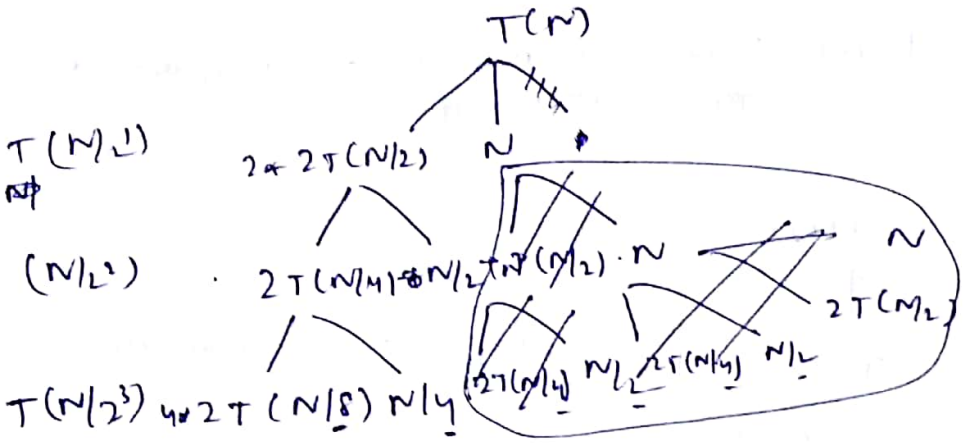
2) $T(N) = 2T(N/2) + 1$

3) $T(N) = T(N/2) + 1$

4) $T(N) = T(N/2) + N$

5) $T(N) = T(N/2) + T(N/4) + N^2$

1) $T(N) = 2T(N/2) + N$



$2T(N/$

T

$8T(N/8) + N/4 + 2N$

$8T(N/8) + N + 2N$

$8\ 2^k T(N/2^k) + 3N$ $\quad \therefore T(1)=1$

$2^k T(N/2^k) + kN$ ——— (1)

$1 = N/2^k$

$N = 2^k$

$k = \log_2 N$

~~$T(1) + N\log N$~~

Substi in eq (1)

$2^k T(N/2^k) + N\log N$

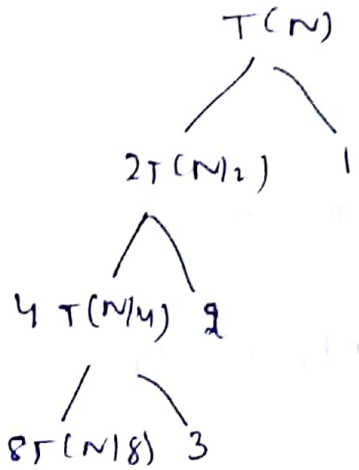
$T(1) + N\log N$

$N * 1 + N\log N$

$2^k = 2^{\log_2 N} = N$ $\quad O(N\log N)$

$\rightarrow \Theta(N)$

2) → $T(N) = 2T(N/2) + 1$



$2^k T(N/2^k) + 3$

$T(1) + k$

$N/2^k = 1$

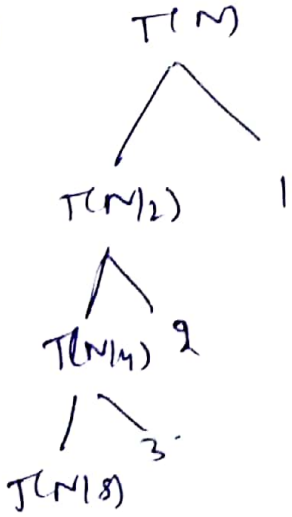
$N = 2^k$

$k = \log_2 N$

$2^{\log_2 N} = N$

$O(N)$

3)



$T(N/2^k) + 3$ $\quad \therefore (T(1)) = 1$

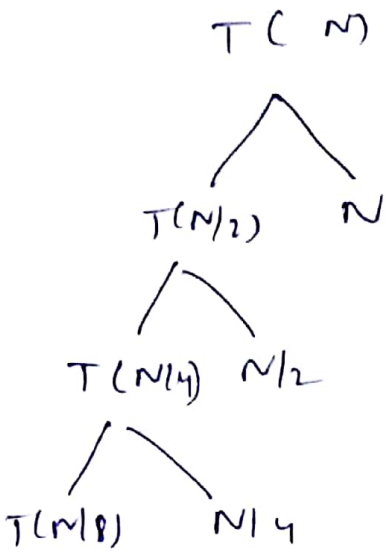
$N/2^k = 1$

$N = 2^k$

$k = \log_2 N$

$O(\log_2 N)$

4)



$= T(N/4) + 3N/2$

$= T(N/8) + 7N/4$

$T(N/8) + N + N/2 + N/4$

$T(N) = T(N/2^k) + \left(\frac{2^k - 1}{2^{k-1}}\right) N$

$\therefore T(1) = 1$ base case

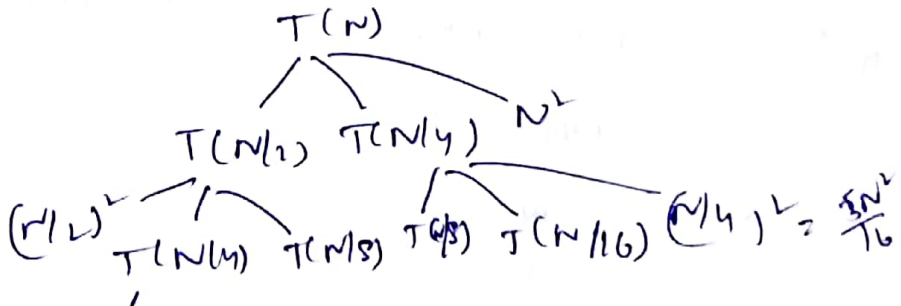
$\frac{N}{2^k} = 1$

$k = \log_2 N$

$1 + \left(\frac{2^{\log_2 N} - 1}{2^{\log_2 N - 1}}\right) N$ $\quad \because 2^{\log_2 N} = N$

$1 + 2 \frac{(N-1)\, 2 \cdot N}{N}$

$O(N)$



~~Error~~

$T(N) = T(N/4) + T(N/8) + (N/2)^2$

$= T(N/2^k) + T(N/2^{k+1}) + 3N^2/2^k$

$\frac{N}{2^k} = 1$

$N = 2^k$

$k = \log_2 N$

$\therefore T(1) = 1$

$T(1) + T(1) + \frac{5N^2}{2}\log_2 N$

$\frac{N^4}{2^k} = O(N^2)$

$N^2 + \frac{5N^2}{16}$ $N^2\left(1 + 5/16 + (5/16)^2 - \cdots\right)$

$N^2\left(\frac{1 - 5/16^k}{1 - 5/16}\right)$

$\frac{16N^2}{11}$

$= O(N^2)$

$N/2^{k+1} = 1$

$2^{k+1} = N$

$k+1 = \log_2 N$

$k = \log_2 N - 1$

Masters theorem

→ $T(N) = aT(N/b) + f(N)$

$f(N) = N^c$, $t = \log_b a$

① $c < t$: $O(N^t)$

② $c = t$: $O(N^t \log N)$

③ $c > t$: $O(N^c)$

① $T(N) = 2T(N/2) + N$

$a = 2$ $b = 2$ $f(N)$

$N^c = f(N) = N$

$c = 1$ $t = \log_2 2$

$t = 1$

$c = t$ ∴ $N \log N$.

② $T(N) = 2T(N/2) + 1$

$N^c = f(N) = 1$

$c = 0$ $t = \log_2 2 = 1$

$t = 1$ $c < t \Rightarrow O(N^1)$

Note

* Separate the terms
* pass them into the function
* at the end consider all the terms & add the constants also
* then equate with base
* substitute it in place of k. in the constant.
→ You will get Big Time Comp.

③ $T(N) = T(N/2) + 1$

$a = 1 \quad b = 2 \quad c = 0 \quad t = \log_2 1$

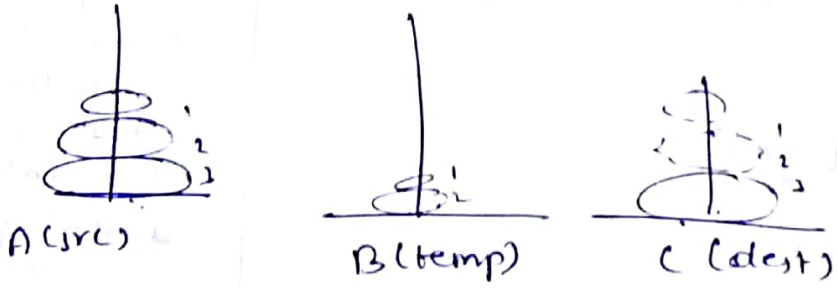
$c < t \sim O(N)$

⑤ $T(N)$ = cannot apply master's theorem.

$a > 0$ ∴ no negative or fractional values can be put.

$b > 1$ ∴ if $b = 1$ then it means we are not reducing the value of N and it will go to an infinite loop. Hence $b > 1$.

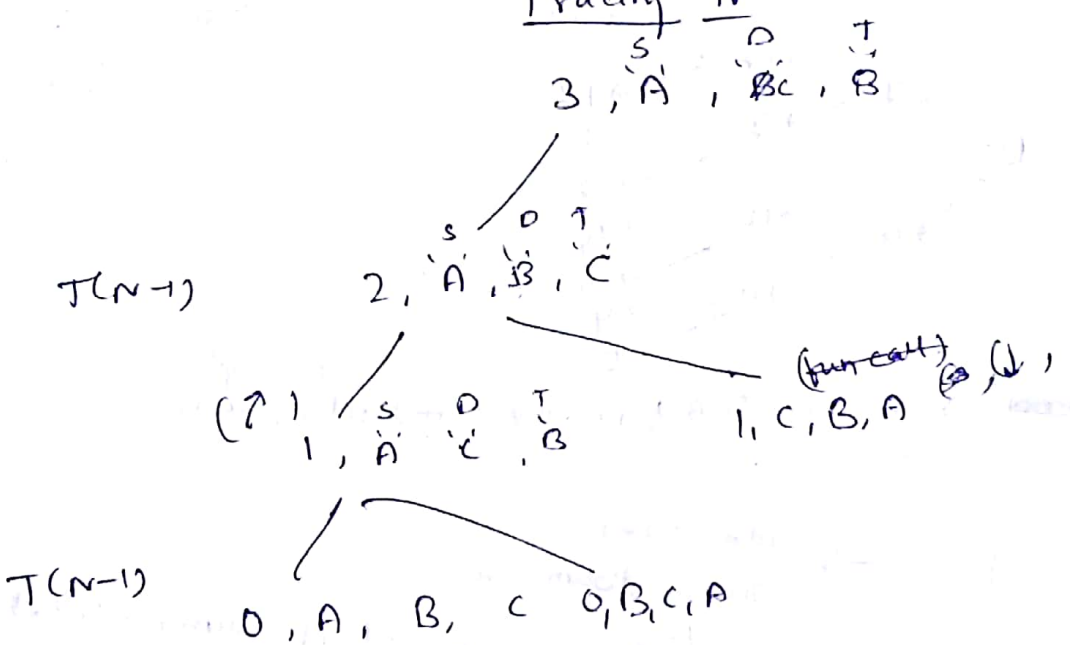
Towers of Hanoi



1: A → C
2: A → B
1: C → B
3: A → C
1: B → A
2: B → C
1: A → C

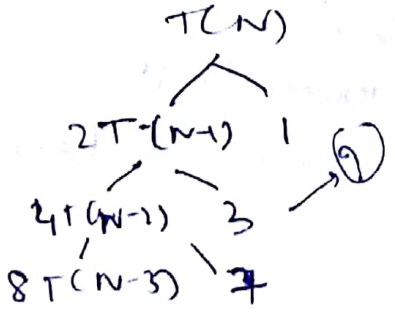
```
void TOH(int N, char src, char dest, char temp)
{
① if(N==0)
    return;
  else
  {
② TOH(N-1, src, temp, dest);
③ print("Nth disc : src → dest");
④ TOH(N-1, temp, dest, src);
  }
```

Tracing it



$$T(N) = T(N-1) + T(N-1) + 1$$

$$= 2T(N-1) + 1$$



$T(0) = 1$

$T(1) = 1$

$2^K T(N-K) + 2^K - 1$

$N = K$

$2^N + C$

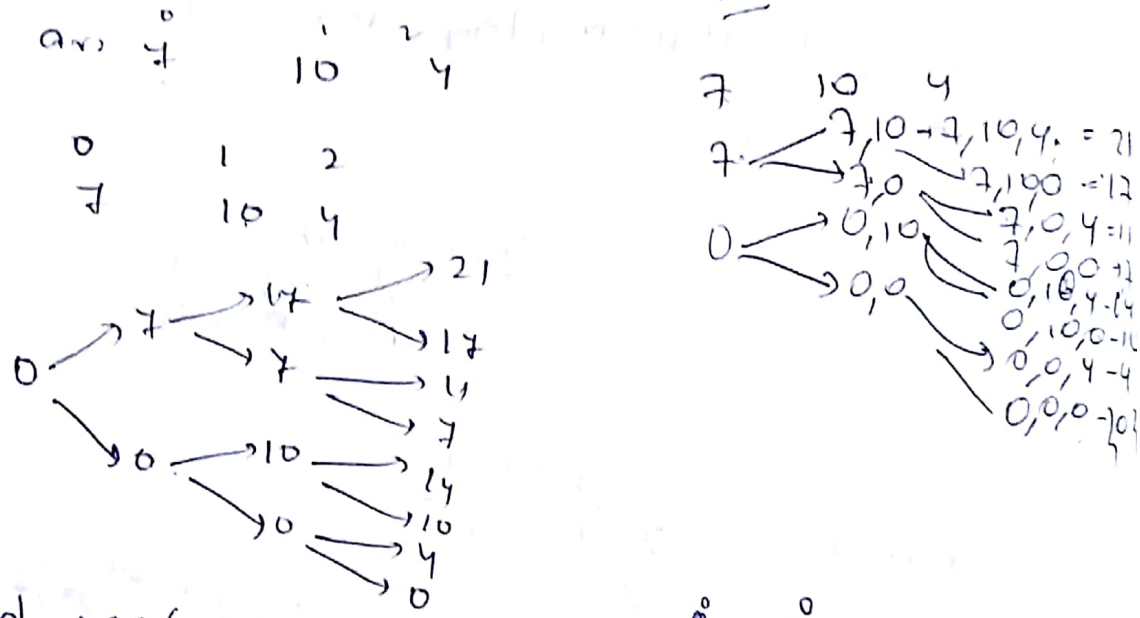
$2^{N+1}-1$

$2^{N+1}-1$

$\rightarrow O(2^N)$ $N=15$

$N=3$
then
$2^{3+1}-1$
$= 15$

→ Sum of the random numbers in an array is equ to given no.



bool rec(int[] ar, N, K, ~~sum~~ idx, sum)

{

1 ← if (idx == N)
return if(sum == k;)

N+1 ← return rec((ar, N, K, idx+1, sum+ar(idx)))

N-1 ← rec(ar, N, K, idx+1, sum))

} }

$T(N) = 2T(N-1)+1$ → (?) $T(0)=1$

$\therefore O(2^N)$

similar to towers to Hanoi we get $O(2^N)$

in iterative code we have $O(N \times 2^N)$

Balanced paranthy's

~(( ) ( )), ~((( ) ( )))

x (( )) (( )) ~

(( ) ( )) ( ) ( ) *

N = 4

( ) ( )

(( ))

N = 6

① ((( )))

② (( ) ( ))

③ ( ) ( ) ( ) op < c

④ (( )) ( )

⑤ ( ) (( ))

Code to print all the balanced paranthesis in lexeographical order

```
Void fun( ~~int N~~, ~~char ar[]~~, op, cl, indx )
{
    if (id == N)
        print ( );
    if (N % 2 == 0)
    {
        if ( op < N/2 )
        {
            ar[idx] = '(';
            fun(ar[], N, op+1, cl, indx+1)
            if (cl < op)
                fun(ar[], N, op, cl+1, indx+1)
        }
```

0 1 2 3 4 5

((( )))

(( ))

(( )) ( )

( ) ( )

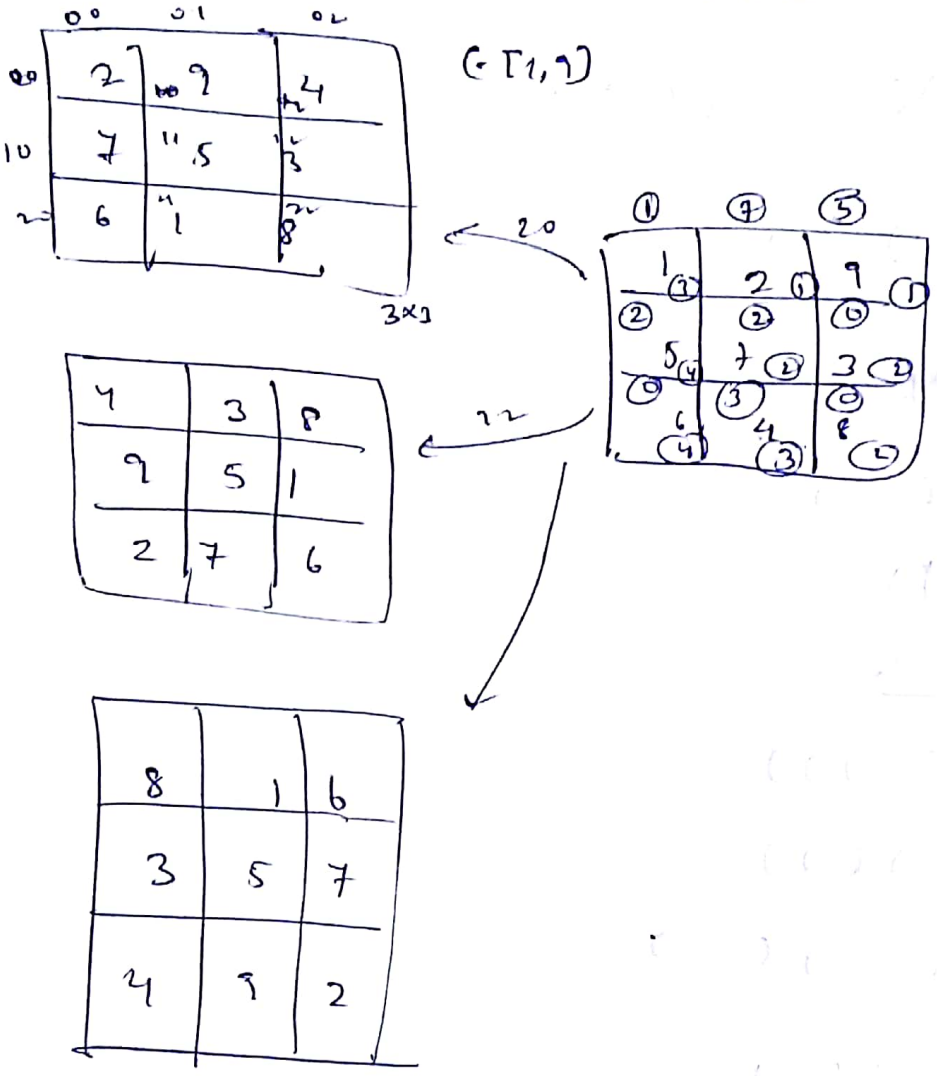
( ) (( ))

( ) ( ) ( )

3 2

2 < 3

Magic Square : Sum of all the diagonals, columns, rows should be same.



Convert a matrix using rec. fun calls using min no. of calls.

```
mat(indx, n*[], mat[])
{
    for(i=0; i<n; i++)
    for(i=1 to n)
        { if (m[i]==F)
            {
              m[i]=T ;
              mat[indx]=i;
        mat(indx+1, m[], mat);
              m[i]=F;
        if (indx==n)
            { if (check())
              return mat;
```

SORTING

Sorting means array in asc / inc / non-dec order

	W.C.T.C	S.C	B.C
BS →	N^2, 1		N^2, 1
SS →	N^2, 1		N^2, 1
IS →	N^2, 1		N, 1

1) BS

N ← 5 1 10 2 -3 8 4

N-1 ← 1 5 2 -3 8 4 10

N-2 ← 1 2 -3 5 8 4 10

1 2

$N + N-1 + N-2 \cdots 1$

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

$= O(N^2)$ ly

2) SS

5 1 10 2 -3 8 4

-3 | 1 10 2 5 8 4

-3 -4 | 10 2 5 8 1

$N + N-1 + N-2 \cdots 2+1$

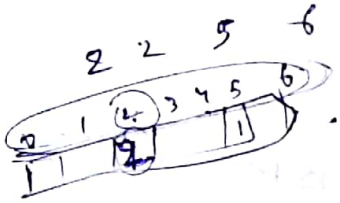
$\frac{N(N+1)}{2} = \frac{N^2}{2} + \frac{N}{2}$

$O(N^2)$

-3 -1 2 | 10 5 8 1

-3 -1 2 1 | 5 8 10

-3 -1 2 1 5 | 8 10



(9) Bubble sort can be optimised by maintaining the no. of swaps.

5 1 10 2 -3 8 4

```
int a[10], temp
for (i=0; i<n; i++)
{
  temp = a[i];
  loc = i;
  for (j=i+1; j<n; j++)
  {
    if (temp < a[j])
    {
      temp = a[j];
      loc = j;
    }
  }
  temp = a[i];
  a[i] = a[loc];
  a[loc] = temp;
}
```

Count sort

/* Code to sort an array */

```
void CS (int a[], int N, int m)
{
  int cnt[m+1] = {0};
  for (i=0; i<N; i++)
  {
    cnt[a[i]]++;
  }
  for (int i=0; i<m+1; i++)
    for (j=0; j<cnt[i]; j++)
      a[k++] = i;
}
```

N

M

O(M+N)

7 8 5 | 2 | 2

0 1

| 0 | 2 | 4 | 0 | 0 | 2 |

0

$[a,b] \rightarrow b-a+1$

$(a,b] \rightarrow b-a$

$[a,b) \rightarrow b-a$

$(a,b) \rightarrow b-a+1$

$10^9 \le ar[i] \le 10^9+100$

int cnt[21]={0};

10 → 0
9 → 1
8 → 2
⋮
0 → 10
1 → 11
⋮
10 → 20

$10^9 \le ar[i] \le 10^9+100$

```
cs(int ar[], int N, int m, int M)
{
  int cnt[m-M+1]={0};
  for(i=0; i<N; i++)
  {
     cnt[ar[i]-m]++;
  }
  for(i=0; i<M-m+1; i++)
  {
     for(j=0; j<cnt[i]; j++)
     { ar[j++]=i+m;
     }
  }
}
```

* We can

Note*

1) $R > 10^6-7$
↓
cannot use count sort because of space constraints.

2) Use count sort when $R <= N$

$m \le ar[i] \le M$.

ar:	-10	500	10000	4

HR → PS → Algorithms → Implementation
→ Migratory Birds
→ Picking Nos
→ Sock Merchants
→ Non-divisible subsets

Week-2
Day-2
19/8/2018

Sorting & Searching

- Linear Search
- Binary search.

→ Consider two arrays each with size N and M

A_N: −1 7 10 12 15

B_N: −6 −2 4 8 18 20 25 31

Print the elements in sorted order. /*both the given arrays are also in a sorted order. */

```
Sort(int[] a, int[] b)
{
        n, m
int i=0, j=0;
while(i<n && j<m)
 {
   if(a[i] < b[j])
   s.p(a[i++]);
   else
    s.p(b[j++]);
   }
while(i<n)
 s.o.p(a[i++]);
 while(j<m)
  s.o.p(b[j++]);
 }
```

Possible solutions

1) $N+M + (N+M)^2, (N+M)$

↓ Time complexity ↓ space complexity.

→ In this $N+M$ represents that array elements from array A which is of size N and array elements in B are to be copied to an array C.

→ Then the array C should be sorted so the complexity is $N+M+(N+M)^2$

2) So as we can see that both the arrays are in a sorted order so compare the elements then shift the other elements and place the particular element in that place.

T.C → N*M (Using insertion sort)

S.C → 1

3) $N+M, N+M$

This way we tried in our solution.

As the elements are in sorted order so pick the least of two and print.

picking from the first would take time of N and from second would take M

4) N+M, 1

↓ ↓

T.C S.C

Here we shall compare the elements from the end if it is greatest the place it in the array else compare with the rest.

⟹ Given an array of size N

$ar_{N=9}$:	10	3	12	5	-1	2	-6	71
	0	1	2	3	4	5	6	7 8

#(i,j) → i, j are indexes

$0 \le i \le j \le N$ → i is less than j and j is less than N

if $ar(i) > ar(j)$ → array of i element is greater than array of j.

Manually we get the results as:

5,-1	10,3	2,1	12,5	3,-1	-1,-6
5,2	10,2	2,-6	12,-1	3,2	
5,-6	10,-1	71	12,2	3,1	
5,1	10,-6		12,-6	3,-6	
	10,7		12,7		
	10,1		12,1		
	10,5				

(25) pairs exist.

1) N^2, 1 running represents running two for loops from 0 to N

1 is the space which means that we do not require any extra space for solving the problem.

→ still if any one digs into this and asks about 1 then say 1 is N+m which

is the space consumed.

⇒ Merge sort:

→ Merge sort

Given an array of n elements, using merge sort sort them

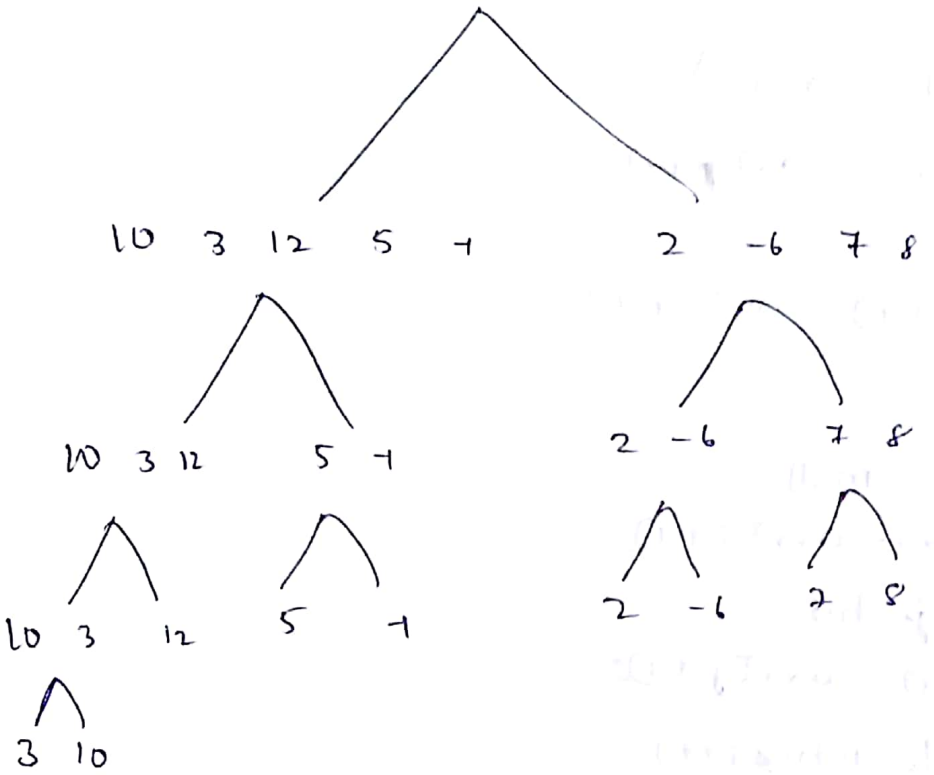
	0	1	2	3	4	5	6	7	8
$a_{N=9}$:	10	3	12	5	-1	2	-6	7	8

→ Merge sort is based on the principle of sorting the elements based on 'divide' and 'conquer' rule.

→ We keep on dividing the array into halves, until we get each and every element in a single way then we compare the arrays if the element is smaller then shift it to a temporary array else the next element will be shifted to the temporary array.

→ At the end a few elements will be left over shift them also into the array.

10 3 12 5 1 2 −6 7 8



Code:

```
void ms(arr[], lo, hi)
{
  if (lo == hi)
    return;
  mid = lo+hi/2; lo+(hi-lo)/2;        ∴ [a, b]
  ms(arr[], lo, mid);   → T(N/2)      N = b-a+1
  ms(arr[], mid+1, hi); → T(N/2)
  merge (arr[], lo, mid, high) → N
}
void merge (arr[], lo, mid, hi)
{
  int tmp[hi-low+1];
  int i = lo, j = mid+1, k = 0;
```

```
while (i<mid && j<=hi)
{
  if(arr[i]<arr[j])
    tmp[k++]=arr[i++];
  else
    tmp[k++]=arr[j++]
}
while(i<=mid)
  tmp[k++]=arr[i++];
  while(j<=hi)
  tmp[k++]=arr[j++];
  for(i=lo; i<hi; i++)
  {
    arr[i]=tmp[i-lo];
  }
}
```

Time complexity of merge sort:

→ The first function will take $T(N/2)$ as it is called twice it will be $2T(N/2)$

→ The last one will take $N/2+N/2=N$
∴ merging two $2(N/2)$'s

$$T(N) = 2T(N/2)+N$$

$$a=2 \quad b=2 \quad c=1 \quad t=\log_2 2 = 1$$

Applying Masters theorem we will get
$\therefore c=t$

$$N^c \log_2 N$$

$$= O(N \log_2 N)$$

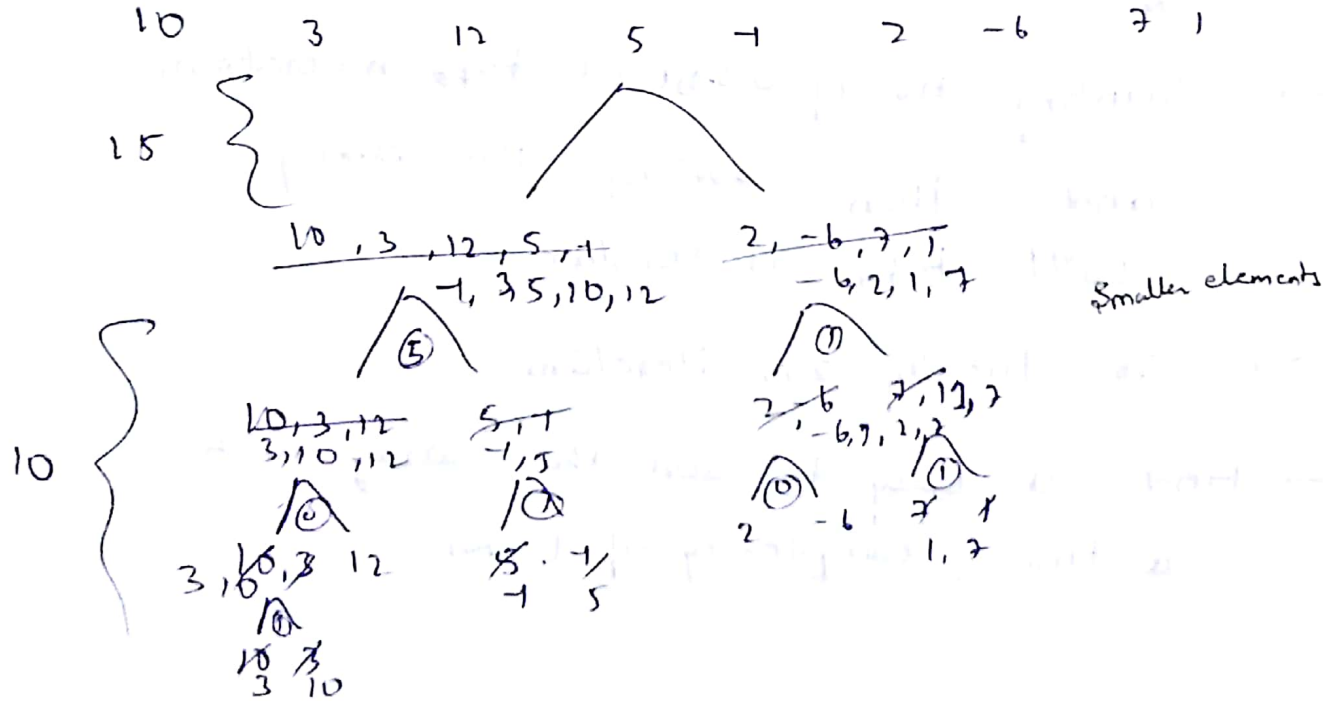
	0	1	2	3	4	5	6	7	8
arr N=9	10	3	12	5	-1	2	-6	7	1

→ Here we count the no. of swaps with the condition $i<j$ and $arr[i] > arr[j]$.

→ We will check the condition such that left side element should be smaller than right side element and then if true then swap it and increment the count by 1.

→ For a sorted array as there would be no swaps the count value will be 0.

→ For a reverse sorted array as all the elements have to be sorted the count value will be N^2.



→ So in the top part there will be 15 swaps and at the bottom part there will be 10 swaps.

→ Totally there would be 25 swaps.

→ By counting the no. of inversions we can calculate the degree of sortedness.

→ This will be helpful to find the data types of variables.

⟹ Consider an array of elements consisting of 0's and 1's. sort the array.

$0 \le i \le j < N$ $ar(i) < ar(j)$ then swap $O(N)$

ar_N: 0 0 1 0 1 1 0 1 1 0 1 1

→ counting no. of 0's, 1's will take n iterations and then sorting the array will take n iterations.

→ So totally 2N iterations

→ Find a way to sort the array with a time complexity of $O(N)$.

→ For this consider two pointers one at the starting and other at the end.

→ if the starting pointer found a '1' and at the same time the ending pointer which is decreasing found a '0' then swap them.

→ This will continue and at last we will get 0's followed by 1's hence it will be in a sorted order.

→

```
sort(int[] a, int n)
{  j=n-1;
  while(i<n && j>0)
    {
      if(a[i] !=0 && a[j]!=1)
        {  swap(a[i], a[j]);
           j--; i++;
        }
        else {
            j--; i++;
        }
    }
}
```

⟹ Consider an array of size N

arr: 10 -2 8 3 15 -6 12

K = 2, 17, 7

We have to select two ~~consecutive~~ elements such that their sum is equal to the given value K.

$arr[i] + arr[j] = k$ $i \neq j$

```
Sort(int[] a, k, n)
{
 for(int i=0; i<n; i++)
 {
  for(int j=1; j<n; j++)
  {
   if(a[i]+a[j] == k)
     return true;
   else
     return false;
  }
 }
}
```

→ As there are two for loops running the time complexity will be $O(N^2)$

SC → 1.

→ To decrease the time complexity we use merge sort to sort the array.

→ Then use two pointer technique find the sum such that sum of two elements

in the array is equal to the given element k.

Now suppose the array is in sorted order

$-6, -2, 3, 8, 10, 12, 15$

```
void sort(int[] a, int k, int n){
  {
    int p1=0;
    int p2=n-1;
    while (p1 >=0 && p2 <=n-1)
    {
      if (a[p1] + a[p2] <= k)
          p1++;
      else if (a[p1] + a[p2] == k)
          return true;
      else if (a[p1] + a[p2] > k)
          p2--;
    }
  }
```

⇒ Similarly now consider three elements of an array such that when they are added they produce a sum equal to k.

$$a[i] + a[j] + a[k] == k$$

$$a + b + c = k$$

$$b + c = k - a = x.$$

-6, -2, 3, 8, 10, 12, 15

→ fix the pointer at -6 and another pointer at 15 then check the result is x.

$$a[j] + a[k] = k - a[i]$$

→ If it works means if it is true then okay or else we need to increment the pointer to the right hand side by 1.

→ So in this way it will be $O(N^2)$

Total time complexity = $N \log N + N^2$, $N \rightarrow$ S.C

SEARCHING

1) Linear search : N, 1
   T.C ↓ S.C ↓

2) Binary Search

→ Write the recursive and iterative code for binary search.

```
bool recursiveBS(a[], low low N, k)
{
    low=0, hi=n-1
    mid = lo + (hi-low)/2 ; if(lo>hi) return false
    if(k == a[mid])
        return true;
```

```
else if ( k > a[mid])
  return BS(a, mid+1, hi, k);
  else if ( k < a[mid])
    return  BS(a, low, mid, k);
   }
```

Iterative code

```
bool  BS(a[], n, k)
{
  lo = 0, hi = n-1;
     while (lo <= hi)  =>T(N/2)
     {
       mid = lo + hi-lo/2;
       if ( a[m] == k)
         return true;
         if (a[m] < k)
           return m+1;
         if (a[m] > k)
            return
              hi = m;
         }
     return false;
    }
```

→ Time Complexity → $T(N) = T(N/2) + 1$

$a = 1 \quad b = 2 \quad c = 0 \quad t = \log_2^1$

$c \leq t$

$T(N) = N^c \log N$

$= N^0 \log N$

$= O(\log N)$

S.C → As no extra space is required the space complexity is 1.

→ If it is an unsorted array then we need to sort it.

→ We shall sort it by merge sort whose complexity is $O(N \log \frac{N}{2})$ then applying binary search we will get $O(N \log \frac{N}{2} + \log \frac{N}{2})$

→ If the array is already sorted then use binary search as the time complexity is $\log \frac{N}{2}$

→ If it is not sorted then use linear search as the time complexity will be $O(N)$.

→ $O(n) * \log \frac{N}{2}$ $O(n)$ represents the number of elements.

→ arr_N: −1 10 2 8 7 −6 15

k = 19, 5, −4

Consider the elements in the array.

→ Apply binary search such that sum of three elements is equal to the given k.

lo	hi	$\frac{lo+hi}{2}$	$lo+\frac{hi-lo}{2}$
2	8	5	5
2	9	5	5
−8	−2	−5	−5
−9	−2	−5	−6

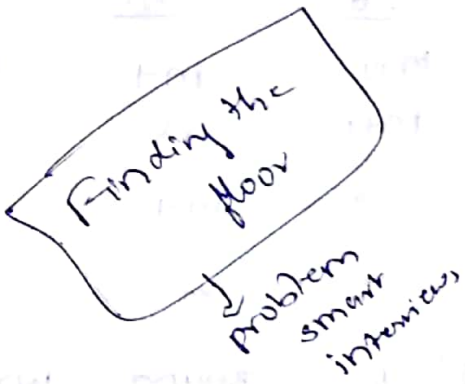
$lo+\left(\frac{hi-lo}{2}\right)$ is consistent as it gives correct values of mid even for negative ranges of numbers.

→ Hence use consistent formulae to achieve all testcases.

→ Consider the given array and find the greatest element less than the given no.

arr: → 10 2 16 −7 12 18 −11

i/p	o/p
5	2
20	18
−4	−7
100	18
−100	∝ (INT_MIN)



Code:

```
BinarySearch (arr, n, key)
{ if(key<arr[0]) :print(INT_MIN);
  while(lo<hi)
  {
   m = lo+(hi-lo)/2;
   arr[m] > key : hi = mid-1;
     else : lo=mid;
     }
     print(lo);
  }
```

Week-3

Day-1

29/8/2016

→ Binary search can be applied only if we know that the answer lies within the given range of values.

→ To apply binary search the elements should be in a sorted order.

→ Whenever you write low = mid then we have to consider ~~mid+1~~ upper mid.

→ To do so we have to write +1 in the mid.

lo	hi	m	while(?)
m+1	m+1	$lo+\frac{hi-lo}{2}$	lo ≤ hi
m+1	m	$lo+\frac{hi-lo}{2}$	lo < hi
m	m+1	$lo+(\frac{hi-lo+1}{2})$	lo < hi
X m	m		

→ If we round up mid then hi & mid will be same and it will be an infinite loop.

→ Never write the code as low = mid and hi = mid.

→ Then both lo & hi will be same and there would be an infinite loop.

→ Solutions for getting the output when inputs are given are:

1) $Q \times N$,1

compare with all the key with N

space complexity will be 1 as no extra space is required.

2) $N\log_2 N + \log_2 N$,1

↓ sort the array, ↓ apply binary search and get the value.

→ While printing the output we loose their order so we follow indexing to store the order.

3) $N\log_2 N + Q\log_2 Q + N+1Q + Q\log_2 Q + 3Q$

- $N\log_2 N$: sort the array
- $Q\log_2 Q$: sort the input elements
- $N+1Q$: search, 2 pointer technique
- $Q\log_2 Q$: sort on basis of index
- $3Q$: space for ans, index of array

→ Consider an array find the count of each element

ar_N: 5 1 3 10 3 5 3 3 8 12

10 1

I/P	O/P
3	4
5	2
15	0
8	1

Finding frequency of the given no. ↓ Use hashmap

The possible solutions for this are:

1) $Q \times N$, 1

Compare each and every key of the array with N and then inc. the count

2) $N+Q$, R

take a temporary array put all the elements into it then increment the count.
The space taken by it will be some extra R.

3) $N\log_2 N + Q(\log_2 N + N)$, 1

sort the entire array by mergesort.

Apply Binary search get the element increment the count by 1.

4) $N\log_2 N + Q\log_2 Q + (N+Q) + Q\log_2 Q$, Q

Sort the given array

Sort the set of i/p's

Apply two pointer technique

Sort the given set of ans. based on indexes

space

5) $N\log_2 N + Q(2\log_2 N)$, 1

Sort the given array

Apply two Binary searches get the starting and ending indexes P_1, P_2 then do $P_2 - P_1 + 1$.

... 1, 3, 3, 3, 3, 5, 5, 8, 10, 10, 12 ...

BS1: lo = m+1 ; hi = m ; hi = m-1

BS2: lo = m ; lo = m ; hi = m-1

```
int sqrt(int n)
{
   for(i=1; i<n/2; i++)
   {
      if( i*i == n)
         return i;
   }
}
```

→ If N = 25
Then we have to find the square root of 25

→ In built square root function use Binary search to find the square root.

lo	hi	m
1	25	13
1	12	6
4	5	3
4	5	4
5	5	5

	10^{11}	10^{36}
1) $\sqrt{N}$, 1	1sec	31.7yrs
2) $\log_2 N$, 1	60hrs	120hrs

$\log 10^3 = 10$

$12 \log 10^3 = 12 * 10 = 120$

$6 \log 10^3 = 66$

→ Consider an array of n elements where each no. element represents the time taken to complete a task.

arr: 5 6 8 1 4 2 10 3 7 1

k represents the no. of people involved

k = 3

Cabinet partitioning

means that there are 3 workers.

5 6 8 | 1 4 2 10 | 3 7 1

(19) (17) (11)

→ While assigning tasks to the workers we have to take care that they get tasks in a consecutive manner.

→ We have to find the minimum time in which the k workers can complete the task.

Solutions for this are

1) $\binom{N-1}{k-1} * N, 1$

2) $N \log_2 S, 1$

① $\binom{N-1}{k-1}$ The k-1 workers can be placed at N-1 locations. ∴ In between the numbers.

Adding them and finding the max will take N.

→ This was a brutforce code but making it easy use backtracking to solve this.

→ Add the elements move the pointers.

```
lo=S/k , hi=S;

while(lo<hi)
{
  mid= lo+hi-lo/2 ;
  if(valid(m)=='T')
  {
    hi = mid;
  }
  else :lo=mid+1;
}
return lo;
```

S → sum.

lo = sum / no. of people = avg.

```
valid(m, k)
{
  sum=0
  while(sum<=m)
  {
    sum = sum + arr[i];
    ret true
  }
  ret false
}
```

F F F . . . T T T
8, 9, 10 . x . 20, 21, 22 . . .

→ whenever we point if the ans exists then we mark it as true and move backward till the lowest true value. (hi=m)

→ N log S/2

No. of elements — For each sum binarysearch will run.

. . . F F F (T) T T T . . .

→ If we get TTTTFFFF... —
Then in this case we try to find the last true value.

→ Take care that the validation should not be as a brute force code.

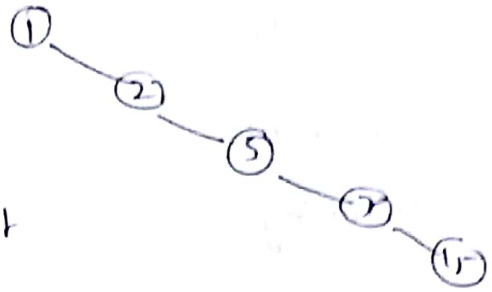
→ Consider a list of 10-digit mobile no's.

Soln	Insert(x)	Search(x)	Delete(x)
Unsorted	1	N	N
Sorted	N	$\log_2 N$	N
US SLL	1	N	N
S SLL	1	N	N
Stacks	1	N	N
DAT	1	1	1
BST	H	H	H

→ DAT means Direct Access Table.
→ Theoretically it is good but to implement it practically it is difficult.
→ In BST H is the height of the tree
→ In the worst case we require it will take O(N) as the time complexity

to insert the values.

→ In the best case it will take $\log N$ to insert the elements.

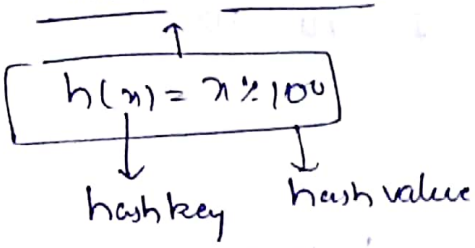


→ If I want to insert 15 then it would take the complexity as Height of the tree.

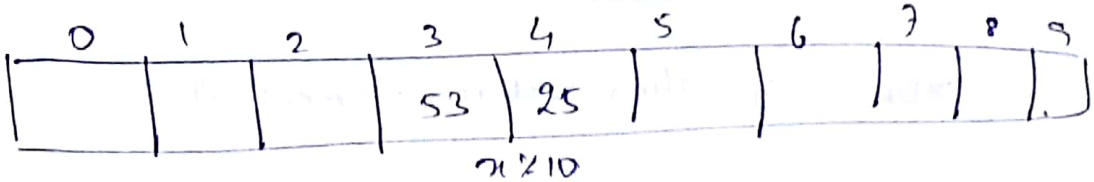
I: cnt(x) = T → To insert we increment count so T
D: cnt(x) = F → To delete we dec. the count so F.
S: cnt(x) == T → While searching if true then found.

→ Space is same so we do not declare bigarray.

Hash Function



Hash table: used to store the values



53 → 53 % 10 = 3

25 → 25 % 10 = 5

45 % 10 = 5

→ This type of problem where at the same position two values need to be stored is called collision.

→ It can be resolved by
  → separate chaining

  Open addressing:
  → Linear probing
  → Quadratic probing
  → Double hashing

Week-4

Day-1

→ Consider the array of elements find the ele which occurs only once.

arr: -1 -1 3 3 6 6 7 10 10

11 11 12 12 18 18

HINT: (0,1) (2,3) (4,5) (6,7) (8,9) (10,11) ... Indexes

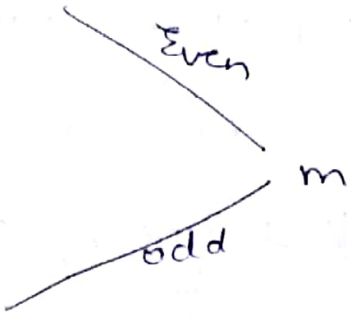
1) O(N), 1 Brute force code

2) N, 1 XOR operation between elements →?

3) $\log_2 N$, 1 : Binary search code:

```
while(lo<=hi)
{  mid = lo+(hi-lo)/2;
   if(a[mid+1] ≠ a[mid-1] ≠ a[mid])
        return a[mid];
```

```
if (ar[mid] == ar[mid+1])
    lo = mid+2
else
    hi = m-2;
if (ar[mid] == ar[mid-1])
    lo = mid+1;
else
    hi = mid-1;
}
}
```



→ Test case where this soln. will not work is

→ If the element which occurs only once is in 1st or last position. Then place a condition as

```
if((ar[0] == ar[1]) || (ar[N-1] == ar[N-2]))
    ret. F;
```

→ It even won't work if there is only one element in the array.

```
if (a.length == 1)
    return a;
```

→ Given a sorted array find the 1st missing positive integer.

① ar: 1 2 3 4 7 8 10 15 → 5

② ar: 1 3 4 5 6 → 2

③ ar: 1 2 3 4 → 5

④ ar: 3 5 6 8 10 → 1

Solutions:

1) K,1 → perform XOR operation

2) arr[i] == i+1 ; N,1

3) Modified Binary Search

Consider two pointers A and B pointing to the indices.

→ In (1) after A the no. exists so we consider index 2 and no. 3.

→ Take any one of them

A = lo+2
(or) ar(lo)+1
(OR)
B = lo+1
ar(lo-1)+1

(A)

```
while ( lo<hi)
{
  mid = lo + hi-lo+1 / 2 ;
  if (arr[m] = m+1)
   {
     lo = m ;
   }
   else hi = m+1 ;
}
return lo+2 ;
    (or)
return a[lo]+1;
```

if we consider pointer B (B) then

```
while(lo<hi)
{
    m= lo + (hi-lo)/2;
    if (arr[m] == m+1)
    {  lo = m+1;
    }
    else
    {
      hi = m;
    }
}
return lo+1;
   (or) arr[lo-1]+1
```

→ A fails for (4)

Handle A

```
if (arr[0] != m+1)
    return m+1
```

→ B fails for 3

to solve this

```
if (a[N-1] == N)
    return N+1;
```

→ Given a sorted array with +ve and -ve elements, find the first +ve missing element. (6)

	0	1	2	3	4	5	6	7	8	9	10	
ar:	-5	-2	-1	1	2	3	4	7	8	10	15	→ (5)
ar:	-10	-8	0	1	3	4	5	6				→ (2)
ar:	-6	-3	1	2	3	4						→ (5)
ar:	-8	-5	-2	3	5	6	8	10				→ (1)

1st Binary search
Figure out the index where the last negative number is located

```
while(lo<hi)
{
  m = lo+hi-lo/2;
  if(ar[m]==m-idx)
    lo = m+1;
  else
    hi=m;
}
return lo;
```

(2)

2nd Binary search should be applied to find the missing element.

```
while(lo<hi)
{
  if(ar[m] =< 0)
    lo=m;
  else
    hi=m-1;
}
return lo;
```

(1)

→ Consider an array

ar: 4 6 7 10 12 18 19 26 32 39 40 46 48 52 55

N=15

k=6

→ Given a 1-D plane. You have to figure different co-ordinates such that each co-ordinate has one house. You have to place k the relatives such that you maximise the distance between relatives.

'N' is the no. of houses

'k' is the no. of relatives.

Solutions:

1) ${}^{N}C_{k} \times k!$

N relatives can be placed in k locations × max. distance b/w k relatives.

distance → 8 9 10 ... (F) 30 31 32

T T T ... (T) F F F

→ We have to find the last truth value (T) in-order to get the maximum distance at which relatives can be placed.

```
while(lo<hi)
{
  mid = lo+hi-lo/2
  if(valid())
    lo=m;
  else
    hi=m-1;
} return lo;
```

```
valid(arr,k,mid,lo)
{
  while(c<k)
  {
    if(arr[j]-arr[i]>=0)
```

Complexity of this code

valid fun. will take N.

$N * \log_2 R, 1$

valid function

valid (arr,

→ Given two arrays find the medians

arr A_N: 1 3 8 12 18 25 31

arr B_N: 2 4 6 13 17 19

Solution

$N+M$ = odd

1) C_N: $(N+M) \log_2 (N+M), N+M$

↓ merge. ↓ apply B.S

2) $N+M, N+m$

↳ Merge both the arrays using merge function

3) $N+m, 1$

To optimise space

if (a[i] < b[i]) then

c++

when c == n/2 print the

value of median.

lo	hi	m	
1	31	16	$1+\frac{31-1}{2} = 15$
1	15	8	$1+\frac{15-1}{2} = 8$
9	15	12	

4) $\log_2 R (\log_2 N + \log_2 M), 1$ → (?)

After we calculate the mid we have to perform B.S on A and B.S on B

→ If there are duplicates

AN: 1 3 8 8 8 12 18 25 31

BM: 2 4 6 8 8 13 17 19

→ For each array we have to find the median so

→ here median is the smallest value than mid.

→ If mid exists then median will be mid.

→ Here median, mid hi and lo will be 8.

Complexity $\log_2 R (2 N \log_2 M), 1$

---

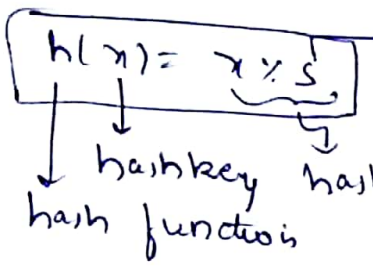
5) $N \log_2 M + M \log_2 N, 1$

Find the no of elements greater than 1 and less than 1 and then print the element which has a equal no. on both sides.

Balanced Binary Search Tree

	BBST	HF	DAT (Direct Access Table)
Insert (x)	H / $\log N$ — N	1	1
Search (x)	H	L	1
Delete (x)	H	L	1
Size —	1	1	1

→ To insert duplicate in linked list we have to check if it is present or not for that we need to do comparisons.



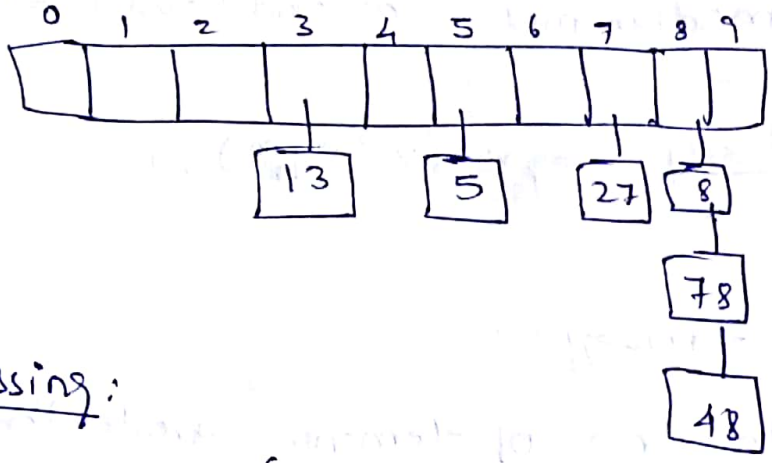
13, 18, 5, 27, 78, 68, 23, 48, 27

Collision Resolutions :-

① separate chaining: (use linked list to store the values)

s = 10.

h(x) = x % 10



② Open addressing:

Linear Probing: $h(x) = (x+i) \% s$ — (LP)

After finding empty slots put the element whenever you delete something put 'D' and if slot is empty put 'E'.

→ While searching our search should stop at empty slots.

→ This is done in order to avoid stopping search for an element.

0	1	2	3	4	5	6	7	8	9
D	48	E	13	23	5	E	25	18	78

→ Here instead of storing the element in a linked list they will be stored in next slot(i)

Quadratic probing:

$h(x) = (x + i^2) \% 5$

→ 0, 1, 2, . . .

0	1	2	3	4	5	6	7	8	9
		68	13		5			18	73

Advantage of quadratic probing over linear probing:

LP: 12 (0), 112 (1), 2012 (2), 1112 (3), 312 (4), 812 (5), 13 (5), 113 (6), 2013 (7), 1113 (8), 313 (9), 813 (10)

So, 60 probs are required in L.P

In QP: 12, 112, 2012, 1112, 312, 812, 13, 113, 2013, 113, 313, 813

So, 36 probs are required in Q.P

Double hashing

$h(x) = (h_1(x) + h_2(x)) \% 5$

$h_1(x) = x \% 5$

$h_2(x) = ①(x/5) \% 5$

②(x+15) =

No. of collisions in double hashing will be less than that in linear probing and quadratic probing.

$h_1(x) = h_1(y)$

$h_2(x) = h_2(y)$

is a very rare case in double hashing.

→ $h(x) = h_1(x) + i^2 h_2(x)$

$h_1(x) = x \% s$

$h_2(x) = (x/s) \% s$

→ To reduce the no. of collisions hash function should follow two properties.

① Uniformly distribute the keys over entire hash table.

② Simple to compute.

Suppose

$S = 100 \rightarrow$ slots

$N = 500 \rightarrow$ Numbers

The best way to put 500 numbers in 100 slots is 5 in each

→ Hash function should be simple to compute.

Ex: adding two digits

4 8

4+8=12

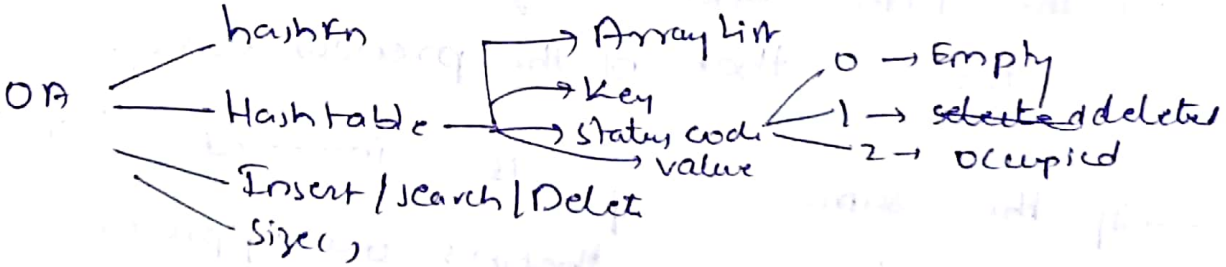
→ if take higher values like (x!)% s then it will take very long values.

→ While allocating space we must take care that the space allocated is greater than the number.

SC	OA
S>N.	10N.

→ Using a large space in open addressing helps to perform insert, delete, search operations very fast.

→ Hence performance also increases.



Insert(x),
Search(x),
Delete(x),
} O(1) Avg. Case

Hashmap & Hashset
unordered

Treemap & Treeset
sorted → (log N)
BBST

JAVA : ArrayList
C++ : vector
Python : [ ]
C# : ArrayList

} .add .push_back .append .add

→ How are the pre-defined libraries such as ArrayList implemented.

→ declare an array of size 10

arr[10]

when 10 elements are put

then array is full

then a new array will be created

→ The RAM and its size will be doubled than that of the previous array.

→ If the same array is increased then it may happen that it is out of process.

→ Asymptotic analysis of an algorithm:

→ If there are 10^6 numbers and I have created an array of size 1.

then the no. of operations to be performed are $10^6 = 2^{20}$.

$2^0, 2^1, 2^2 \ldots 2^{20}$.

copying the elements $1+2+4+8+16 \ldots 2^{19}$

$1+2+4+16 \ldots 2^{20}$.

$2(2^{20}-1)$

$\therefore 2^{22}$ operations are to be performed in order to insert 2^{20} elements.

Amortized analysis:

$2^{20} = 20$ operations.

→ when 2^{20} elements, i.e 10^6 elements are to be inserted then 11th & 21st element will be having a complexity of O(n) but we neglect it because all the other elements will be having a complexity of O(1).

Built in hash-map code:

```
class Key{
  String key;
   Key(String key)
   {
      this.key=key;
   }
   @override
```

```
public int hashCode()
{
    int hash = (int)key.charAt(0);
    System.out.println("hashCode for key:" +key+
            "=" +hash);
        return hash;
    }
public boolean equals (Object obj)
    {
        return key.equals ((Key) obj).key);
    }
}
public class CFG{
    psvm(String args[])
    {
    HashMap map=new HashMap();
    map.put(new Key("vishal"),20);
    map.put(new Key("sachin"),30);
    map.put(new Key("vaibhav"),40);
```

```
System.out.println();
System.out.println("Value for key vaibhav: " + map.get(new
        Key("vaibhav"));
S.O.P("value for key sachin: vaibhav: " + map.get(new Key("sachin"));
  }
}
```

Output:

hashCode for key: vishal = 118
hashCode for key: sachin = 115
hashCode for key: vaibhav = 118

hashCode for key: sachin = 115
Value for key sachin: 30
hashCode for key: vaibhav = 118
Value for key vaibhav: 40

→ There is even something known as threshold value which may be 0.75 or 0.5 which means that after 75% or 50% of the table is filled create a new array, double the size of previous one.

if (x is a negative no.) then
h(x) = (x%5 + 5)%5

ASCII code values

'a' → 97	'A' → 65	'0' → 48	: →
'b' → 98	'B' → 66	1 → 49	. →
⋮	⋮	⋮	, →
'z' → 122	'Z' → 90	'9' → 57	

205+97+++2+3+1

Rahul = 25%s → store

Raj = index%s

Manish

Abhishek

sum the ASCII values for each character and then store it.

→ Write a better hash function to reduce the no. of collisions and even simple enough to store a string.

A) $h(str) = ① \sum_{i=0}^{N-1} str(i)\%s$ → summation of ASCII values

② strsize()%s

③ $\sum_{i=0}^{N-1} (i+1) * str(i)$

abbcd → 1a+2b+3b+4c+5d

addcb → 1a+2d+3d+4c+5b

1*65+2*61+3....

④ (97|98|99)%s

(97|98|99)%s

⑤ $hash(str) = \sum_{i=0}^{N-1} (2^{i+1} * str(i))\%s$

$h(abcd) = 2^1 * a + 2^2 * b + 2^3 * c + 2^4 * d$

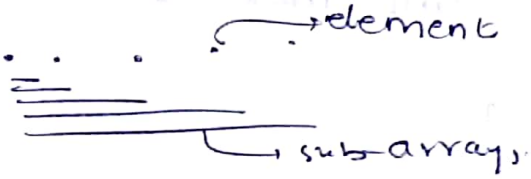
→ Whenever you wish to hash a string hash it with fifth function it would give less collision.

→ Going with 2i+1 means giving weight to first character.

⇒ Given an array find the maximum sub-array sum.

arr: 1 5 -2 3 -15 10 6 -4 8 1 -25 4

maximum no. of sub-arrays $\frac{n(n+1)}{2}$



starting with 1 we get N
starting with 2nd we get N-1
and so on ...

Brutforce code :

```
N3,1
    ans = INT_MIN;
    for(i=0;i<n;i++)
    {
      for(j=i;j<n;j++)
      {
        sum=0;
        for(k=i;k<=j;k++)
        {sum = sum+a[k];
         ans= max(ans, sum);
        }
      }
    }
```

Trying to reduce the time complexity as it is N^3, 1

→ Take the previous sum.

2) N^2, 1

```
for(i=0; i<n; i++)
{
  sum=0;
  for(j=i; j<n; j++)
  {
    sum= sum+a[j];
    ans= max(ans, sum);
  }
}
```

3) N, 1 → we shall learn in dynamic programming

→ Subsets: The set of elements from a set is called as subset. 2^N subsets are possible. They are not ordered (unordered)

→ Subsequence: The set of elements from a set is taken in an order is called as a subset.

- They are ordered based on indexes
- for an array n. 2^n subsets are possible.

→ for an array of size n we will have 2^n subsequences

arr. 5 3 7 -1 6 7

$2^6 = 64$

(5,3) (5,7) (5,-1) (5,6) (5,3)

(5,3,7) (5,3,-1) (5,3,6) (5,3,7)

(5,3,7,-1) (5,3,7,6) (5,3,7,7) (5,3,7,-1,6,7)

(5,3,7,-1,7) ..... these are subsequences

But we need the subsequences in a.o

(5,7) (5,6) (5,7) (7,7) (6,7)

(3,1) (5,3,8) (-1,6) (5,6,7)

(3,7) (3,6) (-1,6,7) (-1,3,6,7)

(3,7,7) (6,7)

Code:

```
for(i=0;i<n;i++)
{
  for(j=i;j<n;j++)
  {  c=0;
     for(k=0;k<=j;k++)
     { if(arr[i]<arr[j])
           c++;
     }
  }
}
```

→ Consider an array of n elements find the longest sub array whose elements can be arranged in an order.

ar_N: 5 3 8 10 9 7 6 -5 -10 -13 -11 -12 25

1) $N^2(N\log_2 N + N)$, N → store it in temporary array
   - N^2 → No. of sub-arrays
   - $N\log_2 N$ → sort the array
   - N → validate it.

   $\Rightarrow O(N^3 \log N)$

2) $N^2(N+N)$, N $\quad O(N^3)$
   - using the previous sorted elements put the next element in between carry forward the array to get new sub-array.

3) $M - m + 1 == len(subarray)$ → $[j-i+1]$: $N^3, 1$
   - M → max
   - m → min

4) $N^2, 1$ Code:

```
for(i=0; j<n; i++)
{
   m = m = arr[i];
   for(j=i; j<n; j++)
   {
      m = max(m, arr[j]);
      m = min(m, arr[j]);
      if(m*-m+1 == j-i+1)
         ans = max(ans, j-i+1);
   }
}
```

$\Rightarrow$ Given arr N: 1 5 4 2 8 3 7 6 5 7 4 12 19 14 17 16 17 18 20 19

Find the maximum sub-array whose difference will be 1.

Solutions:

1) $N^2 (N \log_2 N + N)$ → doing operation
   ↳ sorting it
   ↓
   No. of subarrays

2) $N^2 (N+N), N$
   ↓
   using the previous sorted array elements

→ As soon as you find the duplicate the inner for(j) loop breaks.

→ So, for this we store the elements in a temporary array that is in the hash set.

→ Then we check the elements which ~~are~~ we want to add are present inside the array or not.

→ If it is present inside the hash map then break.

→ If it were an array then we should have iterated it from starting onwards.

3) $N^2 (N+1), 1$
   → min & max. updations
   ↳ search in the previous & break
   ↳ No. of sub arrays

4) $N^2 (1+1-1), N$
   → insertion
   → update min & max
   ↳ in an unsorted array check duplicates are present or not.

Code:

```
for(i=0;i<n;i++)
{
  m = m = arr[i];
  HashSet<int> hs = new HashSet<Integer>();
  for(j=i; j<n;j++)
  {
    if(hs.find(arr[j])
      break;
    hs.add(arr[j])
    m = max(m, arr[i]);
    m = min(m, arr[j]);
    if(m+m-1 == j-i+1)
      ans = max(ans, j-i+1);
  }
}
```

⟹ Given arr: 1 5 4 2 8 3 7 6 5 7 4 12 19

14 17 16 14 18 20 19 17

Find the max. ~~length of~~ sub-array a whose dif. b/w elements is 0 or 1.

1) N^2 ($N\log_2 N + N$), N

2) N^2 ($N+N$), N

```
for(i=0 to n)
{
  m = m = arr[i];
  for(j=i;j<n;j++)
  { m = max(m, arr[j]);
    m = min(m, arr[j]);
    if((m-m+1) == mbize())
      ans = max(ans, j-i+1);
  }
}
```

Brutforce code

```
for(i=0 to n)
  for(j=i to n)
    if(a[j]-a[i] == 0 ||
       a[j]-a[i] == 1)
      print a[i];
```

given arr_N: 5 −7 1 2 1 2 8 2 6 −10 2

K=4

For a given size of a window find the distinct no of elements.

1) $(N-K+1)\,K^2, 1$ → comparing the K element for distinct
   └ size of the sub-array

2) $(N-K+1)\,\overbrace{K, K}$ → ∴ hashset will only contain distinct elements no need to compare
   └ insert the elements into hashset

3) $K+(N-K)\,(1), K$ {(5,1), (1,1), (−7,1), (2,1)}

Insert the elements in hash map.

Week 5
Day 1

STRINGS

→ String is an array of characters.

'A' → 65 'a' → 97 '0' → 48 ... →
'B' → 66 'b' → 98 '1' → 49 , →
⋮ ⋮ ⋮ ; →
'Z' → 90 'z' → 122 '9' → 57

to print all the ASCII characters

```
for(i=0;i<255;i++)
{
    System.o.p(i + "=" + str(i));
}
```

→ To calculate the frequency of each character in a given string

str: a y z s m y x y a m n s t ....

'a' → 2
'b' → 0
⋮
'z' → 1

```
int cnt[26]={0};
for(i=0;i<n;i++)
    cnt[str(i)-'a']++;
```

→ | 1 | 2 | ... |
a b

→ Find the 1st repeating character

① str_N: m y z t g s t y o p z
1 1 1 1 1 1 0

(1) y (2) t

② Str N: a b c d d c a y 3
① ① ① ① -

A	B
a	d

Case B: put the elements in into a hash set if not present and if present then print.

```
for (i=0 to n)
{ if (hs.contains(ch[i])) ret c;
  else hs.add(ch[i]);
```

For this we will get two type of solutions.

For case A: put a pointer at starting and move it forward till the same character in string appears.

For case B: cnt[str(i)]++; smlr question.

→ Find the length of longest paliondroming substring

str N: s y z x x z y z x x z m n o o n m n o g n m

Solution: 1) $N^3, 1$

↳ Iterating one pointer starting from start and another pointer starting from end search for the similar one and count the length.

p1=0 p2=n-1

p1++ p2--

while(p1<p2)till.

2) If I point at the centre decrementing the pointer P_1 and incrementing the pointer P2. Each and every character will be pointed once. Once they become mid then iterating towards left and right so $N^2, 1$ for odd length.

3) $N^2+N^2,1$ $p1=i$, $p2=i+1$

If found th same then increment the other pointer store the value ~~and at the~~ end & print the length.
calculate the max.

→ to count the no. of paliondromic strings

$N^2,1$

put it at the middle check towards left and right.

→ Count the no. of substrings which can be re-arranged to form a paliondrome.

str N: s y z x x z y z x x z m n o o n m n o q n m

↓

This is not a sub-string but if it is rearranged then it can form a sub-string.

xzyzx

→ Count of all the characters should be even and one character should be odd in an odd substring and no of characters can be even in the even substring.

Solutions:

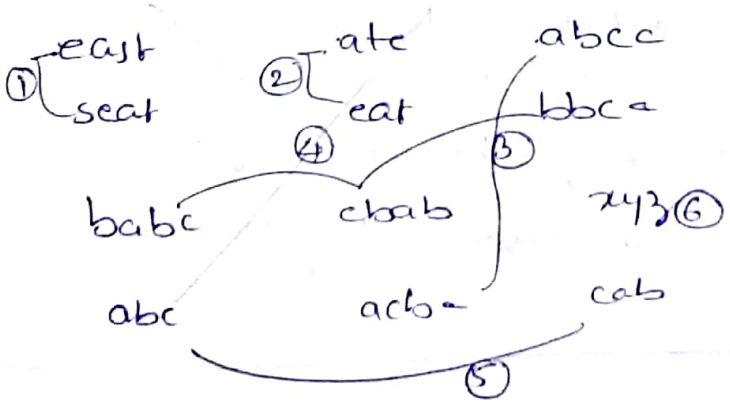
1) $N^3,1$ — i=0 to N
j=i to n
~~k=i to j~~ k=i to j
{ cnt[str[kj-'a']]++;
}
}

2) $N^2, 1$ →
```
{i=0 to n
 {j=i to n
  cnt[str[j-'a']]++;
 }
}

odd=0;
for(i=0;i<n;i++)
 { if(odd!=2)
 { if(cnt[i]%2==1){
        odd++;
   }
  else
   break;
  }
}
```

ANAGRAMS:- Given a string of any permutations of that string will be an anagram.

{abcd
{bcda
{cdbc

→To check how many of the given strings are same



To check how many of the given strings are same (Q2) →

{east, seat} {ate, eat} {acba, bbac}

{baba, abc} {abab, abca} {xyz, aab}

Solutions: (Q)

1) N, 1

2) $N \log_2 N * N$

```
C1[A[i]-'a']++
C2[A[i]-'a']++
for(i=0;i<25;i++) C1[i]==C2[i]
```

3) XOR (✗) T: e^a^s^t^s^e^a^t=0)

→ It doesn't work when the given strings are aa bb a^a^b^b=0

4) ASCII values (✗) e+a+s+t = s+e+a+t

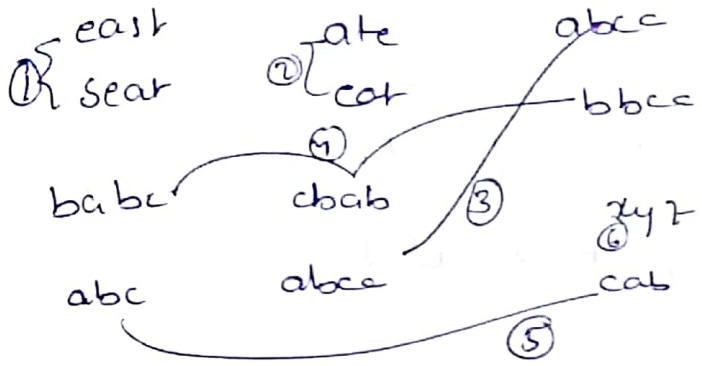
a+d=b+c

a+d=b+c

ASCII doesn't work for this as it gives the same value even if they are not palindromes.

→ Given find the no of groups formed for a given list of n words.

see that all of them are anagrams



Solutions:

1) $N^2, 1$

2) $N\log_2 N * N, N$

↓

sort the elements and insert them into a hashset size of the hash set will give you size of the group.

→ Find the no of paliondromic subsequences

str n: x s y z m s t y z s x m z z y x

x y y x
m s s m
y z z y

1) $2^N * N, 1$

2) Dynamic programming you will see

→ Find the smallest substring of A which has all characters of B.

A_N: s m a d r y a s t y z d a y a m t d r y

B_m: d a t a

Solution:

1) $N^2(mN)$, N ← comparing; ← space required for count array

$\sum_{i=0}^{25} C_A[i] = C_B[i]$

↳ substring of A

2) $m + N^2(N+26), 1$ → count array of A (?)

3) m ← create count array of B
$+ N^2(1+26), 1$ → count array + carry forward
↳ substring of A carry forward.

4) $\log(N-m+1) * N, 1$
Less than the given number nonsmaller no.

of subsets are not possible and greater than that no. in largest no. of subsets are not possible.

4	5	6	7	...	10	11	12	13	14
F	F	F	F		T	T	T	T	T

→ So, here we have to find a point of 1st truth value

```
lo = m , hi = n;
while (lo < hi)
{
  m = lo + (hi - lo + 1)/2 ;
  if (valid(mid) == T)
    hi = mid ,
  else
    lo = mid + 1 ;
}return lo;
```

N, 1 → 2 ptr technique

Consider the two strings check whether string B exist in A or not

A: x y s a s m a r t y s m s a

B: s m a r t

Solution: (N − M + 1) × M, 1 (hashA, no of elements, hashB, no of elements, No of elements in B)

2) KMP (topcoder)

3) Robin-Karp
↓
RollingHash

→ Both KMP and Robinkarp use have a time complexity of linear search. O(n).

Robinkarp use RollingHash.

$H_A = x + y + s + a + s$

$H_B = s + m + a + r + t$

$H(str1) \neq H(str2) = H_A - x + m$ —①

$$hash(str) = \sum_{i=0}^{N-1} str(i) * p^{i+1}$$ p is a prime no.

(-) $N + M + 1 + (N-M) * 1$ $N > M$ (hashA, hashB)

→ N, 1

if p = 5

$H_B = (5^1 \times s + 5^2 \times m + 5^3 \times a + 5^4 \times r + 5^5 \times t) \% K$

$H_A = (5^1 \times x + 5^2 \times y + 5^3 \times s + 5^4 \times a + 5^5 \times s) \% K$

$= \left(\frac{H_A}{5} - x + m \times 5^5\right) \% K$

→ If there are 1000 characters in B then the string will go till 5^{1000} which will give an overflow error.

→ To fix it we implement modulus of k k=1000

$$\therefore (a/b) \% k \neq \frac{a\%k}{b\%k}$$

→ As modulus is not distributive over division so we cannot use the given hashfunction.

→ $h(s) = \sum_{i=0}^{N-1} str(i) * p^i$

→ So, instead of increasing powers of 5 we can decrease power of 5.

$$H_B = (5^5 \times 5 + m \times 5^4 + 9 \times 5^3 + 1 \times 5^2 + t \times 5^1) \% k$$

$$H_A = (a \times 5^5 + y \times 5^4 + s \times 5^3 + 9 \times 5^2 + s \times 5^1) \% k$$

$$= (H_A - x \times 5^5) * 5 + m \times 5^5 \quad \text{— (2)}$$

$$\boxed{hash(str) = \sum_{i=0}^{N-1} str(i) * p^{N-i}}$$

Generalised equation of (2)

$$(H_A - A(1-m) \times p^m) * p + A(i) * p$$

① $k = 1e9+7$, $p_1 = ?$, $p_2 = ?$ T.C → 1,1

② $pwr_1(m+1)$, $pwr_1(0) = 1$ T.C → m,m

$$\forall_{i=1}^{m} pwr_1(m+1) = (pwr_1(i-1) * p_1) \% k$$

$pwr_2(m+1)$, $pwr_2(0) = 1$

$$\forall_{i=1}^{m} pwr_2(m+1) = pwr_2(i-1) * p_2) \% k$$

③ $H_{B_1} = 0 \quad H_{B_2} = 0$

$\forall_{i=0}^{m-1}$ $H_{B_1} = (H_{B_1} + B[i]) * pwr_1(m-i)) \% k \rightarrow m, 1$ (T.C)

$H_{B_2} = (H_{B_2} + B[i]) * pwr_2(m-i)) \% k \rightarrow m, 1$ (T.C)

④ $H_{A_1} = 0 \quad H_{A_2} = 0$

$\forall_{i=0}^{m-1}$ $H_{A_1} = (H_{A_1} + A[i]) * pwr_1(m-i)) \% k$.

⑤ $H_{A_1} = = H_{B_1}$ && $H_{A_2} = = H_{B_2} \rightarrow 1, 1$ (T.C)

⑥ $\forall_{i=m}^{N-1}$ $H_{A_1} = (H_{A_1} - A[i-m] * pwr(m)) * p + A[i]) * p) \% k \rightarrow N-m, 1$

(Go to the next 3rd character
leave the first one)
Take the required string

$H_{A_1} = (H_{A_1} + k) \% k$

repeat step ⑤ N, M, A

→ To avoid collision we are using two power values which are P_1 and P_2

→ O(n), m T.C for Robin Karp

→ Given an array

	0	1	2	3	4	5	6	7	8	9
arr_N	5	3	4	2	8	−6	10	3	12	−10

two indices of the array are given we need to find the sum of the elements between the indices

$Q(i, j) \rightarrow \sum_{k=i}^{j} arr[k]$

$0 \le i \le j \le n$

3, 5 → 4
1, 4 → 11
3, 7 → 19
⋮ ⋮

Solutions 1) Q * N, 1 (Brute force code)

↳ iterating through each query i times

2) N+Q, N ← space required for pharmacy

- By using cummulative sum we can reduce the time complexity

The array will be

0	1	2	3	4	5	6	7	8	9
5	8	7	9	13	11	21	24	36	26

This can also be considered as fibonacci series

Also,

$cs[6] - cs[1] = key$

$cs[j] - cs[i-1]$ if $(i \neq 0)$ else if $(i = 0)$ then $cs[j]$

This formula can also be written as $cs[j] - cs[i] + arr[i]$

$cs[5] - cs[2] + arr[2]$

$11 - 9 + 12$

$= 4$ → (?) ← No of test cases

$cs[0] = arr[0]$; ← No of iterations

$\forall_{i=1}$ $cs[i] = cs[i-1] + arr[i]$

→ For the string of elements

str: a b c d x y x z x y x d e m o n o m c d

x z p z p for odd length palindrome

1) $N^3, 1$

2) $N^2, 1$

3) $N \log_2^N, 1$ → applying Binary Search

```
lo=0, hi= min(i, N-i-1)
while(lo<hi)
{
 mid = lo+ (hi-lo+1)/2;
   if (valid(i-mid, i-mid, str) == T)
     lo=mid;
   else
     hi=mid-1;
   }
   return 2*lo+1;
Valid (str i+mid, i-mid, str)
{
   p1= i-mid;
   p2 = i+mid;
   while(p1 < p2)
   {
     if (a[p1]==a[p2])
     { p1++; p2--; }
       else
         return false
       }
       return true
     }
```

→ abc = cba if it is a palindrome

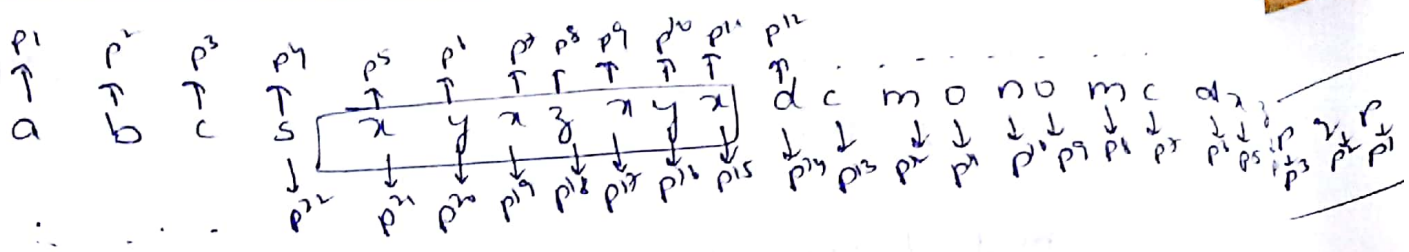
$a*p^1 + b*p^2 + c*p^3 = c*p^1 + b*p^2 + a*p^3$

actually it would have been

abc = abc

$a*p^1 + b*p^2 + a*p^3 = a*p^1 + b*p^2 + a*p^3$

so if both have the same hash value both of them are same. Hence it is a palindrome



$$FH = \sum_{i=0}^{N-1} str(i) * p^{i+1}$$

$$BH = \sum_{i=N-1}^{0} str(i) * p^{N-i}$$

Computing forward hashing:

$$FH(0) = str(0) * p$$

$$FH(i) = \sum_{k=0}^{i} str(k) * p^{k+1}$$

$$= FH(i-1) + str(i) * p^{i+1}$$

Backward hashing:

$$BH(i) = \sum_{k=N-1}^{i} str(k) * p^{N-k}$$

$$= (BH(k+1) + str(i) * p^{N-k}) \% k$$

$$BH(N-1) = str(N-1) * p$$

equating:

$$h_1 = FH(p_2) - FH(p_1 - 1) * p^d$$

$$h_2 = BH(p_1) - BH(p_2 + 1)$$

~~$1) N^2 * N = N^3$~~ If equal then its a paliondrume.

→ steps to solve

1. $k = 1e9 + 7$
2. compute pwr(N+1)
3. compute FH( )
4. compute BH( ) initialisation

⑤ for each centre call Binary search with

lo = 0 , hi = n

valid (str, i-mid, i+mid)

$h_1 = FH(R) - FH(L-1) * P^k$

Week 5 Day 2

→ To find the length of the longest substring

① $N^3, 1$ ② $N^2, 1$ ③ $N \log_2 N, N$ ④ N, N → manacher's algorithm.

→ Consider an array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
A:	a	b	c	x	y	z	z	a	y	z	m	n	s	m	a	r	t	d	a	t	a	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
B:	m	y	z	a	y	z	s	m	a	r	t	x	y	z	a	b	c	d	z	z	a	y	m	n	c

→ Check whether the substring of A i to j is equal to the sub-string of B from k to l.

Q. i, j, k, l : substr(A, i, j) == substr(B, k, l)

i	j	k	l	
3	5	11	13	→ T
A	8	10	14	→ F
12	16	6	10	→ T

Solutions: 1) $Q \times (N), 1$ (or) $Q \times m, 1$ if $N < m$ then N

Brute force code substr(A, i, j) == substr(B, k, l);

2) $N + m + Q, N + m$

Convert the first string into hash value and 2nd string also into hash value and then compare.

$H_A = FH_A(j) - FH_A(i-1)$ $i \neq 0$

$H_B = FH_B(1) - FH_B(k-1)$ $k \neq 0$

if $(i < k)$ then
$H_B = H_B * p^{k-i}$

else
$H_B = H_B * p^{j-k}$

2) $N+m+q, N+m$

Topics discussed so far:

① Bit manipulation

② Recursion

③ Searching

④ Sorting

⑤ Backtracking

⑥ 2-ptr

⑦ prefixsum

→ Construct an array b which will hold all the values of multiplied by all the elements except i

ar_N: 1 5 2 8 7 4 6 8 10 3

br_N: $br(i) = \prod_{k=0, k \neq i}^{N-1} ar(k)$

1) $N^2, 1$ (Brut force code

2) $N, 1$

br(i) = P/arr[i]

P is the product of all the numbers.

arr[i] is the no. which needs to be divided in order to neglect it.

If arr[i] is equals to 0 then we have to handle it separately

Then we will get br(i)

If there are 2 zeros the entire result ~~to~~ array will be 0.

3) N, N

How to handle this in a system where '/' operation is costly

prefix product * suffix product

br(i) = PP(i-1) * SP(i+1)

it doesn't work for index 0 and n-1

br(0) = SP(1)

br(N-1) = PP(N-2)

① $\forall_{i=1}^{N-1}$ PP(0) = arr(0)

PP(i) = PP(i-1) * arr(i)

Space will be 2N but instead of storing it in PP(i) we can directly compute it.

② $\forall_{i=N-2}^{0}$ SP(N-1) = arr(N-1)

SP(i) = SP(i+1) * arr(i)

③ $br(i) = pp(i-1) * sp(i-1)$

$br(0) = sp(1)$

$br(N-1) = pp(N-2)$

directly computing the prefix part instead of storing it

$\forall_{i=1}^{N-2}$ $pp = ar[0]$

$br(i) = pp * sp(i+1)$

$pp = pp * ar(i)$ → update the prefix product

→ Given array

	0	1	2	3	4	5	6	7	8	9	10
ar_N:	0	3	5	5	3	1	1	-1	4	5	5

Replace each element in the array from the given start to end index by adding it to the elements of array.

Finally print the array formed.

1	4	3
2	6	2
8	10	5
5	8	7

By applying this we will get an array as

O/P: 0, 3, 5, 5, 5, 1, 1, 7, 4, 5, 5

1) Q*N, 1 iterate from i to j for every query you will have 'n' iterations. This is a brut force code.

2) Q+N, 1 → (?)

Prefix sum should stop at index 4.

0 ∅ 0 0 0 0 0 so we place negative of that no. at index 5.

0	1	2	3	4	5	6	7	8	9	10
0	∅	0	0	0	∅	0	0	0	0	0
	-3				-3					
0	3	3	5	5	5	1	1	1	4	5	5

put ← ar[i] = n
ar[j+1] = x

x is negative of n.

→ Given multiple ranges check in how many it occurs.

1	4
2	6
8	10
5	8

3 occurs twice
9 occurs in 1 of them.

→ Given

0	1	2	3	4	5	6	7	8	9	10	11	12
arN: 0	0	0	0	0	0	0	0	0	0	0	0	0

∅ | 1 i → ar(i) = 1
2 i → find

2	5	→ 1
1	3	→
1	7	→
2	6	→ 1
2	12	→ 5
1	10	→
2	12	→ 2

∴ All the elements in the given array are 0's.

whenever you find 1 then you have to replace the 0 at the i^{th} index by 1

→ if you find 2 then you need to find the minimum distance between the given index and the index which has 1.

Solution:

1) 1 → i , 2 → ← i →
   - add indexes
   - search

N : O(N, 1)

2) 1 → N, 2 → 1 : O(N, N)

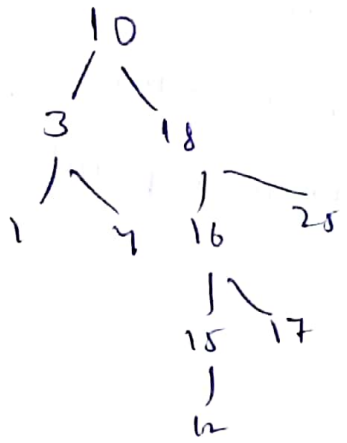
0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0
→	→	→	→	→	→	→	→	→	→	→ add 1's to all 0's
3	2	1	0	0	1	2	3			→ then replace them by 0,1,2,3 on left and right hence it will be the distance

3) 1 → 1, 2 → N ; O(N, N) → store the indexes with N

↓

to reduce it we sort list the entire array elements and find the floor and ceil then calculate the distance and minimum distance.

1) 1 → $\log_2 N$, 2 → $\log_2 N$, N → space
↳ using BBST



It says to insert the elements into the tree it will take $\log_2 N$ and to search it will take $\log_2 N$.

→ Given an array

A_N:	10	8	15	25	31	47	53
B_N:	60	73	20	3	18	22	63
C_N:	13	1	28	34	68	42	58

$$\min_{i,j,k} ((\max(A_i, B_j, C_k) - \min(A_i, B_j, C_k))$$

Choose one element from A_N, one from B_N & one from C_N such that we get min(max-min)

Solutions:

1) N^3, 1 → sort the arrays
2) $N \log_2 N + N$, 1 → iterations to find min.

AN: 8 10 15 25 31 47 53

BN: 3 18 20 22 60 63 73

CN: 1 13 28 34 42 58 68

to reduce the difference between A and B we have to either dec. A or inc. B.

Here we inc. B i.e move B pointer to right side by incrementing it.

Cf → 10027 → C

→ Given 'n' no. of ranges find the length of the overlapping segment.

5

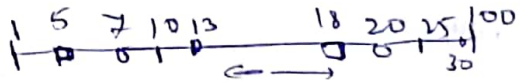
7 20

5 18

10 25

1 100

13 30



Here we have to skip exactly once.

If we skip 13 30 we will get the range as 10 → 18

Solution:

1) N^2, 1 : N × N → finding the min. distance
↳ to neglect one of them

1) $N \log N_1 + N - 1N$, $2N$ → Find the minimum distance
   - $N \log N$ → sort the elements
   - N → exclude one of the element
   - $2N$ → after sorting to store in an array.

→ Given array

arr: 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 0

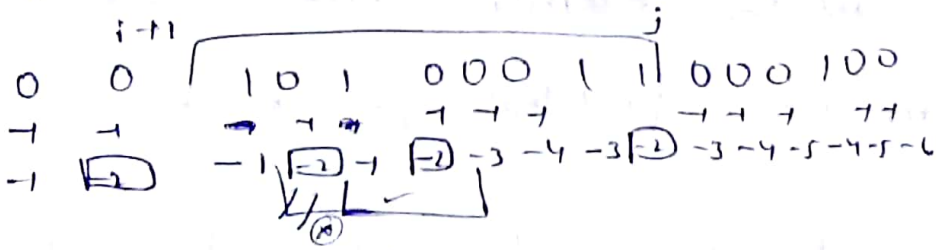
Find the largest sub-array which has equal 0's & 1's

1) $N^2 \times N, 1$

```
i=0 to n
{j= i to n
{ k = i to j
  {
    count 0's;
    count 1's;
     if (0's == 1's)
     ans = max (ans, j-i+1);
  }
}
}
```

2) $N^2, 1$ → carry forward.

3) $N + N^2, N$
   → using two pointer approach take the max when found 2 pointers are same.



4) $N + N, N$
   → using hash map

HM1	HM2
(-1, 0)	(-1, 0)
(-2, 1)	(-2, 1)
(-3, 1)	(-3, 1)

if we use count array T-N1N)

count[T2N+1]

→ In hash map we can make pairs and calculate the difference between elements with same key

(6,15) (-5,12) (-5,14)

14-12 = 2

→ Solve the given matrices

Mat N*m :

-12	-9	-2	5	10
-8	-3	0	8	13
-1	5	10	18	23
3	10	21	27	41
17	23	34	51	93
18	25	38	63	107

found 3

In this all the elements are sorted rowwise and column wise find the given key k.

1) N*M, 1 Brutforce code for row & col. search.

2) NlogM, 1 (or) MlogN, 1

As they are sorted rowwise & col. wise perform binary search rowwise and for all rows it will take NlogM or col wise.

3) $N+M, 1$ (RT, LB)

↓

You can try to find the element from Right Top or left Bottom.

To find (3)

(2) ~~-9~~ -7 point at 10

10 is greater so move left

then point at 5

5 is greater so move left

then point at -2

-2 is smaller so move down reach 3 similarly.

(2) Mat_{N*M}:

-19	-11	-8	-3	-1	5
7	9	12	18	21	23
25	28	31	35	38	47
51	53	59	62	68	73
75	78	79	81	83	89
92	95	99	101	112	119
122	128	133	142	207	523

Rows are sorted last element of n-1 is smaller than 1st element of n find key.

Solutions

1) $N*M, 1$

2) $N\log_2^M, 1$ || $M\log_2^N, 1$

3) $N+M$ (RT, LB)

4) $N+\log_1^M, N$ → apply BS

↳ take limits for each & every row you need to take.

⑤ $\log_2 N + \log_2 M$, 1

Find the floor at left & ceil at right & apply binary search.

⑥ $\log_2 NM$, 1

lo=0 hi=4,

mid=20

20th element has to be represented in the form of 2d matrix

mat[3][2]

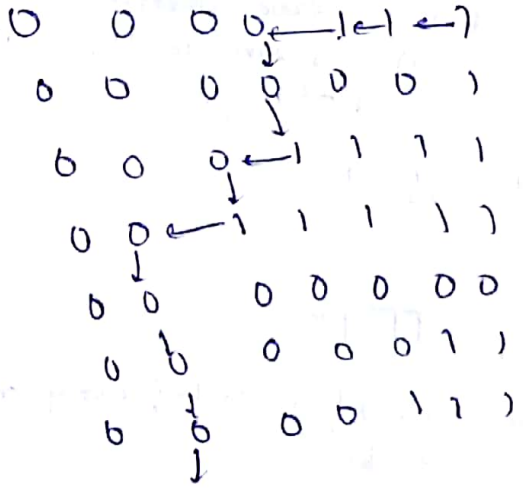
20 → 20/6 → 3 → mid/m

→ 20%6 → 2 → mid%m

25 → 25/6 → 4 →

→ 25%6 → 1 →

3) Find the max. no. of 1's in the given sorted matrix.



1) N×M, 1

2) $N \log M$ Applying B.S ~~Count~~ ~~to~~ stop when 1 appears count M-1 you will get no of 1's the max. no should be considered.

~~3) $N \log \frac{M}{2}, 1$~~ if (1)
lo=0, hi=mid;
else
lo = mid

4) $N+M, 1$ Every time we move we skip one row or one column we apply it at RT & RB.

puzzle: 0 0 0 3 4

such that 0 should appear 3 times.

Week-6
Day-1

Prime Numbers & Game theory

Given an array rotate it by d times.

ar_N:	1	3	5	8	-3	10	2	15	-1	-5
d=1:	-5	1	3	5	8	-3	10	2	15	-1
d=3:	15	-1	-5	1	3	5	8	-3	10	2

Here we can see that $d = d \% N$.

if d=43 then 43%10 = 3

Here we can observe that the left and right sides have been swapped.

Left side is from (0 to n-1-d) and the right side contains (n-d to n-1)

For d=1. (∵ d indicates no. of rotations/swaps)

take a temporary variable t.

t = arr[N-1] // copy the last digit to t

$\forall_{i=N-1}^{1}$ arr[i] = arr[i-1] // shift the position of each element towards right.

arr[0] = t // copy the temporary variable value to first position (index) of array.

Solutions:

1) d×N, 1

2) N, N → space required
   ↳ swapping N elements

take a temporary array ~~by taking p1~~ pointer at the send ith index to i+d

d = d % N

$\forall_{i=0}^{N-1}$ B[i+d] = arr[i]

B[i] = arr[N-1]

3) N, 1

Reverse the array by taking p1 pointer at starting and p2 pointer at ending and swap.

① Reverse (0, N-1) // reverse the entire array

~~Reverse (N-d, N-1)~~

② Rev(0, d-1) // reverse from start to d-1

③ Reverse (d, N-1)

work(0, N-d-1)

② R(N-d, N-1)

③ R(0, N-1)

→ Given an array of elements, search a key in sorted then rotated array

ar_N:	-8	-5	1	3	8	9	12	15	18	19	25	28	30
d=5:	18	19	25	28	30	-8	-5	1	3	8	9	12	15

Solution:

1) converge to point A: if(mid < L) : lo = m → if(ar[m] > ar[0])

else : hi = m-1;

2) If mid lands at point B : if(ar[m] > ar[0]) (mid < L): lo = m+1

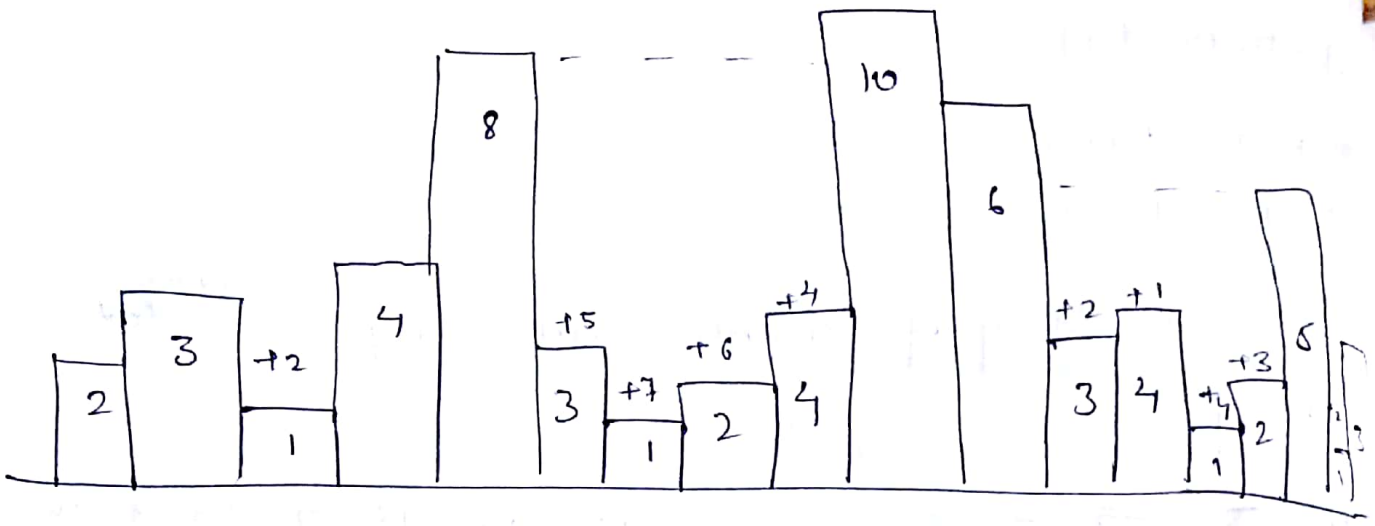
else hi = m

mid < Left hand side → k ∈ [ar[lo], ar[m])

→ else

else →

→ Given that there are buildings with different heights and a constant width of 1m calculate the amount of water which can be poured in between them.



Consider it as an array N.

ar_N: 2,3,1,4,8,3,1,2,4,10,6,3,4,1,2,5,1,3

manually to calculate the amount of water it is (36).

$$Water\_on(i) = \min(Lm(i), Rm(i)) - ar[i]$$

$$Lm(0) = ar[0]$$

$$\forall_{i=1}^{N-1} Lm(i) = \max(Lm(i-1), ar[i])$$

$$Rm(N-1) = ar[N-1]$$

$$\forall_{i=N-2}^{0} Rm(i) = \max(Rm(i+1), ar[i])$$

- Given its 3-d version calculate the amount of water which can be put

7	8	15	18	9	6
20	4	1	3	4	12
17	3	12	6	18	23
91	7	3	4	29	80
93	25	16	12	33	20

SPOJ Water → Heaps + Graphs

→ given a no. 'n' check if it is prime

```
bool isPrime(int n)
{
    c=0;
    for(i=2 ; i<=√n; i++)
    {
        if(n%i==0)
            return false;
        else
            return true;
    }
}
```

Complexity → $\sqrt{n}$, 1

- To find the no of prime no's ~~between 2 given nos~~ upto a given no.

```
int primes(int n)
{
    c=0;
    for(i=2 to n)
    {
        if(isPrime(i))
            c++;
    }
    return c;
}
```

① T.C → $2^{1/2} + 3^{1/2} + 4^{1/2} + 5^{1/2} + \ldots N^{1/2}$

→ To find the ~~no~~. prime no's upto a given number.

(2) TC → $N(1/2 + 1/3 + 1/5 + 1/7 \ldots 1/n)$ $n < n$

~~0~~ ~~1~~ 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~

11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~

~~22~~ 23 ~~24~~ 25 ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31

~~32~~ ~~33~~ ~~34~~ 35 ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~

41 ~~42~~ 43 ~~44~~ ~~45~~ ~~46~~ 47 ~~48~~ ~~49~~ ~~50~~

Find the first no such that they do not exist in the list of prime numbers. iterate through its multiples making it false

2, 4, 6, 8. . . .
3, 6, 9, 12. . .

→ Iterate through all and mark their multiples as F:

```
int prime(int n)
{ bool a[N+1]={T}; // take a boolean array and mark it as true
  for(i=0 to √n)
  { a[i] = 1;
    a[0] = F;
    a[1] = F;          // 0 and 1 are made false.
    if(a[i] == T)
    { cnt;
    }
    for(j=i to n)
        a[j] = F
    }.retc;
}
```

i	j	total
2	4,6,8	N/2
3	6,9,12	N/3
4	———	0
5	———	N/5

$i < \sqrt{n}$
$j = j + 2$

time complexity will be

$N(1/2 + 1/3 + 1/4 + \dots + 1/n)$ $\quad n < N$

If the limits are too high

such that

find the no of prime nos between

$1 \le N \le M \le 10^{10}$ $\quad [5, 12]$

$0 \le M - N \le 10^6$ $\quad [107, 2514]$

5, 7, 11

(3)

$(P - N\%P)$

Go through algorithms

SIEVE OF ERATOSTHANESE

$[N, M] \rightarrow f(\sqrt{M}) \leftarrow$ list

$1 \le N \le 10^8$

$1 \le T \le 10^2$

$O(T \times \sqrt{M}) = 10^3 \times 10^9 = 10^{12}$

$O(T \times M) = 10^3 \times 10^5 = 10^8$

```
bool p[M-N+1];
{
  for (p in list){
    for (int j = (p - N%p) %p)
        j <= M-N ; j = j+p
        p[j] = F;
  }
```

Miller Rabin Primality Testing

Game theory:

Consider there are two players A and B

Each can pick either 1, 2, or 3 coins

whoever picks the last coin out of the given 'n' coins wins the game.

Optimal vs optimal

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	W	W	W	L	W	W	W	L	W	W	W	L	W	W

∀ moves leads to W position → L

∃ moves leads to L position → W

if (N%4 == 0)

Who so ever starts looses the game.

Suppose there are 17 coins

17 —(I pick 1)→ 16 —(He picks 3)→ 13 —(I pick 1)→ 12 —(He picks 2)→ 10 —(I pick 2)→ 8 —(He picks 3)→ 5 —(I pick 1)→ 4 —(He picks 1 or 2 or 3)→ I win

So Here I tried to push the player to multiples of 4 coins so that at last he looses.

Now if there are two players (optimal vs greedy)

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
O:	W	W	W	L	W	W	W	W	W	W	W	W	L	L
G:	W	W	W	L	L	L	W	L	L	L	L	L	L	L

Here the greedy player always wishes to pick 3.
Optimal player picks the coins in such a way that he can defeat the greedy (L).

Greedy vs Greedy

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G:	W	W	W	L	L	L	W	W	W	L	L	L	W	W

```
if (N % 6 == 1,2,3): W
    else : L
```

Moves = {1, 2, 4, 8, 16, 32, 64 ...}

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	W	W	L	W	W	W	W	W	.	.	.				
							16	17	18	19	20				

→ Consider that there are N files each file has certain no. of coins.
The player who picks the last coin will win the game.

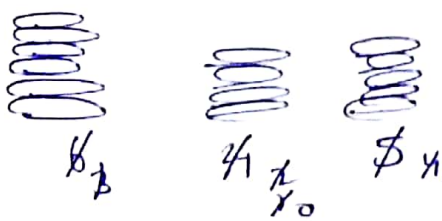


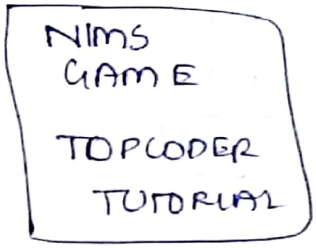
Figure out the logic to win the game.

S. No.	Date	Title	Page No.
	Day 1		
want.	Week 7	Meet In the Middle Stacks Queues	
	Day 2	Stacks Queues Linked List (Basics)	
Week 8	Day 1	Advanced Linked Lists	
	Day 2	Basic Trees. & LL's.	
Week 9	Day 1	Advanced Trees	
	Day 2	Trees	
Week 10	Day 1	TRIES, MinHeap.	
	Day 2	MinHeap, MaxHeap Dynamic prog	
~~Week 11~~	Day 1		
	~~Day 2~~		

Week 7
Day-1
22/9/2018

Meet in middle Stacks Queues
Linked List

→ There are n files, each file has (n coins) n moves. The player who picks the last coin wins the game.



6, 4, 5 can also be considered as no. of possible moves.

→ If the no. of coins are ~~eq~~even or same then the one starting the game will loose.

6 → 1 1 0 → 0 0 1
5 → 1 0 1 → 1 0 1
4 → 1 0 0 → 1 0 0
1 1 1 → 0 0 0

So, we can see that 5 coins are to be removed from the file containing 6 coins

If XOR = 0 then the player at that point will loose
If XOR ≠ 0 then the player at point will win.

~~If~~ move, the XOR ≠ 0
∃ moves makes XOR = 0

This can be related with the previous one.

∀ moves leads to W → L

∃ moves leads to L → W

HO

110	001	001
101	101	011
100	100	010
111	000	000

You have to perform the XOR between the moves

Meet In the Middle

→ Sum of subsets of the given array should be equal to the given no. k.

ar$_N$: −4, 5, 3, 6, 10, −7

k = 12, 7

→ this is not divide and conquer as in that we divide it and then again merge it. Here no merging.

1) $2^N \times N$, 1

2) 2^N, 1

↳ using this

Divide the elements into two equal halves

	0	1	2	0	1	2
	−4	5	3	6	10	−7
0	0	0	0	0	0	0 → 0
−4	0	0	1	0	0	1 → 6
5	0	1	0	0	1	0 → 10
1	0	1	1	0	1	1 → 16
3	1	0	0	1	0	0 → −7
−6	1	0	1	1	0	1 → 4
8	1	1	0	1	1	0 → 3
4	1	1	1	1	1	1 → 9

Solution:

1) $2^{N/2} \times \frac{N}{2} + 2^{N/2} \times \frac{N}{2}$, $2^{N/2} + 2^{N/2}$

(possible combinations; taking out ($\because n = N/2$); a; b)

This is a brut force solution where $2^{N/2}$ represents all the possible combinations and $N/2$ represents the half part of it in this 4,5,7.

2.) a) B.F → $2^{N/2} \times 2^{N/2}$

b) $2^{N/2} \log_2 2^{N/2} \Rightarrow N/2 * 2^{N/2} + N/2 * 2^{N/2}$, 1 (→ bsearch; → sort)

c) $N * 2^{N/2} + 2(2^{N/2})$, 1 (→ sort; → 2 pointer approach)

d) $2^{N/2} + 2^{N/2}$, $2^{N/2}$ (→ Hashing; → searching)

→ 2^n are the possible number of combinations.

Here the no. of elements are $2^{N/2}$.

1) $2^{N/2} \times N/2 + 2^{N/2} \times N/2$, $2^{N/2} + 2^{N/2}$

→ We divide the problem into two part, & then somehow try to get the result is sth. basic concept of meet in the middle.

→ Try to reduce the value of 2^N to $2^{N/2}$ which decreases the time complexity to a lot extent.

Generating Quadraples

$a \wedge b \wedge c \wedge d = 0$

↓

NY

$a^n b = c^n d$

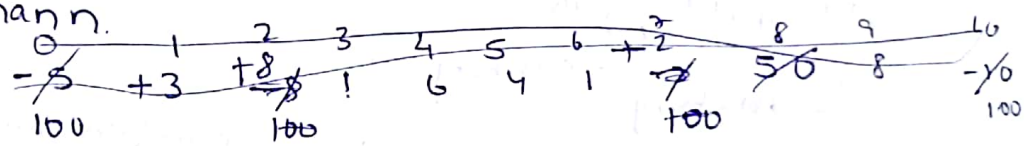
$(N^2 = N^2)$

$a_{rN}: a$
$b_{rN}: b$
$c_{rN}: c$
$d_{rN}: d$

→ Given an array find the first missing positive integer.

1) a_{rN} −5 , +3 , +8 , 1 , 6, 4, 1, +2 , 50, 8 , −10

Replace negative numbers with a no. greater than n.



0 1 2 3 4 5 6 7 8 9 10
−8 3 8 1 6 4 1 2 50 8 −10
100 100 100

1) $N^2, 1$ ∴ Ans ∈ [1, N+1] Applying linear we will get it.

2) $N \log_2 N \oplus N$ ⟶ Sort and apply Linear Search

3) $\log_2 N$ ⟶ Sort and apply Binary Search

4) ~~⟶~~ insert
$N+N$, N

Insert elements in unordered hash set and search for the required element

5) bool [n+[N+2]

N+N,N

```
int f(arr[],n)
 {
   for(i=1; i<=n; i++)
    {
       if( abs(x)-1>0)
         -x = abs(x)
       else
         return x;
    }
 }
```

→Given an array find the frequency of numbers in the array

ar_N: 1 5 3 1 2 5 7

arr[i] ∈ [1,N]

1 → 2
2 → 1
3 → 1
5 → 2
7 → 1

0	1	2	3	4	5	6
	↓			↓		
1	5	3	1	2	5	7
+10	+10			+10		
11	+5 +10 25			12 +10 22		

find the frequency with complexity N,1

→Put two pointers at 1st location then add 10 it will be 15 take %10=5 so 5-1=4 then move the pointer to 4th location and again add 10 It will be 12

take 12 %10= 2 2-1=1 so move to the first index add 10 it will be 25

take 25%10= 5 5-1=4 so move to 4th index add 10 it will be 22 22%10= 2 2-1=1

move to next 3 add 10 it will be 13 13%10=3 3-1=2 add 10

7 % 10 = 7 7-1=6 add 10 at 6th location.

You will get 7+10=17

→ Then divide all of them by 10.

we will get the frequencies.

STACKS

LIFO

- push(x)
- pop()
- top() / peek()
- size()

QUEUES

FIFO

- enqueue(x) // add
- dequeue()
- rear() / front()
- ~~front~~size()

These are the operations that can be performed on stacks and queues.

→ Stacks and queues can be implemented using Arrays and linked list.

t = -1

Stack: arr[++t] = x; //insert

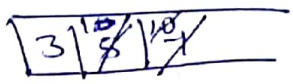
t--; //pop

return arr[t]; //return top element

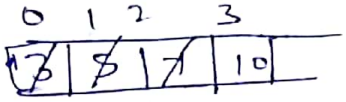
size → t+1 //size.

∴ upon putting one element t will be 0 so size will be t+1

3, 5, -1 ↑ ↑ 10 ↑ ↑



Queue: 3, 5, -1, ↑ 10 ↑ ↑



put 3, 5, -1 to pop remove the first element inserted and again to put put the elements at the end.

```
f = -1   r = -1

arr[++r] = x ;   // insert

  f++;           // remove

arr[f++] ;       // front

  n-1  ;         // size
```

Stack

→ Now t = -1

push( ) → 7 element
t will be 6.

pop( ) → 3
t = 3

push( ) → 5
t = 8

Queue:

N = 10 f = -1 r = -1

enque → 7
f = -1 r = 6

deq → 3
f = 2 r = 6

enq → 5

∴ r + 5 = 11. So we cannot put 5 elements

as the size increases beyond n so we make the queue a circular one.

→ In order to make the queue circular we have to arrange (insert) the elements in such a way that when it fills upto n it shall again start filling from the start onwards.

arr[(r+1)%N] = x ; //insert element into the queue.

arr[(f+1)%N] ; // front to get an element.

r-f ; // size

arr[r%N] ; // rear

But these will not work in all cases.

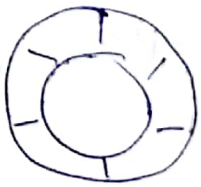
r = (r+1)%N

arr[r] = x; //insert

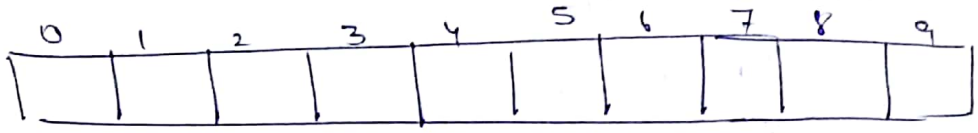
f = (f+1)%N

arr[f] = // front

(r-f+N)%N // → size



Real time practical example:- If I consider a server and run it then it will not stop at any point till we stop it.



N=10	enq(7)	deq(5)	enq(5)
r=-1	f=4	f=4	f=4
f=4	r=6	r=6	r=1

r-f = -3

so it is not correct size

r-f+N = -3+10 = 7

7%10 = 7

∴ size() = (r-f+N)%N

→ So while adding a process we must do it r+1 so that it is at 0 and then %N.

r = (r+1)%N

So in this way we make it circular for insertion and also to calculate the size.

N=10	Enq(7)	Deq(7)
f=4	f=4	f=4
r=4	r=6	r=6
	s=7	s=0

So Here every thing is true.

enq(N)

f = -1 f = -1

r = -1 r = N-1

(r-1+N)%N

(N-1+1+N)%N

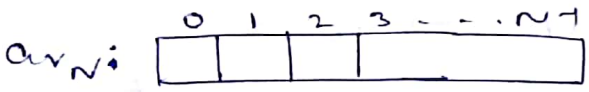
2N%N = 0

size = 0.

→ So, here we are getting size as 0

But we know that the size is N.

→ So if r is pointing to N-1 then we say it is full.



S1.push(x1)

S1.push(x2)

S2.push(x3)

S1.top() → x2

S1.pop() → x2

S2.top() → x3

S2.pop() → x3

S1.size → 1

S1.top → x1

→

	top	push(x)	pop()	peek()	size
S1:	$t_1 = -1$	$arr[++t_1] = x$	t_1--	$arr[t_1]$	t_1+1
S2:	$t_2 = N$	$arr[--t_2] = x$	t_2++	$arr[t_2]$	$N-t_2$

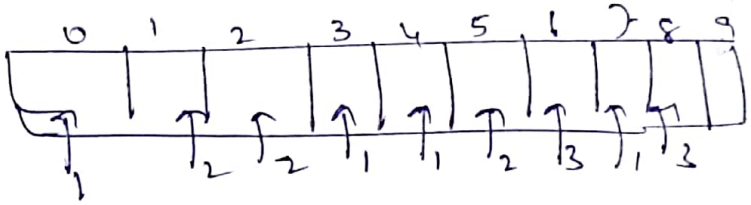
→ Implement m stacks in a single array of size N. such that $m << N$.

$m = 5 \quad N = 1000$

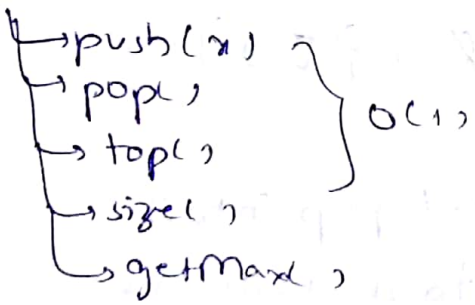
$5 << 1000$

Here m stacks means as in a double ended queue th we can add elements from both sides in m stacks we must be able to insert and retrieve elements at any point.

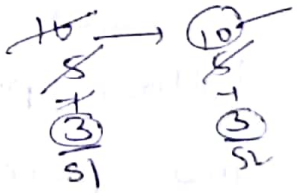
→ For this you can increase the space of the entire array as $N, 2N, 3N \ldots < m \times N$.



Stacks



3, -1, 5, 10 gmc(), ↑, gmc(), ↑↑, gmc()



→ Collecting mangoes problem.

→ In this we are inserting the elements poping them and finding the max when required.

int.

Validation of braces

```
int main()
{
int ar[] = {1,5,3};
    for(){
        if(){
        ar[]= —
         }
        else{
          }
    }
}
```

Consider the braces as a string.

str: ( ) { ( ) { } ( ) , { ( ) , { ( ) } }

↑

→ braces follow push and pop into it.

→ The arrow pointing to it will show an invalid error.

→ If a brace is pushed into stack it should be poped or else it will give an error.

→ {([])}{[]}

(1) ⊗ → invalid

→ [should be followed by only] immediately in order to pop it or else it will give an error.

T.C → N, N → a, insertion in stack.
↳ Elements are to be iterated N time

Week-7

Day-2

23/9/2018

Check the no. of strings which are valid.

① ((()())()

② ((()))(

③ (()()(

Soln) N, N → space to store them
↳ N iterations are to be done to check their validity.

```
1 2 1 0 1 0
( ( ) ) ) (
```

(→ c++

) → c--

if(c == 0) return true

if(c < 0) return false

→ So, if c reaches a negative no then return false
if c == 0 then it's true (valid)

Valid substrings

Count the no. of valid substrings

① (( )( ))( ) → 4

②
(( ))(

③
(( ))( )

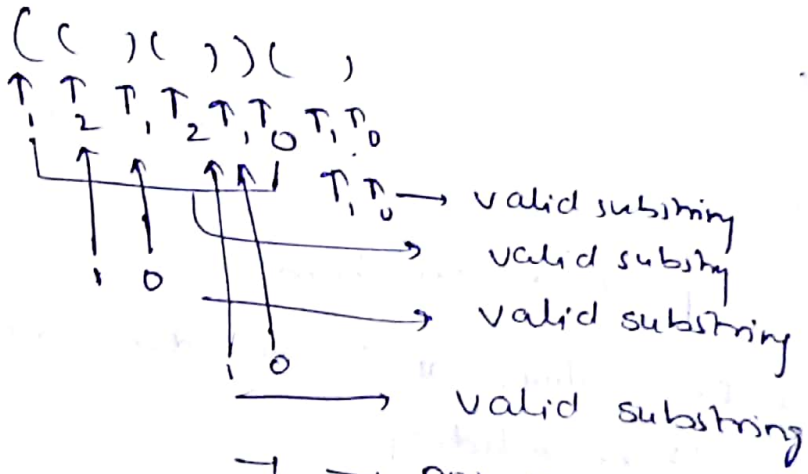
Soln

1) $N^2 \times N$,,

→ check the substring is valid or not

→ As it is performed for each of them so it has to be performed for N^2.

2) N^2,,

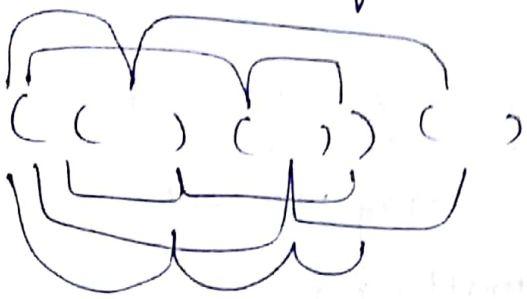


→ → not a valid substring.

→ If a negative no. is encountered then we can say that it is not a valid substring.

→ Here for every brace we check the count is 0 or not.

→ For the given set of braces count the valid subsequences.



Soln) 1) $2^n \times N$, 1 → Generate all the subsequences
code is in page no. 27 book-1

→ For all the p braces you have to traverse once so in.

2)

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

idx=0
char[] =
a[idx] = b[j]

We only need subsequences with positive sum.

→ Consider a string given as input.

i/p:- Data structures and algorithms

o/p:- algorithms and Dat structures data.

Reverse the given sentence.

1) N, N → split the string into array & print from end.

2) $N, 1$
→ two pointer technique, you will get the whole reverse sentence.

Like: ~~s~~ m ~~d~~ ~~x~~ t h ~~x~~ structures ...

Given an array of size N. For every element find the first smaller element on the right side.

arr_N:	5	10	8	12	18	9	11	6	13
brr_N:	7	8	6	9	9	6	5	-1	7

Sol) ans → traverse through the array you will get the next smaller no.

1) $N^2, 1$

2) N, N

→ From left put 5 into the stack then put 10 into the stack the put 8 into the stack as 8 is smaller than 10 so put it into the same index of another array.

→ Then remove 8 put 12 then put 18 as 18 > 12 then put 9 so now we have found an element less than 18. it is even less than 12 so put it in the index of 12 and 18 in another array.

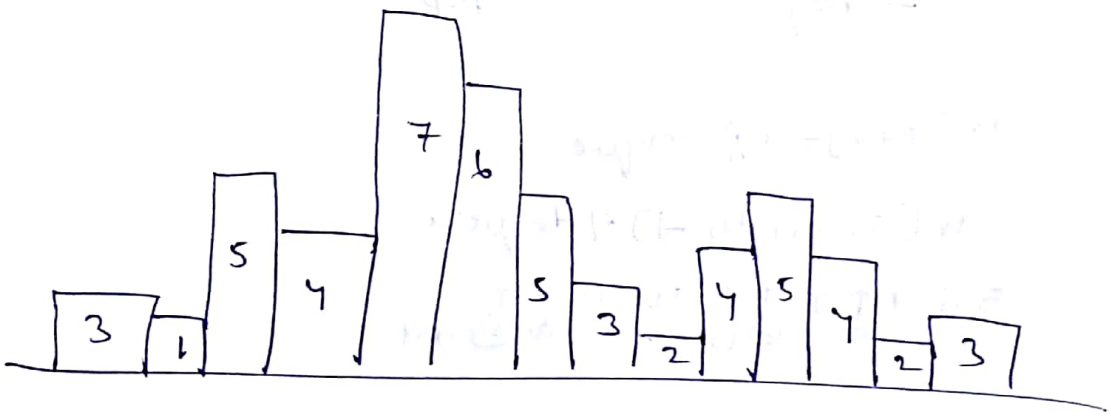
→ The put 11 on as 11 > 9 then put 6 as 6 is smaller than 9 and 11 put it in the index of 9 and 11.

→ Then put 13

→ So now the array arr_N will be having 5, 6, 13 put -1 in their respective indices.

→ If we traverse from right then put
push 13 now ans of 13 is -1
then no no. will have ans as 13 as 6 is less than that.
ans of 13 will be -1 and then push 6 its ans will
also be -1 then push 11 as top of the
stack is 6 so 6 will be the ans
push 9 and ans of 9 will be 6.
push 9 and go to 18 then ans is 9
as it is top of the stack then push 12 ans is 9 and so on.

→ Given a set of buildings width of every building is 1unit. Calculate the maximum area covered by the rectangle.



1) N^2, 1 Brutforce code.
2) Consider N,N two pointers p1 and p2 from a particular mid pointer go towards left and right.
→ The point at which they find

the value of length of the rectangle is less than the given value.

→ Then to get the no. of rectangles we take $P_2 - P_1 - 1$ and multiply it with actual value.

→ Then take the max. of the ans.

$$\max(R(i) - L(i) - 1) * arr[i]$$

→ So as we are using two pointers the complexity will be N, N

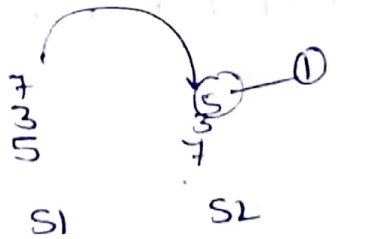
→ Implement Queue using Stacks

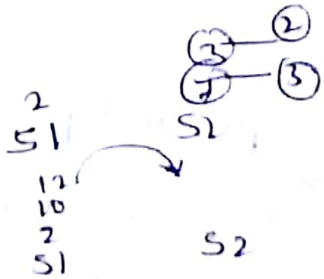
(FIFO)	(LIFO)
→ Enq	→ push
→ Deq	→ pop

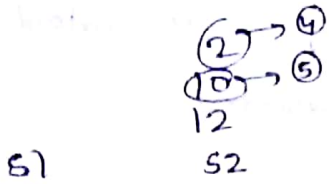
Stack: arr[++r] = x // Enque

arr[arr.length - 1] // dequeue

5, 3, 7 ↑ 2 ↑ ↑ 10 12 ↑ ↑ → pop
① ② ③ ④ ⑤







Enqueue: push the elements into the stack S1

Deque: pop the elements from stack S2.

If there are no elements in S2 then pop push all the elements from S1 and push it to S2.

→ So here in-order to push 100 (enque) element

100 Enque → 1
1 Deque → 100
1E → 1
1D → 100

the complexity is 1.

→ To shift the element the complexity is 100.

→ So here we follow amortized analysis where the cost of shifting the elements is neglected.

O(1) ← Enq O(1)
O(1) ← Deq O(1)

→ Given an array of elements find the max element for a given size k.

arr: 3 12 5 10 8 6 3 1 5 7

k = 4

So) → (N - k + 1) * k, selected elements
Brute force each traversing no. of times

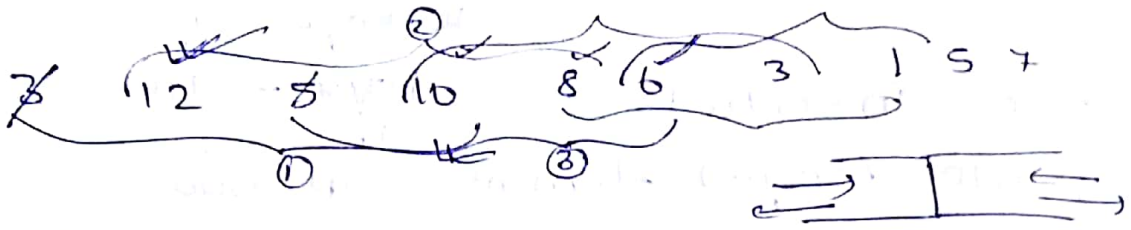
2) Insert the elements into hash map in sorted order for every step moving forward remove the first element and find the max.

$k \log_2 k + N-k(3\log_2 k)$, k

↳ Inserting and sorting them into hash map

→ insert new element
→ delete
→ print max element

3)



N, K we use Double ended Queue Here

Here consider the 1st 4 elements marked as 0

As 3 is smaller than 12 we need not insert 3 and the max. value of the window will be 12.

In the next window as 12 and 10 both are there we need not take 5 Here we have to take 10 because 10 may be the max. value for any other window.

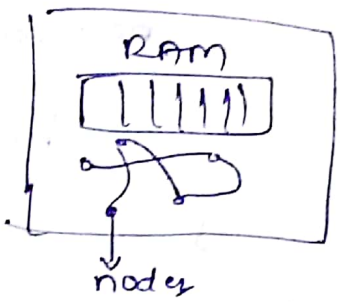
So we print the max as 12.

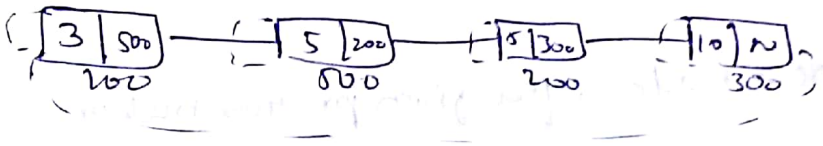
→ Then removing 10 taking 3, if we calculate the maximum then it will be 8 & so on

Linked List (Basics)

→ Four types of Linked list

① Single Linked list
② Double Linked list
③ Circular Linked list
④ Double circular Linked list





→ Perform the operations using single linked list such as insert.

```
/* Java program to illustrate all insertion
methods into a single linked list */

class LinkedList
{
    Node head; // head of the list

    /* Node class */
    class Node
    {
        int data;
        Node next;
        // Constructor to create a new node
        Node(int d)
        { data = d;
          next = null;
        }
    }
```

/* Insert a new node in front of the list

```
public void push(int new_data)
{
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
}
```

/* Insert a new node after given previous node or position.

```
public void insertAfter(Node prev_node, int new_data)
{
    /* 1. check if the given node is null */
    if (prev_node == null)
    {
        System.out.println("Given node cannot be null");
        return;
    }
    /* 2 & 3 : Allocate the node and put in the data */
    Node new_node = new Node(new_data);
    /* 4. Make next of the new node as
       next of the prev node */
    new_node.next = prev_node.next;
```

```
/* 5 Make next of the prev_node as new_node */
        prev_node.next = new_node;
    }

public void append(int new_data)
{
    /* 1. allocate the node
       2. put in the data
       3. set next as null */
    Node new_node = new Node(new_data);
    /* 4. if linked list is empty then make the new
          node as head */
    if (head == null)
    {
        head = new Node (new_data);
        return;
    }
    /* 4. This new node is going to be the last node
       so make next of it null */
    new_node.next = null;
    /* 5. else traverse till the last node */
    Node last = head;
    while (last.next != null)
        last = last.next;
```

```
/* 6. change the next of last node*/
        last.next = new_node;
        return;
    }

/* This function prints the contents of the
   linked list starting from the given node*/
    public void printList()
    {
        Node tnode = head;
        while (tnode != null)
        {
            System.out.print(tnode.data+" ");
            tnode = tnode.next;
        }
    }

public static void main(String args[])
{
    LinkedList llist = new LinkedList();
    llist.append(6);
```

```
llist.push(7);
llist.push(1);
llist.push(2);
llist.append(4);
llist.insertAfter(llist.head.next,8)
System.out.println("\nCreated Linked list is:");
    llist.printList();
  }
}
```

Perform the following DS operations in linked list

① void print(Node head): 1 → 7 → 3 → 6 → 8

② int size(Node head);

③ void printR(Node head)

④ Node delete(Node head, int pos)

⑤ Node insert(Node head, int pos, int x)

⑥ Node insert(Node head, int x)

①
```
void print(Node head) /* print the elements in
                         linked list */
  {
    while(head!=null)
    {
      print(head.data)
      h=h.next;
    }
  }
```

② int size(Node head) /* size of the linked list */

```
{   c=0;
  while (h!=null)
     {
       c++;
      h=h.next;
      }
    S.O.P(c);
}
```

③ void printR(Node head) /*print in reverse order*/

```
{
    if (h==null)
       return;
     printR(h.next); /*recursive call to
                       print from rev. order*/
       print(h.data);
}
```

④ Node delete(Node head, int pos) // delete a node at a particular position

```
{
   if ( pos<1 || pos>size() )
        return h;
        fh=h;
       if (pos==1)
          return h.next;
       while (pos-2!=0)
          h=h.next;
          pos--;
      h.next=h.next.next;
    return fh;
}
```

⑤ void
~~Node~~ insert (Node head, int pos, int x)

{

if (pos < 1 || pos > size())
return h;
th = h;
if (pos == 1)
return h.next;
while (pos-2 != 0)
{ h = h.next;
pos--;
}
~~new node~~
~~h.next = x~~
if new_node.next = null;
Node ~~new node~~ = new Node(x);
if (h == null)
{ return h;
~~cout << "no node";~~
}
else
{
~~new node~~ = h;
new_node
int i = 1;
while (i < pos)
{ new_node = new_node.next;
i++;
}
Node new_node1 = new Node(x);
new_node1 = new_node.next;
new_node.next = new_node1;
}
return h;
}

(6) Linked list code to insert an element at the start and at the end.

```
// Inserting an element at the start
void push( int x)
{
    Node new_node = new Node (x);
    new_node.next = head;
    head = new_node;
}

// inserting an element at the end

void insertAtEnd( int x)
{
    if (head == null)
    {
    head = new_node(x);
        return;
    }
    new_node . next = null;
    Node last = head ;
    while ( last . next != null)
    {
        last = last . next;
    last . next = new_node.
}
```

(7) Node deleteAll(Node h, int x) 1→1→1→7→8→8→10→12→12→12

(8) void remDup(Node h) 1→7→8→10→12

(9) void unique(Node h) ⟶ 7→10

(10) Node reverse(Node h) 12→10→8→7→1.

(7) Node deleteAll(Node h, int x) // to delete all the elements in a ll.

```
{
  h=null;
}
```

(8) void remDuplicates(Node h)

```
{
  Node current=h;
  Node next_nxt;
    if(h==null)
     return;
    else while(current.next!=null)
      {
        {
    if(current.data == current.next.data)
    {
       next_nxt= current.next.next;
          current.next= null;
          current.next=next_nxt;
      }
     else {
       current= current.next;
      }
    }
}
```

Week-8
Day-1
29/9/20

(6)

(10) Node reverse(Node node)

```
{
    Node prev = null;
    Node current = node;
    Node next = null;
    while(current != null)
    {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    node = prev;
    return node;
}
```

(7)

(9) void unique(Node h)

```
{
```

Week-8
Day-1
29/9/2018

Advanced Linked lists Basic trees.

⑥ /*Write a Lcode to ~~delete~~ insert an element x in the linked list*/

```
Node insert(Node h, int x)
{
    Node n= new Node(x);
    Node th; th=h;
    if(h==null || x<h.data)
    {
        n.next=h;
        return n;
    }
    while(h.next!=null && h.next.data<x)
    {
        h=h.next;
    }
    n.next=h.next;
    h.next=n;
    return th;
}
```

1→1→1→2→3→5→5→7

⑦ Program to delete an element x.

```
Node deleteAll(Node h, int x)
{
// I/P:- 1→1→1→5→3→1→1→8→8→1→
// O/P:- 5→3→8→
    if(h==null)
        return h;
    th=h;
    while(h.next!=null)
    {
        if(h.next.data==x)
            h.next=h.next.next;
        else
            h=h.next;
    }
```

```
if(th.data == x)
    th = th.next;
}
```

⑧ Code to remove duplicates from a linked list.

```
void removeDup(Node h, int x)
{
// ip:- 1→1→1→2→5→5→8→10→10→10
// o/p:- 1→2→5→8→10

if(h == null)
    return h;
    th=h;
while(h.next != null)
{
  if (h.data == h.next.data)
    {
      h.next = h.next.next;
    }
  else
    {
      h=h.next;
    }
}
return th;
}
```

⑨ Code to ~~print~~ print only unique elements in a single linked list

s/p: 1→1→1→2→5→5→8→10→10→10
o/p: 2→8→10

```
Node unique(Node h)
{
    Node d = h;
```

```
while(h.next!=null && h!=null)
{
    if(h.data==h.next.data)
    {
      while(h.next!=null && h.data==null)
                h.d==h.n.d)
            h=h.next;
        }
        else
        {
        d.next=h;
          d=h;
         h=h.next;
    }
}
```

10) Node rev(Node h) Reverse a linked list.

```
{
// i/p:- 1→5→3→10→7
// o/p:- 7→10→3→5→1

    p=null;
    while(h!=null)
    {
      t=h.next;
      h.next=p;
        p=h;
        h=t;
    }
  return p;
}
```

Iterab

Recursive code:

```
Node rev(Node h)
{
  if(h.next == null || h == null)
      return h;
  h.next.next = h;
      h.next = null;
    return h;
}
```

Another way of writing the code:

```
rev(H (p,h)
{
  t = h.next;
  h.next = p
    rev(h,t);
    }
```

→ Given two linked list merge both the linked list into and return head.

h1: $-5 \to -1 \to 3 \to 10 \to 10 \to 15 \to$ }

h2: $-2 \to 4 \to 6 \to 20 \to 22 \to$ }

o/p: h
$-5 \to -2 \to -1 \to 3 \to 4 \ldots\ldots$

```
Node merge (Node h1, Node h2)
    {
     Node  d = new  Node ( );
             td=d;
      while ( h1.next!=null && h2.next!=null)
      {
        if (h1.data < h2.data)
        {
          d.next=h1;
           h1=h1.next;
           d = d.next;
        }
        else
         {
           d.next = h2;
            h2=h2.next;
             d = d.next;
         }
      }
       if (h1!=null)
        d.next=h1;
        else
         d.next= h2;

         return td.next;

    }
```

Sort the given Single LinkedList:

→ 5 → 2 → 3

→ Bubble Sort

→ Selection Sort

→ Insertion sort

→ Implementing Bubble sort using linked list.

```
BubbleSort (Node h)
 {
    if (size > 1)
    {
       for(int i=0; i<size; i++)
       {
          Node currentNode = head;
          Node next = head.nextNode;
         for(int j=0; j<size-1; j++)
          {
             if (currentNode.data > next.data)
             {
                Node temp: currentNode;
                currentNode = next;
                 next = temp;
               }
              currentNode = next;
              next = next.nextNode;
            }
         }
    }
}
```

→ Implementing selection sort using single linked list

```
public IntNode nodeSort(IntNode head)
   {
     IntNode holderHead = new IntNode(-1,n
```

```
IntNode cursor;
int currentMax=-1;
int count=0;
While ( count< head.listLength (head))
 {
    IntNode tempHead =new IntNode (-1,null);
    IntNode cursor;
     int currentMax=-1;
     int count=0;
     While (count< head.listLength (head))
     {
public class Sorter {
public void selectionSort (IntNode head)
  {
    for(IntNode node1= head; node1!=null ;node1 = node1.getLink())
    {
        IntNode min = node1;
        for ( IntNode node2=node1; node2!=null ;node2 =node2.getLink
        {
              if ( min.getData() > node2.getData()){
                    min =node2;
                 }
        }
       IntNode temp= new IntNode (node1.getData(),null)
          node1.setData(min.getData());
            min.setData (temp.getData());
    }
}
```

```
public static void main(String args[])
{
    IntNode head = new IntNode(-1, null);
    Sorter sorter = new Sorter();
    head.addAfter(4);
    head.addAfter(5);
    head.addAfter(7);
    head.addAfter(3);
    head.addAfter(6);
    sorter.selectionSort(head);
    for (IntNode i=head; i!=null; i=i.getLink()){
        S.O.P(i.getData());
    }
}
```

→ Program to implement insertion sort

```
public class LinkedListIs
{
    node head;
    node sorted;
    class node
    {
        int val;
        node next;
        public node(int val)
        {
            this.val = val;
        }
    }
```

```
void push (int val)
{
    node newnode = new node(val);
     newnode.next=head;
      head =newnode;
    }

   void insertionSort(node headref)
    {
      sorted=null;
       node current =headref;
          while(current!=null)
          {
            node next= current.next;
             sortedInsert (current);
                current =next;
            }
           head=sorted;
         }
       void      sortedInsert(node newnode)
       {
           if (sorted==null|| sorted.val >= newnode.val)
            {
              newnode.next= sorted;
               sorted = newnode;
               }
             else
               {
                  node current = sorted;
```

```
while ( current.next != null && current.next.val < newnode.val)
{
current = current.next;
}
newnode.next = current.next;
current.next = newnode;
}
}

void printlist(node head)
{
while (head != null)
{
System.out.print(head.val + " ");
head = head.next;
}
}

psvm(String[] args)
{
LinkedList list = new LinkedList();
list.push(5);
list.push(20);
list.push(4);
list.push(3);
list.printlist(list.head);
}
}
```

Given two linked list return the middle elements

① 1→5→2→10→1→6

② 1→5→2→10→1

Soln)

```
s = s.next;
f = f.next.next;
return s;
```

(or)

n: find the length
at pos n/2 +1 if odd length (even) then

```
flag = 0.
if(flag == 0)
    return n/2;
else
    retian n/2 + 1;
```

```
Middle (node h)
{
    while( f.next != null && f.next.next != null)
    {
        s = s.next;
        f = f.next.next;
    }
    if( f.next == null || flag == 1)
            return s.
        else
            return s.next;
    }
```

Implement merge sort for a single linked list

17→5→1→2→10→12→-3→4

```
ms (Node h)
{
    if(h == null || h.next == null)
        return h;
```

```
Node m = findMid(h);
Node nh = m.next;
m.next = null;
return merge(ms(h), ms(nh));
}
```

T.C of merge sort:- $N \log_2 N$.

→ Given a linked list arrange the terms

① I/p:- $1 \to 5 \to 2 \to 12 \to -3 \to 6 \to -1 \to \text{Z}$

o/p:- $1 \to -1 \to 5 \to 6 \to 2 \to -3 \to 12 \to \text{Z}$

② I/p:- $L_1 \to L_2 \to L_3 \to L_4 \to L_5 \cdots L_N$

o/p:- $L_1 \to L_N \to L_2 \to L_{N-1} \to L_3 \to L_{N-2} \cdots$

Soln ① $N^2, 1$ Brutforce soln.

$1 \to 5 \to 2 \to 12 \to -3 \to 6 \to -1$

$1 \to -1 \to 5 \to 6 \to 2 \to -3 \to 12$

② Two pointer

Break the linked list into two equal halves.

$1 \to 5 \to 2 \to 12 \} \to -3 \to 6 \to -1$

L_1: $1 \to 5 \to 2 \to 12$

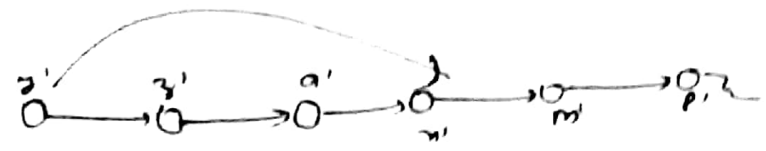
L_2: $-3 \to 6 \to -1$

Reverse the 2nd linked list.

L_1: $1 \to 5 \to 2 \to 12$

L_2: $-1 \to 6 \to -3$

h2 :



create a hash map

<Node, int>

y, 1	yy'
y, 2	yy'
a, 3	aa'
x, 4	xx'
m, 5	mm'
p, 6	pp'

<int, Node>

1, y'	
2, y'	
3, a'	
4, x'	
5, m'	
6, p'	

h2.r = m2.get (m1.get(h1.r))

h2.r = m.get (h1.r);

→ N, N

↓

to decrease the space

```
Node nn = newNode (h1.d);
nn.n = h1.n;
h1.n = nn;
h1 = h1.nn
```

```
h2.r = h1.r.n
```

```
h2.n = h1.n
h2.n = h1.n
h1 = h1.n
h2 = h2.n
```

Virtual Memory (VM)

→ If we are playing a movie of 10GB then as our RAM size is 4GB or 8GB we assume to take only 8GB.

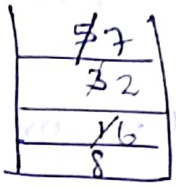
→ The pages which are to be taken are stored inside a page table.

→ Later whatever the pages are required they are swapped.

Scheduling alg.	Page rep
SRTF	LIFO
SJF	LRU
RR	LIFO
	MRU
	LFU

→ Given an array PN: 5,3,1,3,8,7,7,6,2 ...

k=4 means that we cannot put more than 4 page



PN, idx

5, 0 7, 15
3, 1 3, 2, 8
1, 2 6, 6
8, 4

P(i) → Present → 1

→ Not Present → full → k

→ not full → 1

PN k, k

→ Maintain another hashmap.

ordered	unordered
idx, PN	PN, idx.
0, 5	5, 0 7, 6
1, 3	3, 1 3
2, 1	1, 2
3, 3	8, 4
4, 8	
5, 7	
6, 7	

→ Take two hash maps one ordered and the other unordered. Inside the ordered hash map place the idx and page no. & inside unordered place pageno and idx.

→ idx is ordered put 5,0 in ordered and 0,5 in unordered then 3,1 in unordered and 1,3 in ordered.

→ 2,1 & 1,2 again while replacing 3 replace the 1's of 3 by 3 in ordered by 3,3

→ In unordered 8,4 and in ordered place 4,8

→ Now remove the page with least idx with value.

→ 5,0 is replaced by 4,5 in unordered and then again update the value of key 7 in unordered then remove 5,7 in ordered & place 6,7

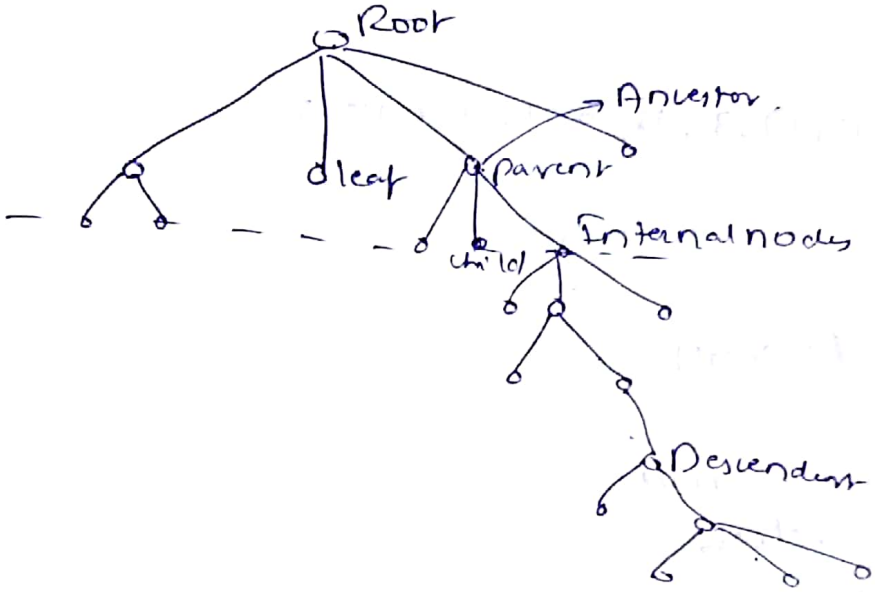
Using linked list rep

c.p.n = c.n
c.n.p = c.p
t.n = c
c.p = t
t = t.n
t.n = null

HashMap <Int, Node> hm;

```
Node* d = hm[i];
Node *st = d;
for(i=0; i<n; i++){
}
```

TREES (Basics)

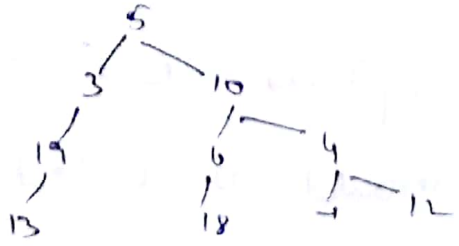


Height (Node) → Length of path from that node to leaf node.

Height (tree) → Height (Root)

Depth (node) → length of path from root node to that node.

Binary tree : Each node can have 2 children



Tree traversals:

Pre-order: (D, L, R) : 5, 3, 19, 13, 7, 10, 6, 18, 4, 4

Inorder :(L, D, R) : 13, 7, 19, 3, 5, 18, 6, 10, 4, 4, 12

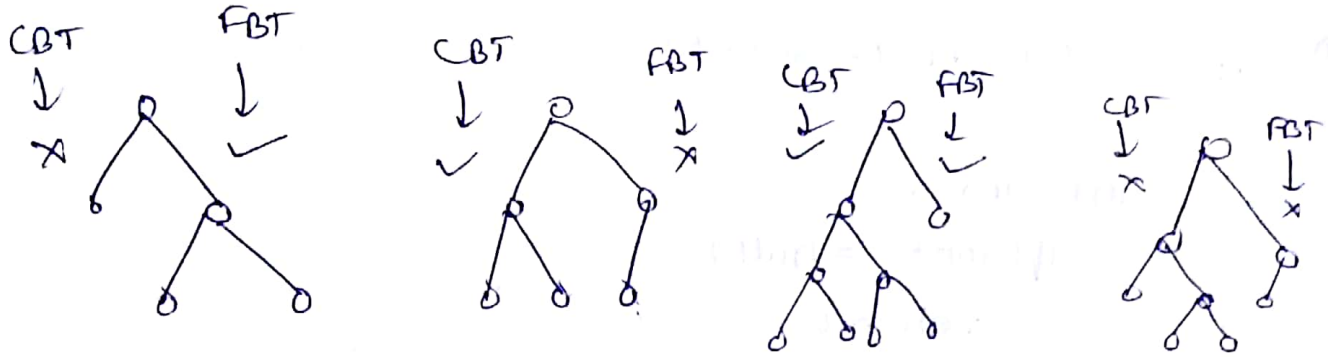
Post-order: 7, 13, 19, 3, 18, 6, 4, 12, 4, 10, 5

```
Inorder (Node root)
  {
  ① if (root == null)
        return
  ② InOrder (root.left);
  ③   print (root.data);
  ④   Inorder (root.right);
  }
```

Week-9
Day-1
6/10/2018

→ Complete Binary Trees (CBT): Level by level left right should be filled.

Full Binary Trees (FBT): Each node should have 0 or 2 child nodes.



→ Find the min and maximum no of nodes a CBT and a FBT can have for a given height.

	Min	Max
FBT	2^H	$2^{H+1}-1$
CBT	$2H+1$	$2^{H+1}-1$

Solve

① int size (Node *root)

② int sum (Node * root);

③ int maxValue (Node *root);

④ int height (Node *root)

⑤ void fillDepth (Node *root)

⑥ void levelOrder (Node *root)

① int size(Node root)

```
{
    if(root == null)
        return 0;
    return 1+size(beforoot.left)+size(root.right);
}
```

(2)

```
int sum(Node *root)
{
    int sum=0;
    if(root==null)
        return 0;
    return sum(root.left) + sum(root.right) +
           sum(root.data);
}
```

(3)

```
int maxValue(Node *root)
{
    if(root==null)
        return 0;
    return max( root.data, maxVal(root.left),
                maxVal(root.right));
}
```

(4)

```
int height(Node *root)
{
    if(root==null)
        return -1;
    return 1+ max(height(root.left), height(root.right
```

BOTTOM-UP approach.

(5) void fillDepth (Node *root, int d)

```
{
    if(root==null)
        return;
    root.depth=d;
    fillDepth(root.left, d+1);      → (9)
    fillDepth(root.right, d+1);
}
```

Here we follow TOP-DOWN approach.

→ Most of the problems in trees can be solved by : Technique + Traversal.

Technique may be TOP DOWN or BOTTOM UP

Traversal may be Inorder, Preorder & Postorder.

(6)

```
static void printLevelOrder(Node *root)
{
    int h= height(root);
    int i;
    for(i=1; i<=h; i++)
    {
        printGivenLevel(root, i);
        System.out.println();
    }
}
```

```
void printGivenLevel (Node root, int level)
{
    if (root == null)
        return;
    if (level == 1)
        System.out.println (root.data);
    else if (level > 1)
    {
        printGivenLevel (root.left, level-1);
        printGivenLevel (root.right, level-1);
    }
}
```

Other solutions:

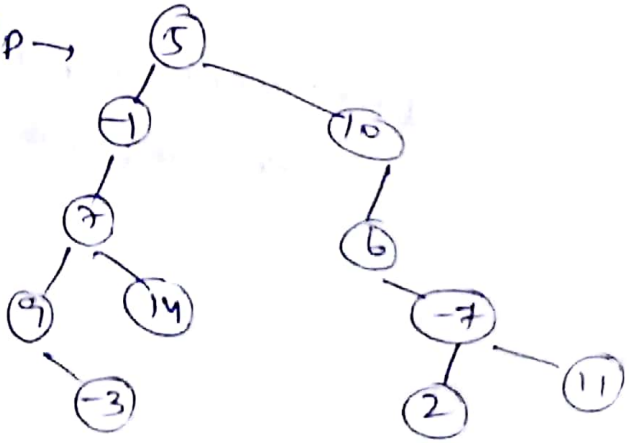
(1)

→ By using queue we can find the level order.

→ We have to maintain a queue which maintains the value and depth.

→ Take a counter cd = 0 if cd change then print in the newline.

If the given input is I/P →



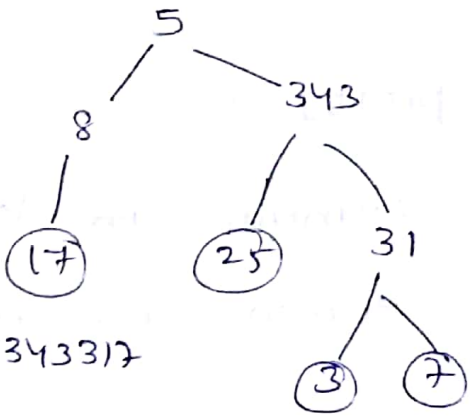
O/P:- 5 \n
-1 10 \n
7 6 \n
9 14 -7 \n
-3 2 11 \n

② Inserting bubbles

5 · -1 10 · 7 6 · 9 14 -7 · -3 2 11

5 \n
-1 10 \n
7 6 \n
9 14 -7 \n
-3 2 11

→ Calculate the sum of path formed along root node to leaf nodes.



O/P:- 5817 + 534325 + 534313 + 5343317

```
int sum (Node *root, int v)
{
    if (root==null)
        return 0;

    v = v * Math.pow(10, digits(root.data)) + root.data

    return sum(root.left, v) + sum(root.right, v);
}
```

→ Time complexity for all of them will be

$$T(N) = T(N-1) + 1$$

(or)

$$T(N) = 2T(N) + 1$$

So, it will be O(N)

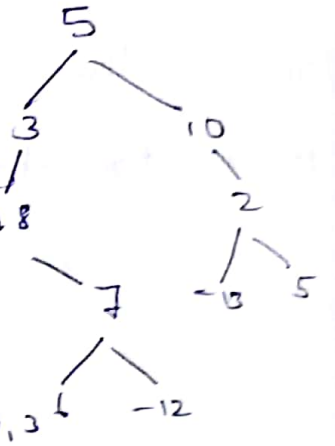
No matter best case or worst case

→ Given two arrays each of size n containing inorder and pre-order traversals, print the post-order traversal.

(L,R,D)

pre[] :- 5 3 18 7 -12 10 2 -13 15

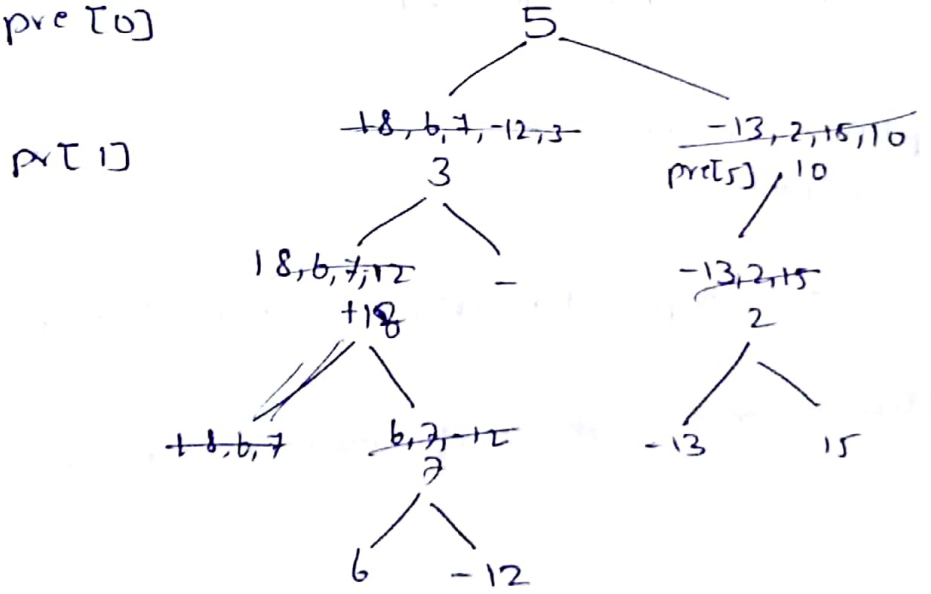
in[] :- 18 6 7 -12 3 5 -13 2 15 10



pre[0] = 5

elements on left will have 18, 6, 7, -12, 3

elements on right will be -13 2 15 10



```
idx=0   // Global variable   // Code to print the elements
                             // with tree formation.
Node  fun(int I[], pre[], lo, hi){
      if (lo > hi)
        return null;
                                  key
                                  ⇑
      int pos = search (pre[idx], in[], lo, hi);

      Node *root = CN (pre[idx++]);

          root.left = fun(in, pre, lo, pos-1);
          root.right = fun(in, pre, pos+1, hi);
          return root;
      }
```

// Code to print the elements without tree
// formation.

```
idx=0;
Void  fun(in[], pre[], lo, hi)
{
```

O(1) ← if (lo>hi) return;

N ← int pos = search (pre[idx], in[], lo, hi)

1 ← idx++;

T(N/2) ← fun(in, pre, lo, pos-1);

O(1 or T(N/2)) ← fun(in, pre, pos+1, hi);

S.O.P(in[pos]);

}

Time Complexity:- $T(N) = 2T(N/2) + N$

$= O(N \log N)$ in the best case

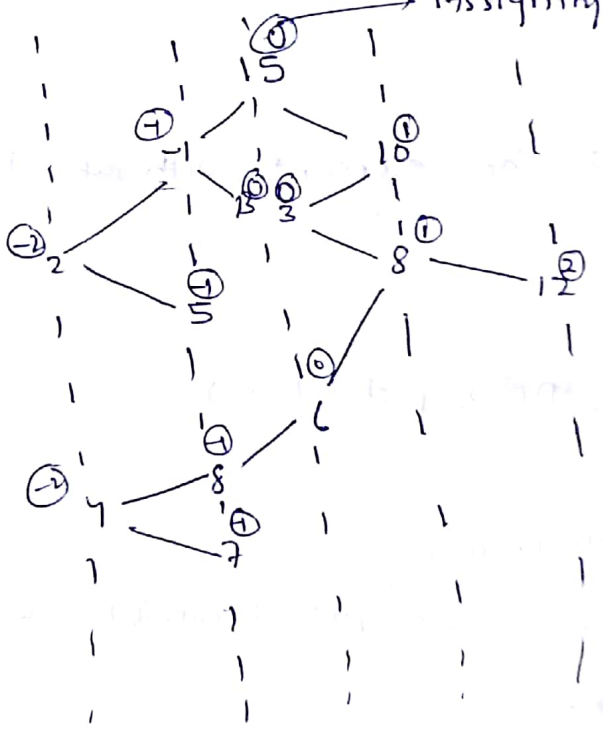
$T(N) = T(N-1) + N$

$= O(N^2)$

→ Given a tree as input. Print the elements level by level in a vertical order such that the output should be

Assigning no's.

I/P:-



O/P:-

2 4

-1 5 8 7

15 3 13 6

10 8

12

Soln) Assign no's vertically level by level such as -2, -1, 0, 1, 2

N log N, N

1) Then take a hash map put the integer and list of elements.

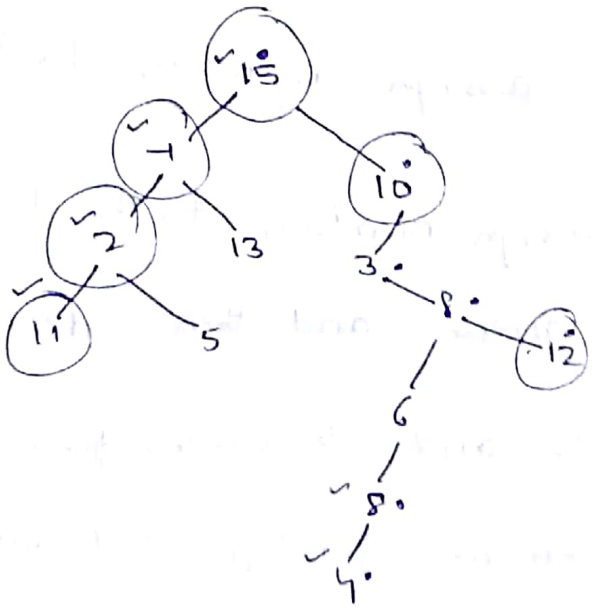
map<int, list<int>>

→ take an ordered map.

→ All the elements will be ordered & then we have to print.

2) N, N iterate from -3 to 2 and print

→ Print the left view of the Binary tree.



✓ → left view

• → right view

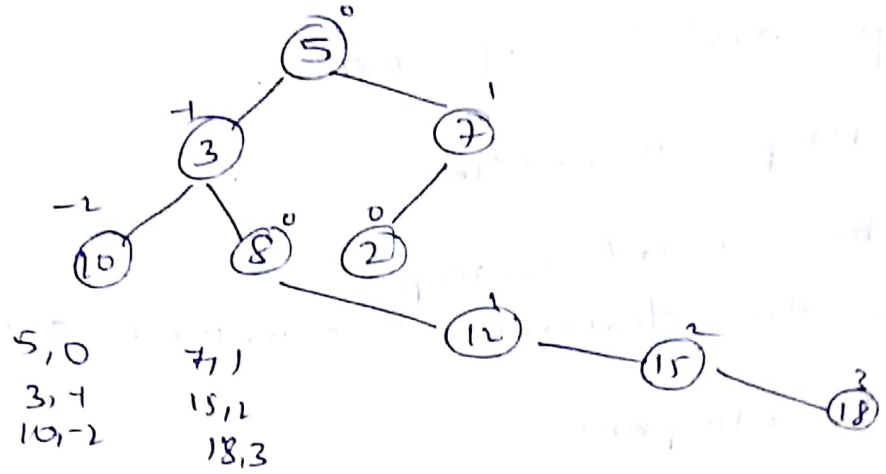
0 → Top view

Soln

Left view: First node of level order traversal.

Right view:- Last node of level order traversal.

Top view:



Call 5 with 0, 3 with -1, 10 with -2, 7 with 1, 15 with 2 put them in a map and then print.

(or)

take a queue assign numbers to the levels

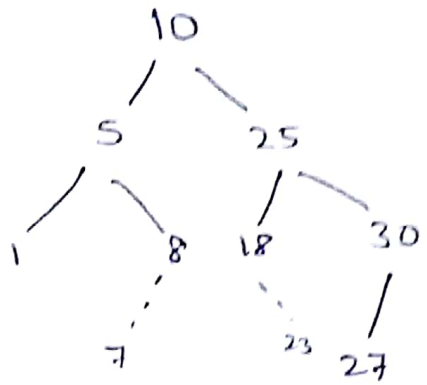
→ Take a queue assign numbers to the levels.

put (5,0) in the queue and then decrement so it will be (3,-1) and increment for right (7,1) then again decrement for left and inc. for right (10,-2) and (8,0) must be pushed into a queue.

→ Then push the element into hash mp.

(0,5)
(-1,3)
(1,7)
(-2,10) and then print.

→ Binary Search Tree



all (L) ≤ D ≤ all (R)

all elements of left should be less than the root

and " " of right " " greater " " " .

Recursive code to insert elements into a binary search tree

```
Node insert (Node *root, int k)
 {
     if( root==null)
        return CN(k);
      if(k<root.data)
        root.left = insert(root.left, k);
       else
         root.right= insert(root.right, k);
        return root;
 }
```

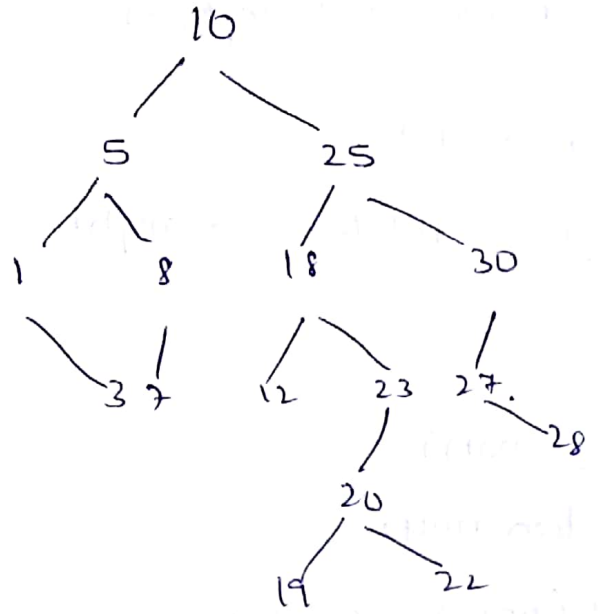
Iterative code :

```
int t;
Read t;
while(t--){
int n;
Read n;
Node *root = null;
while(n--)
{
  read x;
  root= insert(root, x);
}
```

// code to search an element in the tree.

```
bool search(root, k)
    {
        if(root == null)
            return CN(x);
        if(k< root.data)
            search(root.left, k);
        if(k > root.data)
            search(root.right, k);
        if(k == root.data)
            return true;
    }
```

// code to delete an element from the tree.



Case 1:
→ If there is no (child) leaf and we have to delete it then set is as null.

Case 2:
If it has only one leaf. Then swap the element & leaf and then delete it.

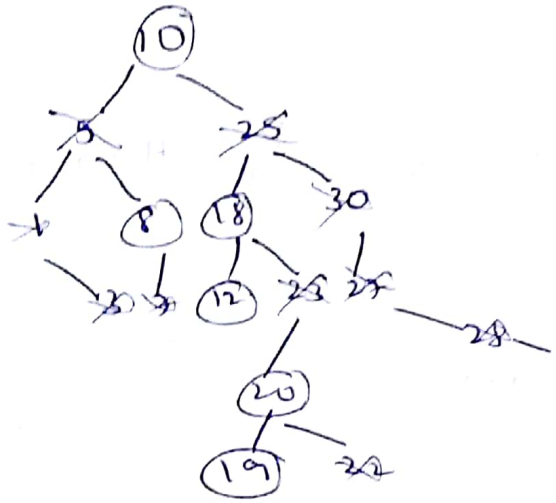
Case 3:
If it has two leaf nodes then take the smallest element from right or largest from left replace it & delete it.

```
Node *delete (Node *root)
{
    if (root == null)
    return null;
```

```
if (k < root.data)
    root.left = delete(root.left, k);
else if (k > root.data)
    root.right = delete(root.right, k);
else
{
    if (leaf(root))
        return null;
    else if (onlyleft(root))
        return root.left;
    else if (only right(root))
        return root.right;
    else
    {
        root.data = findMin(root.r);
        root.r = delete(root.r, root.data);
    }
    return root;
}
```

Surely working code to delete in note.

Print the elements with given range of nodes.



1) N → No. of nodes
   N * H, 1
   → For each node call the delete function.

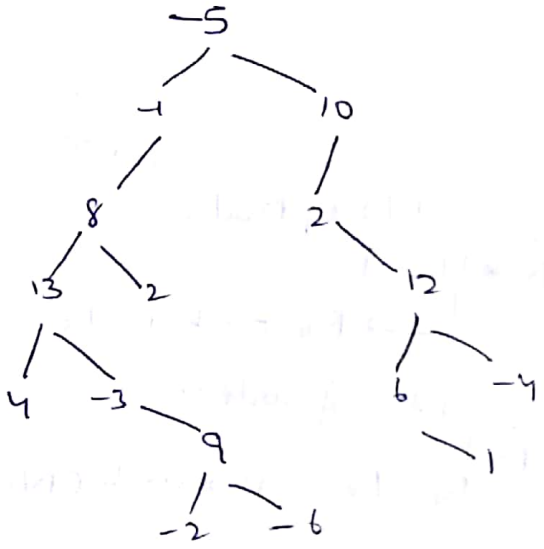
2) N, 1 → No. of vertices.

```
Node BSearch(Node root)
{
    if(root==null)
        return root;
    if(root.data<L)
        return f(root.right);
    if(root.data>R)
        return f(root.right);
    root.left = f(root.left);
    root.right = f(root.right
    return root;
}
```

f → BSearch.

→ Given two nodes x & y (or) a binary tree return the path between two nodes.

→ If the given nodes are (2, -2) (L, R) then return the following.



L_1: P_1: ~~-5~~, ~~-1~~, (8), 13, -3, 9, -2

L_2: P_2: ~~-5~~, ~~-1~~, (8) 2

P_1 indicates the path between root node & the ~~left node~~ L

P_2 indicates the path between root and R.

Then remove the common elements except the last common element then you will get the path.

```
boolean helper (Node *root, Node *k, *List L)
{
    if ((root == k) || (root.left, k, L) || helper (root.right, k
    {
        L.add (root);
        return true;
    }
}
```

List<Node *> path (Node *x, Node *y, Node *root)

{

helper (root, x, L1);

helper (root, y, L2);

}

→ Given above the tree calculate its maximum sum path.

If a node reduces the sum then we have to take out the node.

1) $N^2 * N, N$
   - → going through the tree
   - → going through the path.

maxPath ( ) {

ans = max (ans, root.data + f(root.left, . .)
+ f(root.right, . .))

2) $N(N+N)$
   - → left traversal
   - → right traversal

follow any of the traversal and calculate max.

3) N, 1

int sum (Node root)

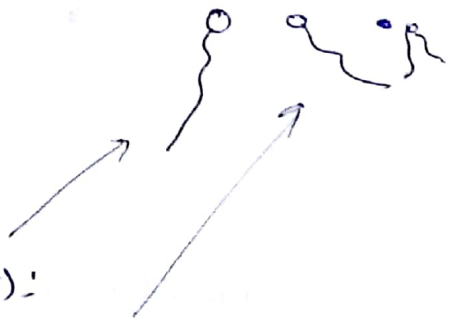
{

if (root == null)

return 0;

l = sum (root.left);

r = sum (root.right);



sp = max(l+root.data, r+root.data, root.data);

single path

ans = max(ans, sp, l+r+root.data);

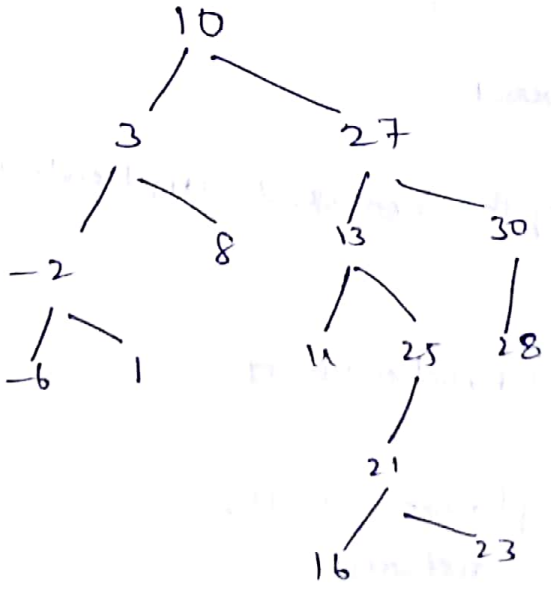
return sp;

}

→ Given a Binary tree check if it is a Binary Search Tree

1)

```
boolean BST(Node *root)
{
    comparision b/w left and right
    node won't work
}
```



2) max(L) < D < min(R)

↳ N+1

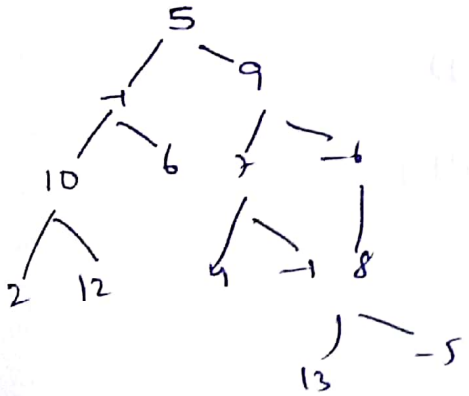
3) Do in-order traversal. Store the resultant in an array if it is sorted then it is a binary search tree. N, N!

4) Optimize the second solution to work in N,, from every function call return max and min.

```
isBst(lo, hi, root)
{
    return  r.d >= lo && r.d <= hi && isBst(r.l, lo, r.d)
            && isBst(r.r, r.d, hi);
}
```

→ Find the Least Common Ancestor (LCA).

1) LCA of 2 nodes in BST



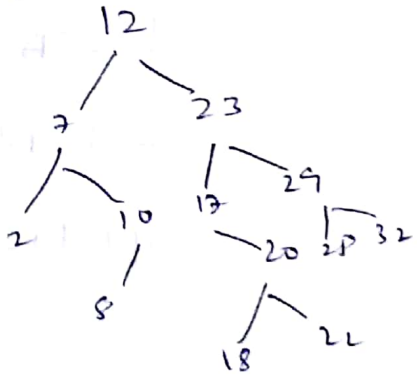
I/P O/P

7, -5 → 9

12, -6 → 5

9, 13 → 9

2) LCA of 2 nodes in a BST



I/P O/P

10, 20 → 12

18, 32 → 23

20, 23 → 23

Soln)

1) BT
Find path → l_1, l_2
→ N, N

2) Pre order → Search → Return → N, 1

1) BST
Find path → l_1, l_2
→ n, n

2) Searching both at the same time → n, 1

① BT N, N

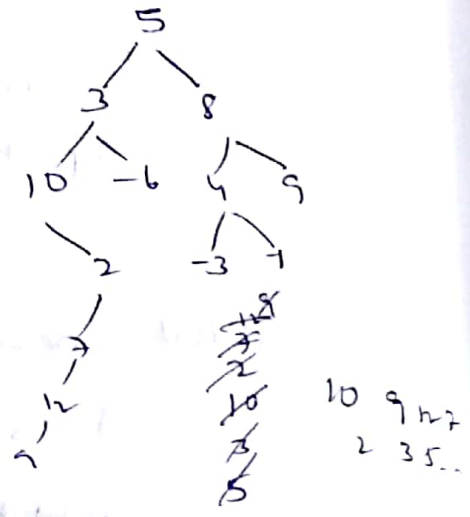
```
LCA(root, x, y){
if(root.d == x || root.d == y)
    return root;
l = LCA(root.left, x, y);
r = LCA(root.right, x, y);
if(l != null && r != null)
    return true;
if(l != null)
    return l;
if(r != null)
    return r;
}
```

(2) BT

```
class HeralincPreOrder {

stack <Node *> s;

HeralincPreOrder ( Node *root ) {

        //s =new _initialise.

        s.push(root);

}

Node  next( ){

        t = s.pop( );

        if(t.right != null)

            s.push (t.right);

        if(t.left)=null)

            s.push(t.left);

        }

Heralinc__PreOrder Obj = new HeralincPreOrder (root);

while(obj.hasNext( ) == T)

{

    Node *n = obj.next;

            return n.data;

}
```

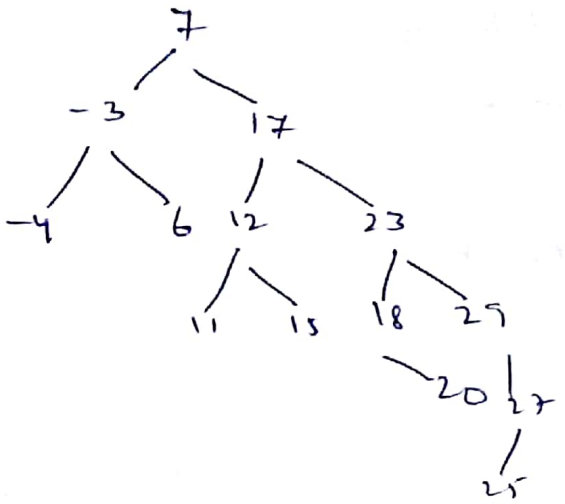
implementing iterative code for inorder traversal.

```
while(1)
{
  while(root!=null)
  {
    s.push(root);
    root=root.left;
  }
  Node th=root;
  if(s.empty()==F)
  {
    Node th=root;
    root=s.pop();
    print(root.data);
    root=root.right;
  }
  return th;
}
```



Time Complexity: N+H → each node is visited only once.
↳ at any point of time you will have only one node at every level.

→ For the given Binary Search tree return true if the sum of two nodes is equal to the value of k. (a+b=k)
↓ ↓
nodes.



$a+b=k$

$k = 18, 21, -2$

Sol) → Consider all pairs & find one.

1) $N^2, 1$

2) $N \times H, 1$

↳ for each node we have to do a search taking

$k - a = b$

3) N, N

Applying two pointer : Apply inorder you will get a sorted array and then search for the difference b/w two elements by using two pointers.

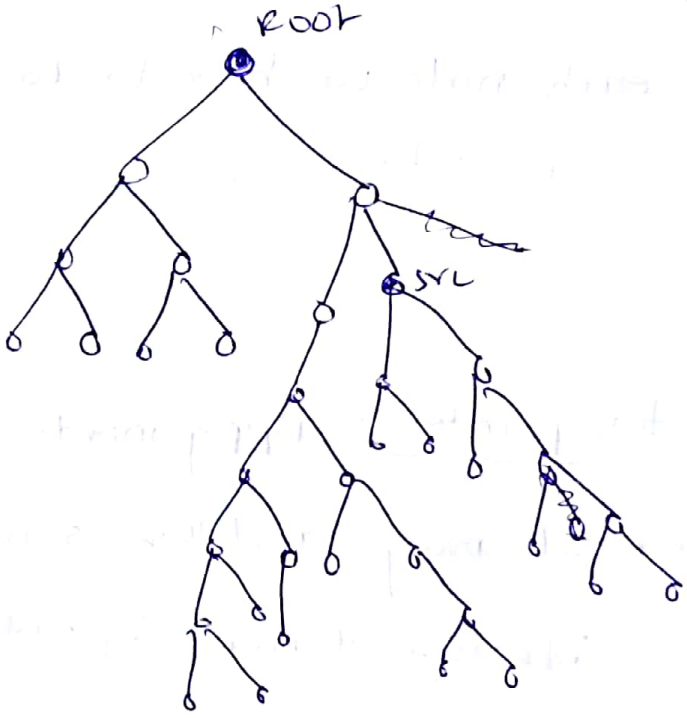
$P_1 \rightarrow LDR \rightarrow$ Inorder code.

$P_2 \rightarrow RDL \rightarrow$ rev. inorder where ever left-right & vice versa.

```
LDR     p1 = new LDR(root);
RDL     p2 = new RDL(root);
        while(p1 != p2)
        {
            if(p1.data + p2.data == k)
                return true;
```

```
if (p1.data + p2.data < k)
        p1 = p1.next;

    else
        p2 = p2.next;

    }
return false;
```

→ Calculate the no. of node from a given source node at a distance k.



1) N^2, N

```
int f(root, k)
{
    if(root == null)
        return 0;
    if(k == 0)
        return 1;
```

return f(root.left, k-1) + f(root.right, k-1);

}

L(i).left = L(i-1)

ans = ans + f(root.right, k-i-1);

else

f(root.right, k-i-1);

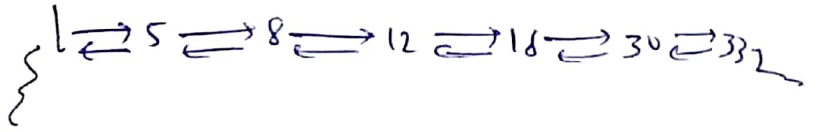
→ Find path

1) N+N+N, N

↳ Iterate through the list

→ Convert

Sorted Double Linked list to Balanced Binary search Tree.



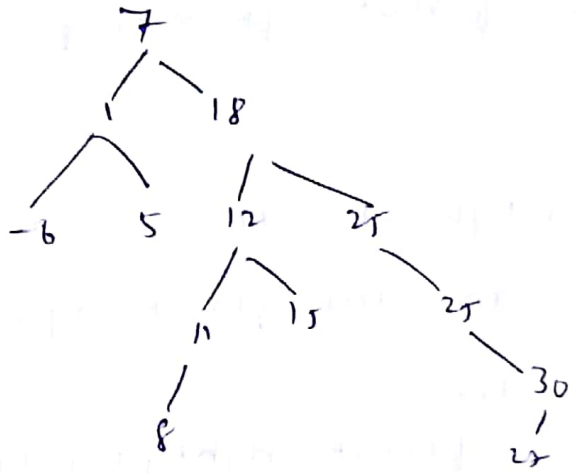
m = findMid(head);

find mid break it create function call to

create left sub-tree. function call to

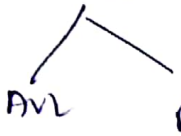
create right sub-tree.

Convert BBST → SDLL.



$-6 \rightleftarrows 1 \rightleftarrows 5 \ldots$

① BBST



(*never asked bout...)

② Huffman-encoding* (important).

13/10/18

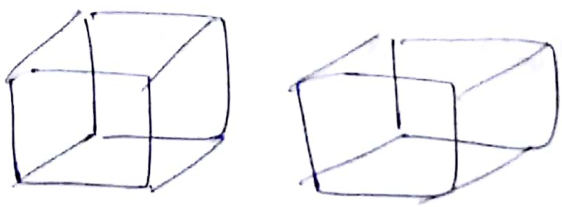
Week-10

Day-1

PUZZLE

Given two dice you can put a digit on any side of it between 0 to 9 so that we can get all the numbers

01
02
03
.
.
.
31



Cube1: 0,1,2,3,4,5

Cube2: 0,1,2,6,7,8

6 can act as 6 & 9.

Given N words each word can have m alphabets & given k no. of queries. Find the number of words which start with the given query.

eg:- shard
smart
struct
serialization
data
disjoint
union
draw
docke
star
store
share
dock
structure
algo
suffer
sand
snow
salt
short
sharepoin
stack
step

Sol^n)

1) $Q \times N \times m$,

2) $m \times N\log N + Q(m\log N)$,
- m: length of each string
- $N\log N$: sort
- Q: queries
- $m\log N$: size
- binary search

$T(N) = 2T(N/2) + Nm$

$= mN\log_2 N$

3) $Nm^2 + Q(m)$, Nm^2
- N: N words
- m^2: Each word is to be stored in the hash map as a string.

Using hash map we can store them as string & integer.

<String, int>

While storing

S, x2

sh, x2

sha, x2

shar, x2

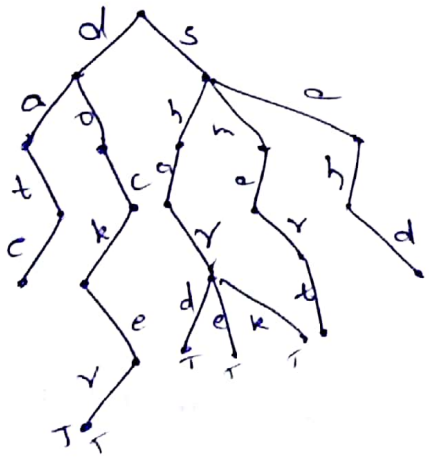
shard, x2

share x3

4) TRIE

```
arr[0] = 'a'
arr[1] = 'b'
arr[2] = 'c'
d - 'a' = 3.
```

```
class Node
{
  int data;
  Node arr[26];
  int cnt = 0;
  // boolean flag = false;
  Node createNode()
  {
    Node n = new Node();
    for (int i = 0 to 25)
      n.arr[i] = null;
    return n;
  }
}
```

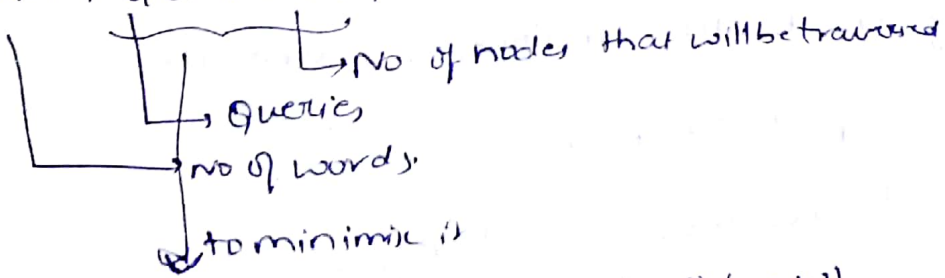


```
main()
{
  Node root = new Node();
  int n;
  Read n;
```

```
while(n--)
{
 read word;
 root=insert(root,word);
 }
}
Node   insert(Node root, String word)
{
  for(int i=0; i<w.size(); i++)
   {
     if(root.ar[w[i]-'a']==null)
      {
        root.ar[w[i]-'a']=new Node();
      }
       root=root.ar[w[i]-'a'];
        root.~~cnt++~~;
             flag = T;
      }
    }
```

to count the no. of nodes after creating atrie.

NxM + Q(M+MN), NM




```
Node  insert(Node root, string w)
      {
          for(int i=0; i<w.size(); i++)
           {
           if(root.ar[w[i]-'a']==null)
```

```
{
    root.ar[w[i]-'a'] = new Node();
  }
  root = root.ar[w[i]-'a'];
  root.cnt++;
 }
)
```

→ N×m + q(m), Nm

Return the count

```
int query(Node root, String w)
{
    for(int i=0; i<w.size(); i++)
    {
        if(root.ar[w[i]-'a'] == null)
            return 0;
        root = root.ar[w[i]-'a'];
    }
    return root.cnt;
}
```

→ For a given N×m matrix count the number of distinct rows.

$1 \le N \le 1000$ $1 \le M \le 50$

Mat N×M:

①	1	0	1	1	0
②	0	1	1	0	1
③	1	0	1	1	0
④	1	0	1	1	0

③ 0 0 1 0 1 1 0 1

④ 0 0 0 1 0 1 0 1

⑤ 0 1 1 0 1 0 1 1

⑥ 0 1 0 1 0 1 0 1

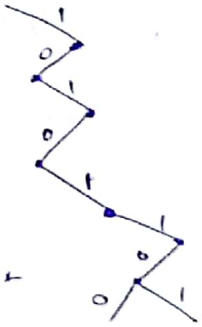
Solution:

1) N X NM, 1
   - N → For N words - rows
   - 1 → For NM use count

   If the given string occurs increment the count by 1.

2) NM, N
   - N → space taken by hashset
   - NM → Using a hashset converting the entire string into integer which would take NM then count the distincts

3) NM, NM → Using a binary trie
   - NM → space
   - NM → time to insert in a trie



Traverse through the entire trie when you get a new node increment the count which means you have got a distinct element.

→ Given an array with positive element

$0 \le arr[i] \le 10^6$

arr[N]: 5 2 10 3 18 25 23 14 20

Find the pair which give max. XOR.

$$\bigvee_{i=0}^{N-1} \bigvee_{j=i+1}^{N-1} (arr[i] \wedge arr[j])$$

1) $N^2, 1$

5 00010 2 10 00011 3 18 11001 25 01110 23 00100 14 20 4

10111 10100 10010

x: 00101

y: 10010

10111 → 23

x = 00010

y = 10100

10110 → 22

x = 01010 ← (10)

y = 10100 ← (20)

11110 → 30

x = 00011

10100

10111 → 22

x = 1 0 0 1 0

y = 0 1 1 1 0

1 1 1 0 0 → 28

x = 1 1 0 0 1

0 1 1 1 0

1 0 1 1 1 → 23

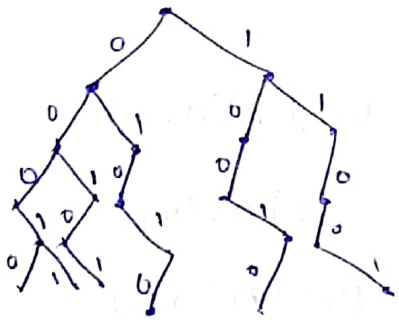
x = 1 0 1 1 1

0 1 1 1 0

1 1 0 0 1 → 25

2) N*30+N*30, N*30

N, N



```
Node insert (Node root, int x)
{
    for (i=30 to 0)
    {
        if (CB(x,i))
        {
            if (root.right = =null)
            {
                root.right=CN(x);
            }
            root = root.right;
            }else {..}
```

Find the max of the sub-array which will give maximum XOR.

ar_N:	5	2	10	3	18	25	23	14	20
	0	1	2	3	4	5	6	7	8

$i \le j \le N$

$(ar[i] \wedge ar[i+1] \wedge ar[i+2] \wedge \dots ar[j])$

Solutions:

1) $N^2 \times N, 1$ → calculate XOR for each sub-array
   ↳ calculate max.

2) $N^2, 1$ Use soln 1, and carry forward

$ar[l:r] = ar[l:r] \wedge ar[l:$

$[6,6] = [6,6] \wedge [0,5]$

$[5,6] = [0,6] \wedge [0,4]$

$[4,6] = [0,6] \wedge [0,3]$

$[3,6] = [0,6] \wedge [0,2]$

$[2,6] = [0,6] \wedge [0,1]$

$[1,6] = [0,6) \wedge [0,0]$

$\downarrow x \quad \downarrow y$

We only need to get the maximum

TRIE:

```
insert(root, 0)
{
    x:=0;
    for(i:=0 to N-1)
    {
        x=x^arr[i];
        ans=max(ans, query(root, x));
        insert(root, x);
    }

int query(Node root, int x)
{
    for(i=20 to 0)
    {
        if(CB(x,i))
        {
            if(root.left)
            ans = ans + 2^i;
            root.left = T;
        }
        else
        {
            if(root.right)
            ans = ans + 2^i;
            root.right = 'T';
        }
    }
```

→ Given a dictionary {abc, abcd, abcde, efg, h, def, fgh}

str: abcdefgh

Divide the string into partition such that they exist in the dictionary

Then calculate the min. no. of partitions.

① T/F

② minimum

Soln:

1) $2^{N-1} \times N$, 1
   - ↳ finding if string partitioned exists or not for N string
   - ↳ possible number of combinations

```
print (f(0, str, N));

f(idx, str, N)
{
    if (idx == N) return 0;

    cans = ∞;

    for (i := idx to N)
    {
        if (str[idx])
            cans = min(cans, 1+f(i+1, str, N));
        }
        return cans;
}
```

Given N words size of each m $1 \leq len(w) \leq m$

By appending two words check whetha we are getting a paliondrome ornot

→ abccde
xyzax
✓ abyza ←
✓ data ←
yz
✓ azybc ←
✓ atad ←

→ possible pairs

Soln) 1) N^2 m → check throughout the length of the string.

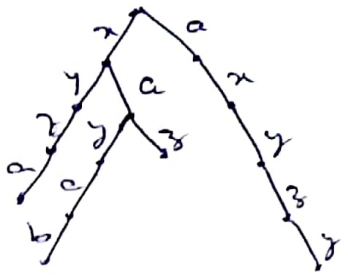
2) Use trie

→ Dictionary : {xyz, abc, abcx, axyzy, xyza, xaz, xayab }

. There are two players A & B.
Both play optimally
Each append a letter to the end the one who completes a word present in the dictionary will win.

$\forall$ moves L. current pos L
(if there exists all moves to win current pos is L

→ if there exist 1 the current pos is W



→ Given an array put it into something and perform getMinimum and deleteMinimum type of operations.

Soln)

	Insert(x)	getMin()	delMin()
Unsorted array	1	N	N
Sorted array	N	1	1
SLL	N	1	1
Stack	1	1	N
BST	H	H	H

BBST H = log N

MinHeap → CBT
→ Node < childNode

if H is height of the tree then no. of nodes are $2^{H+1} - 1$

if no. of nodes are N height of the tree will be $\log_2 N$

This is as we are maintaining a CBT

Implementing MinHeap in Java.

```
public class MinHeap
{
    private int[] Heap;
    private int size;
    private int maxsize;
    private static final int FRONT = 1;

    public MinHeap(int maxsize)
    {
        this.maxsize = maxsize;
        this.size = 0;
        Heap = new int[this.maxsize+1];
        Heap[0] = Integer.MIN_VALUE;
    }
```

```
private int parent (int pos)
{
    return pos/2;
}
private int leftChild (int pos)
{
    return (2*pos);
}
private int rightChild (int pos)
{
    return (2*pos)+1;
}
private boolean isLeaf( int pos)
{
    if (pos >=(size/2) && pos<=size)
    {
        return true;
    }
    return false;
}
private void swap (int fpos, int spos)
{
    int tmp;
    tmp= Heap[fpos];
    Heap[fpos] = Heap[spos];
```

```
Heap[spos]= tmp;
}
private    void   minHeapify(int pos)
{
    if( isLeaf(pos))
    {
       if ( Heap[pos] > Heap[ leftChild (pos)] ||
          Heap[pos] > Heap[rightChild (pos)])
        {
        if ( Heap[ leftChild(pos)] < Heap[rightChild(pos)])
        {
          swap (pos, leftChild(pos));
          minHeapify (leftChild(pos));
        }
        else
        {
          swap (pos, rightChild(pos));
          minHeapify (rightChild(pos));
        }
       }
    }
}
public    void    insert(int element)
{
   Heap[++size] = element;
          int current = size;
```

```
while (Heap[current] < Heap[parent(current)])
{
    swap (current, parent(current));
    current = parent(current);
  }
}

public void print(),
 {
  for(int i=1 ; i<=size/2 ; i++)
  {
   S.O.P("parent" + Heap[i] + "LC:" + Heap[2*i]
   + "RC" + Heap[2*i+1]);
     S.O.P();
   }
  }
public void minHeap(),
{
    for(int pos=(size/2); pos>=1; pos--)
    {
      minHeapify(pos);
    }
  }
```

```
public int remove()
{
  int popped = Heap[FRONT];
    Heap[FRONT] = Heap[size--];
      minHeapify(FRONT);
        return popped;
    }

psvm(String args[])
{
  S.O.P("The min heap is:");
  MinHeap min = new MinHeap(15);
          min.insert(5);
          min.insert(6);
            :   :
        min.minHeap();
        min.print();
      S.O.P("The min val is" +min.remove());
    }
```

Week-10
Day-2
21/1/19

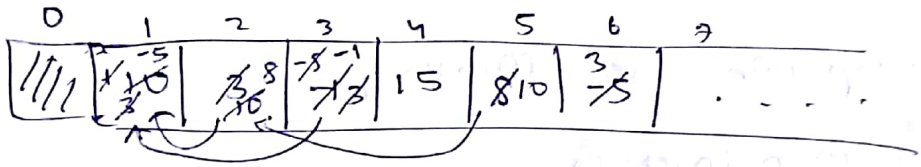
→ Implement a Complete Binary Tree

insert(x)
T.C → $\log_2 N$

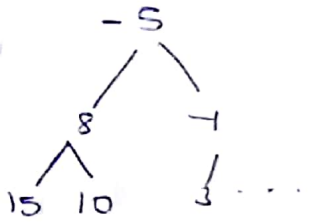
ar N: 10 3 → 15 8 −5 6 −10 25 2

Given an array insert them into the complete Binary tree.

Condition for that is Node < Child Node



// inserting into a CBT




```
insert(x)
{
  int idx=1;
  ar[idx++] = x;
   int t= idx-1;
  while ( t!=1  && ar[t] < ar[t/2]) {
        swap (ar[t], ar[t/2]);
         t = t/2;
      }
}
```

delete the element at 6th position

```
delete()
{ ar[1] = ar[--idx];
  t = 1;
  while (2t < idx){
    int c = 2t, s = ar[2t];
    if (2t+1 < idx && ar[2t+1] < s)
    {
      s = ar[2t+1];
        c = 2t+1;
    }
    if (s < ar[t])
    {
      swap(t, c);
        t = c;
    }
    else
      break;
  }
```

→ to implement maxheap replace '<' with '>'.

→ In java PriorityQueue is inbuilt for minheap and for maxheap we have to overwrite the comparator.

if you implement using arrayList

idx will start at 0

for pchild will be → 2p+1, 2p+2

for child(c) parent(p) will be → (c-1)/2

* "While implementing a library consider it is dynamic & use templates"

Q: Given an array and a value k you have to find k smaller elements.
You can print them in any order.

arr: 10 3 15 -1 12 05 17 8 20

k = 4

o/p:- 3, -1, 5, 8

Soln)

1) $N \log N$, → sort the entire array and pick the first 4 elements.

2) $N \log N + k \log \frac{N}{2}$, N → insert N elements into a minheap
→ do getMin(), O(1)
do delMin(), O(log n)
so it is $k \log \frac{N}{2}$

3) $k \log k + (N-k) 2 \log k$, k → heap size
→ insert 4 elements into the heap
→ Remaining elements
→ getMin(), O(1)
delMin(), O(log k)
insertOne Ele O(log k)

Insert k elements in a heap of size k.

(10, 3, ~~15~~ 12, 7) 12

Consider the next element 12 compare it with the largest element in the heap and then replace it with the largest element if the next element is smaller.

∴ 15 > 12 so replace.

(10, 3, 12, 7) 5

12 > 5 so replace

10, 3, 5, 7.

→ To find k smallest elements use max heap. and for k biggest elements use min heap.

4) Implement Quick sort

10, 3, 7, 12, 05, 8, 15, 17, 20

→ 15 is pivot so we neglect right half

→ Then 5 will be pivot 3, 7, 5, 8, 10, 12

3 + 1 = 4.

So we have got 3 small elements and now we are looking for just one element.

$T(N) = 2T(N/2) + N$ (B.C) $\rightarrow N\log_2 \frac{N}{2}$ ← if pivot is at the middle

$T(N) = T(N-1) + N$ (W.C) $\rightarrow N^2$

↳ pivot is at the last

$T(N) = T(\frac{9N}{10}) + T(N/10) + N$ (A.C) $\rightarrow N\log_{\frac{1}{2}} N$

90% of elements are to be considered which are on left side of pivot & 10% elements are on right side

So, O(N), for quick select.

(9) Given an array sort it.

It is a k-sorted array which means when the elements of the array are sorted they move at max k positions from their initial

	0	1	2	3	4	5	6	7	8	9
ar_N:	3	10	5	-1	20	12	15	11	25	22
ar_N:	-1	3	5	10	11	12	15	20	22	25

K=3

→ k=3 so you can move the

elements k positions towards left or right.

Soln

1) $N \log_2 N$, 1 sort the entire array

2) $N \log_2 k$, k → put all the elements

3, 10, 5, -1

put k+1 elements into minheap

delete the last element and it will reach its position

Q) Given a rowwise sorted matrix print the elements in a sorted order.

$Mat_{N \times m}$:

3	10	12	18	25
-1	4	8	17	18
2	5	9	11	25
-5	-2	15	20	34
8	12	16	23	28
5	9	73	85	92

Soln)

1) $NM + NM \log_2 NM$, $N \times m$ → sorting them

↳ copying the elements will take time of $N \times m$

2) $N^2 M$, NM Merge the first row and the second, third, so on

③ NMxN, NM take N pointers and compare

~~such that~~

④ $NM\log_2 N$, N

Inserting 1st column elements into a heap. Delete minimum element - 5 and go to insert element - 2 then find min again del it and insert the next element & so on.

n, i, j ← indices
↳ element.

i, j+1 j+1<M-1

4) $T(N) = 2T(N/2) + NM$

$= (N\log_2 N) \times M, NM$

→ Apply merge sort on each row considering each row as an element.

for every sub-array 0 to i find and print the median.

	0	1	2	3	4	5	6	7	8	9	10
arr:	-1	10	3	5	7	18	25	15	13	20	12

(0:i)

o/p:-

-1, -1, 3, 3, 5 . . .

-1

-1 → -1, 10

-1 → -1, 10, 3 → -1, 3, 10 (middle element when no of elements odd)

3 → -1, 10, 3, 5 → -1, 3, 5, 10 (left middle element or least middle element)

3 → -1, 10, 3, 5, 7 → -1, 3, 5, 7, 10

5

⋮

Soln>

1) $N * N\log N$,

Sort each sub-array and print the median.

2) N^2, 1 take the elements, compare insert and print.

3)

-1, 5, 3 10, 7

size (MaxH) — size (MinH) = [0,1]

Left (MaxH) Right (MinHeap)

→ Compare greatest of left with next element. If next element is greater put it on the right half.

→ Number of element on left hand side should be greater than or equal if not then compare the max(left) with min(left) and then swap.

Left → M.H

Right → minH

Left → arr[0].ans = arr[0]

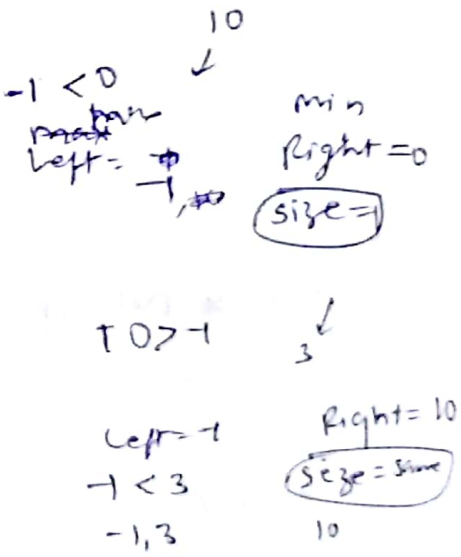
for i = 1 to N-1

arr[i] > left.getMax()

3 < 5

so

Shift 5



T.C → $N \log_2 N$, N

Check if maxHeap is empty
if yes then add the element
else if (both have same size)
{
if (ele < smallest of minheap)
add it to maxheap
else it means
the element is greater
add it to minheap

4. Sort a 100GB file using only 10MB RAM

totally we have 10^5 files

Apply matrix & min heap concept.

5. Return n^{th} fibonacci number.

1, 1, 2, 3, 5, 8, 13 . . .
1 2 3 4 5 6 7

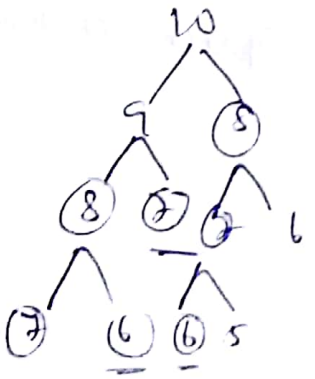
Recursive (TOP-DOWN)

```
int fib(int n)
{
  if(n==1||n==0)
     ret 1;
  ret fib(n-1)+fib(n-2);
}
```

Tabulation
Iterative (use array)

```
int fib(int n)
{
  int ar[N+1]
    ar[1]=ar[2]=1;
  for(i=3 to N)
    ar[i]=ar[i-1]+ar[i-2]
    ret ar[i];
}
```

BOTTOM-UP



} overlapping sub-problems.

take the smallest element from maxheap & put it in minheap
}
else if (maxheap is greater than minheap) {
  if (ele > greatest of maxheap)
    add it to min heap

else {
  add it to maxheap.
  ~~add it to~~
  remove from maxheap
  put it in minheap
}

Dynamic programming

DP state:- Ways to reach the nth number

→ ar[i]

DP expression: which computes DP state

→ ar[i-1]+ar[i-2]

DP table : 1D array (representation)

Base condition: initialisation

→ ar[1] = 1 ar[2] = 1

final ans → ret ar[N]

T.C → No. of states * Time of state

S.C → The amount of space it occupies ar[1:N]

Recursion with memoization

```
int ar[N+1]={-1}

int fib(int N)
{
    if (N==1 || N==2)
        return 1;
    if (ar[N]==-1)
    ar[N]=ar[i] fib(N-1)+fib(N-2)
    } return ar[N];
```

Here we are storing the values in the array for every 'n'.

Suppose if you want '8th' value then it is already stored while comping 9th value so it is called directly.

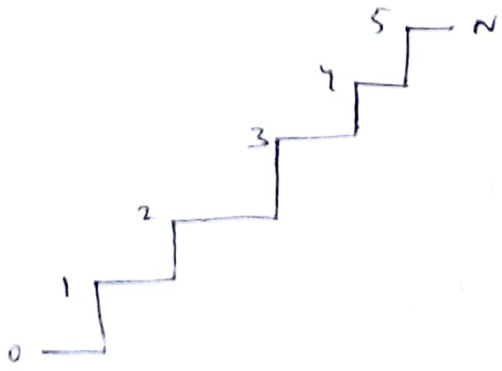
→ Time complexity of fibonacci has been reduced to O(n)

Note:

→ If your problem has concept of overlapping sub-problems then you can use dynamic programming.

→ Iterative code has an advantage of space reduction because as the current state depends upon limited previous states

Q. Find the no. of ways to reach the Nth step



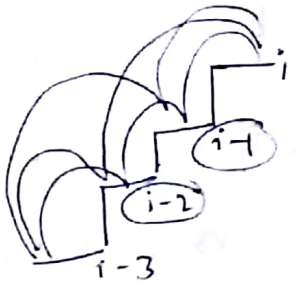
So, to reach to the 5th step you can go using

1,1,1,1,1

1,2,2

2,2,1

1,2,1 soon.



overlapping sub-problem

→ So to reach to ith step you have to definitely pass through i-1 and i-2 steps.

→ As this process is repetetive in nature these can be considered as an overlapping sub-problem and you can apply dynamic programming.

dp-state

dp[i] → # way to reach ith step

dp-expression

dp[i] = dp[i-1] + dp[i-2]

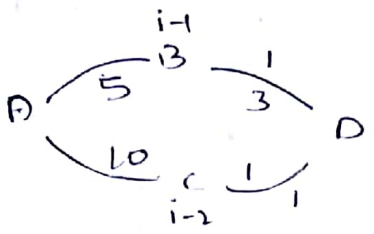
dp table

1D array

Base conditions

dp[0] = 1

dp[1] = 1



→ In this diag, you can see the no of ways one can travel from one point to another.

$5 \times 3 + 10 \times 1 = 25$ ways.

→ So, the points B and C can be considered as overlapping sub-problems.

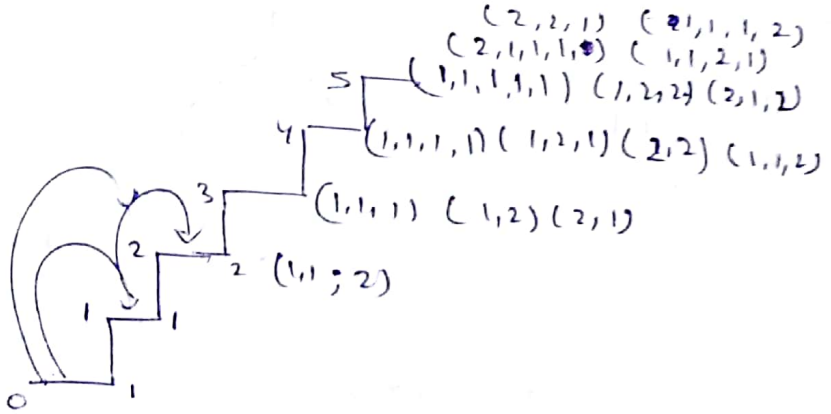
→ i-1 at B indicates to reach i-1 from initial state

→ D is the final point so there exists only one way.

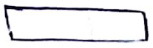
→ similarly with C which is considered as i-2th position.

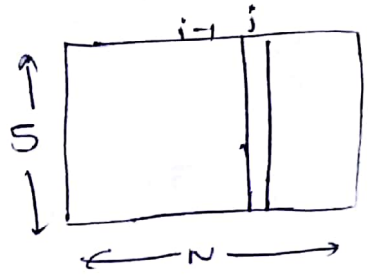
So dp[i] = dp[i-1]*1 + dp[i-2]*1

diagram for this:



dp[0]=1 says you can be at 0th position in 1 way

↳ Place 1×5  and 5×1  tiles across a floor of 5×N.



Do the following

(a) Identify overlapping problem

(b) DP state

(c) DP expression

(d) Base condition

(e) table size

(f) Ans??

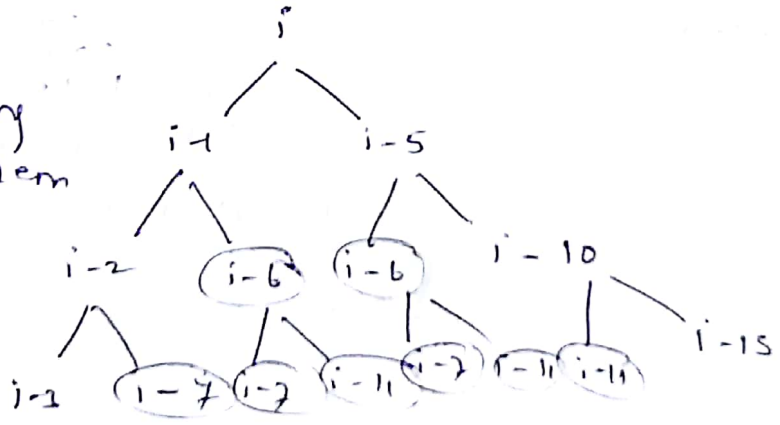
(g) T.C && S.C

(h) S.C??

Soln

→ (a)

overlapping sub-problem



(b) dp[i] → # ways to fill the floor of 5×i.

(c) $\sum_{i=5}^{N} dp[i-1] + dp[i-5]$

dp[0] = 1 // There exists only 1 way to fill a tile the floor of size 5X0.

dp[1] = 1

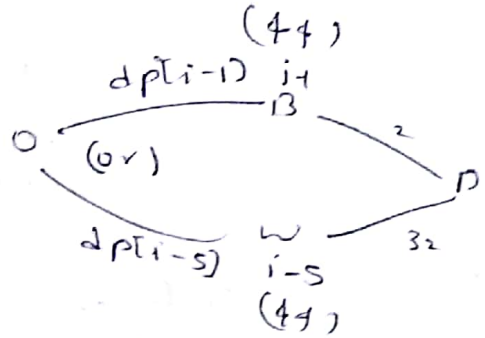
dp[2] = 1 // 1,2,3,4 indicate the lengths. like 5x1, 5x2, 5x3, 5x4.

dp[3] = 1

dp[4] = 1

Consider a plate which contains two colors black and white.

$$dp[i] = 2\,dp[i-1] + 32\,dp[i-5]$$



dp[0] = 1

dp[1] = 2 // 5x1 we can place a plate in 2 ways either turning towards Black or White. B or W

dp[2] = 4

dp[3] = 8

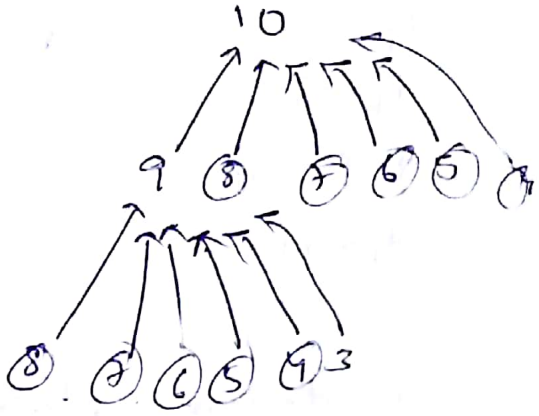
dp[4] = 16 // 5x2 we can place two plates in BB, BW, WB, WW 4 ways.

dp[5] = 32

- Given a 6 sided dice find the no. of ways, to get a sum of N when a 6 sided dice is rolled.

Say N = 10 ⟶ 6, 4
5, 5
4, 6
1, 1, 4, 2, 2
1, 3, 4, 2.

(a) overlapping sub-problem



(b) dp[i] → # dpstate

ways to get sum of i.

(c) $\sum_{j=1}^{6} dp(i-j)$ # dp expression.

$$\bigvee_{i=1}^{N} dp[i] = \bigvee_{j=1}^{6} dp[i-j]$$

(d) Base condition dp[0]=1, dp[1]=1, dp[2]=2, dp[3]=4,

dp[4] = 1+1+2+4

dp[5] = dp[4] + 1+1+2+4.

```
code :- dp[0]=1
        i=1 to N
        dp[i]=0
        j=1 to 6
        if (i-j ≥ 0)
        dp[i]=dp[i]+dp[i-j];
        return dp[N]
```

T.C → N×6, N

(or)

if N sided dice is given

T.C will be N×N, N

Table size: arr[N]

Ans: dp[N].

Week-1	Day1	Dynamic programming
	Day2	Dynamic programming
Week-2	Day1	DP Graph theory
	Day2	Graph theory

Week-11
Day-1
26/1/2019

DYNAMIC PROGRAMMING (CONTINUATION)

Q) You have to form 'n' bit numbers those should not have adjacent 1's.

i/p: N=2

total possible number of combinations

00
01
10
11 *

} ans = 3

o/p: - 3

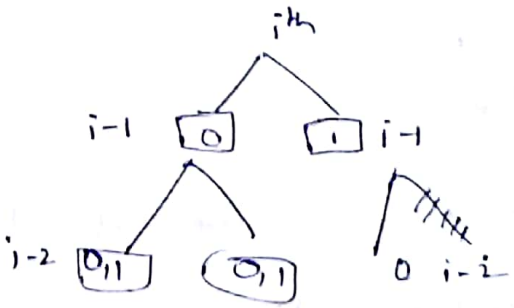
i/p: N = 3

000
001
010
011 *
100
101
110 *
111 *

ans = 5

o/p:- 5

Soln) ① #overlapping sub-problem :



② #dp state :

dp[i] = binary string of length i ending with 0

③ #dp expression:

$$\forall_{i=2}^{N} \; dp0[i] = dp0[i-1] + dp1[i-1]$$

↓ binary string of length i which ends with 0

↓ binary string of length i which ends with 1

$$\forall_{i=2}^{N} \; dp1[i] = dp0[i-1]$$

④ #Base conditions:

$dp0[1] = 1$

$dp1[1] = 1$

⑤ #ans: $dp0[N] + dp1[N]$

⑥ #table size: $N+1$, $N+1$

⑦ SC: 2^n.

How to get optimised space

$x' = x+y$

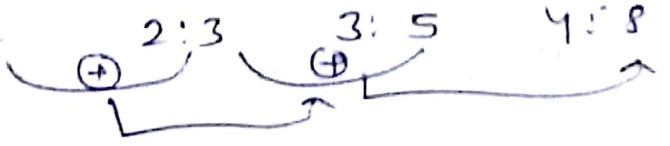
$y = x$

$x = x'$

$y = y'$

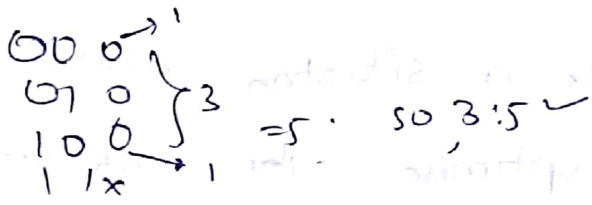
→ You can even solve it by getting the nth fibonacci number.

1 : 2 2 : 3 3 : 5 4 : 8



i/p: 2 then find the fibonacci of 2 you will get 3.

Reason for fibonacci is you will take all the combination without adjacent 1's and add 0 they will satisfy the condition i-e no adjacent 1's and even taking the combinations of 1's when there are 0's in adjacent we will get the count similer to fibonacci



(9) → Given a 6 sided dice. Find the minimum cost to get a sum of N

6 sided dice

number on dice ← $C_1 = 5$ → cost

$C_2 = 6$

$C_3 = 2$

$C_4 = 3$

$C_5 = 10$

$C_6 = 4$

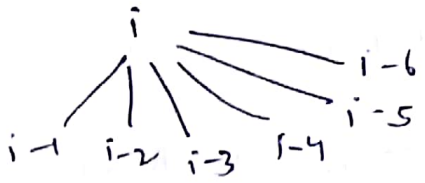
N=10

N 0:- $C_3 + C_3 + C_4 = 10$

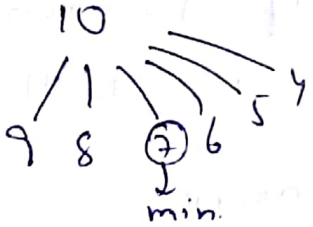
Cost:- $2 + 2 + 3 = 7$

Soln)

① # Overlapping sub-problem



this shows to reach ith state you can go from i-1, i-2, i-3, i-4, i-5 or i-6.



→ Here we can see a situation where we have to optimise after each step

② # Optimal sub-structure: only an optimal sub-problem will reach the actual solution.

→ You have to take the least cost.

③ # dp state: dp[i] is minm cost to get the sum = i

$$\min(dp(i-1)+C(1),\ dp(i-2)+C(2)+dp(i-3 \ldots dp(i-6)+C$$

dp + rp: dp(0) = 0

④ $$\forall_{i=1}^{n} dp[i] = \forall_{i=1}^{N} \min_{j=1}^{6} (dp(i-j)+C(j))$$

ans = dp[N];

⑤ #Table size: dp[N+1]

⑥ TC :- N

⑦ SC :- N

Note:
If the problem has minimum or max. type of things use optimal sub-structure.

⑨

→

N =	1	2	3	4	5	6
R:	5	3	5	1	4	13
G:	2	7	15	1	10	18
B:	10	6	18	1	2	9

N = 6 houses.

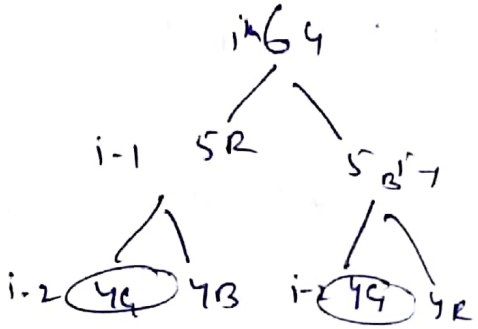
B[i]: Cost of painting ith house with B color.

Find the minimum cost to print n houses.

Two adjacent houses should not have the same color.

Soln> Greedy soln. does not work here.

① #overlapping subproblem:



② #dp state

dpR(i) = minimum cost to paint i houses such that ith house has Red color.

dp G(i) = min. cost to paint i houses such that ith house has Green color.

dp B(i) = min. cost to paint i houses s.t ith house has Blue color.

③ #dp expression:

$\forall_{i=1}^{N}$ dpR(i) = min(dpG(i-1), dpB(i-1)) + R(i)

$\forall_{i=1}^{N}$ dpG(i) = min(dpR(i-1), dpB(i-1)) + G(i)

$\forall_{i=1}^{N}$ dpB(i) = min(dpG(i-1), dpR(i-1)) + B(i)

④ #Base condition:

dpR(0) = 0 dpG(0) = 0 dpB(0) = 0

dpR(1) = R(1) dpG(1) = G(1) dpB(1) = B(1)

⑤ #ans: min (dpR(N), dpG(N), dpB(N))

⑥ #TC : N

S.C : N

to optimise the space :-

$r' = \min(g, b) + R(i)$

$b' = \min(r, g) + B(i)$

$g' = \min(b, r) + g(i)$

⑦

→

Job_N :	1	2	3	4	5	6
A_N :	5	3	15	10	18	6
B_N :	10	8	1	2	5	1

$K = 15$

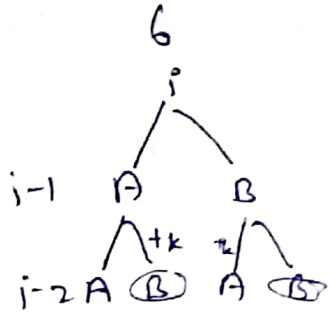
A(i) denotes time taken to execute ith job in machine A.

We have to execute all the jobs in a minimum time.

We also have to consider the cost is added when shifted from one machine to another.

Sol>

① # Overlapping sub-problem



② # dp state :

dpA(i) → minm time to complete ith job on machine A.

dpB(i) → minm time to complete ith job on machine B.

③ # dp expression:

dpA[i] = min (dpA[i-1], k+dpB[i]) + A(i)

dpB(i) = min (dpB[i], k+dpA(i)) + B(i)

④ # Base condition :

dpA[0] = 0 dpB[0] = 0

dpA[1] = A[1] dpB[1] = B[1]

→ ⑤ Given an array of size N find the maximum sub-array sum.

arr: 5 2 -10 2 7 -3 10 6 -1 8

-43 10 2 -1 -15 5

Sol)

1) N^3, 1 Brut force code taking 3 for loops & finding max sub-array sum

2) $N^2, 1$ → using carry forward

3) $N, 1$ → kadane's algorithm for finding max sub-array sum
(1-D)

Note: (2-D) approach will give the sub-matrix.

Using dynamic programming:

① #Overlapping sub-problem  . . . .

② # dp state

dp[i] → maximum sub array sum ending at i and including i

③ # dp expression:

$$\forall_{i=1}^{N-1} \; dp[i] = \max[dp[i-1], 0) + arr(i)$$

④ # Base condition :-

$$dp[0] = arr[0]$$

⑤ $ans := \max_{i=0}^{N-1} (dp(i))$

⑥ S.C → int dp[N]

T.C → N

Space optimization

```
{
  x = max(x, 0) + arr[i];
  ans = max(ans, x);
}
```

Base :- x = arr[0], ans = arr[0].

(9)

→ Find the maximum non-adjacent subsequence sum.

arr: 5 2 -10 2 7 -3 10 6 +15 8 -43 10 2 -1

if you take 2 you can't take -10 & 7.

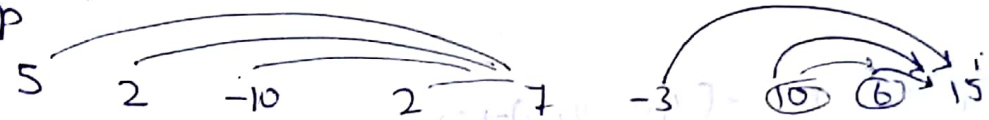
Soln → 1) $2^n \times N, 1$ → iterative $2^n \times N, 1$
→ recursive $2^n, 1$

2) N^2, N ~~using carry forward.~~

3) $N, N \not\otimes 1$

② implementation:

①# ovsp



to reach i we have to go from i-2 to 10 from 7

②# dpstate:

dp[i] = maximum non adjacent subsequence sum till i ending at i and including i.

③ #dp expression:

$$\forall_{i=2}^{N-1} dp[i] = \max_{j=0}^{i-2} (dp[j]) + arr[i])$$

④ Base Conditions :

$dp[0] = arr[0]$

$dp[1] = arr[1]$

⑤ ans: $\max_{i=0}^{N-1}(dp[i])$

⑥ table size: int dp[N]

⑦ T.C : N^2

Space can't be optimised as

3) N, N-1

③ # dp expression

$\forall_{i=2}^{N-1}$ {

$m = m + arr[i]$

$m = \max(m, dp[i-1])$

}

④ Base conditions :

$dp[0] = arr[0]$

$dp[1] = arr[1]$

$m = arr[0]$

Another way with O(n)

$dp[i]$ = maxm subsequence sum till i ending at i not including i.

$\forall_{i=2}^{N-1} dp[i] = \max(dp[i-2] + arr[i], dp[i-1], arr[i])$;

ar_N:	5	2	-10	2	7	-3	10	6	15
dp:	5	5	5	7	12				

Base conditions : dp[0] = ar[0]

dp[1] = max(ar[0], ar[1])

ans : dp[N-1]

T.C : N

S.C : N

Space reduction.

c = max(ar[i] + a), b, ar[i])

a = b

b = c

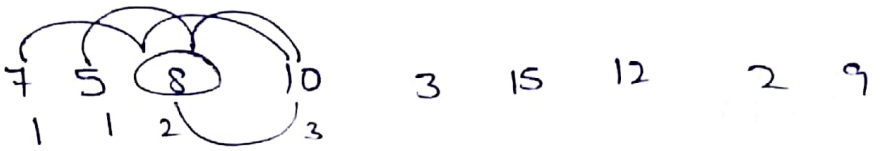
Q) Given an array find the length of longest increasing sub-sequence.

ar_N: 7 5 8 10 3 15 12 2 9 4 6 18 5 9 13

Sol) ① $2^n \times n$, 1

2) Using dynamic programming

(a) # Overlapping sub-problem:



to reach 10

7, 8, 10 } 3
5, 8, 10 }
8, 10 } 2

We take optimal solution so we are using optimal substructure

(b) # dp state

dp[i] → length of longest increasing subsequence till ith location.

(c) # dp expression:- dp[i] = length (LIS) till i including i

if (arr[j] < arr[i]) then

$$dp[i] = \max_{j=0}^{i-1} (dp(j)) + 1 \quad \text{for } i = 0 \text{ to } N+1$$

↳ 1 is for including the ith element.

↳ max will calculate excluding the ith element.

(d) Ans: $\max_{i=0}^{N+1} dp(i)$

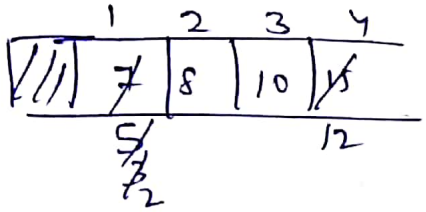
(e) table: N

(f) T.C: N2

S.C: N

We cannot optimise the space.

③ Hint: NlogN, N



Apply binary search put elements in an array finding ceil and putting all the elements will take NlogN.

8

arN: 7 5 8 10 3 15 12 2 9 4 6 1 5 9

dpN: 1 1 2 3 3 1 4 4 1 3 2 3 5 3 4

print the elements from longest subsequence backtrack from nth element to print.

9

arN: 1 8 5 12 17 4 10 6 25 100 2

Find the no. of subsequence with sum=odd

1+5+17 = 23

Sol

1) 2^n

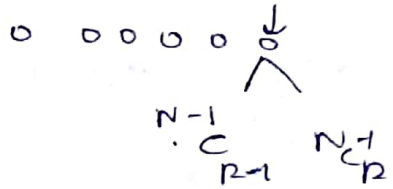
2) use dynamic programming

Figure out.

Week-11
Day-2
27/11/19

$$^{N}C_{R} = \frac{N!}{(N-R)!\,R!} = {}^{N-1}C_{R-1} + {}^{N-1}C_{R}$$

no. of ways to select R items from N items



$$\frac{(N-1)!}{(N-1-(R-1))!\,(R-1)!} + \frac{(N-1)!}{(N-1-R)!\,R!}$$

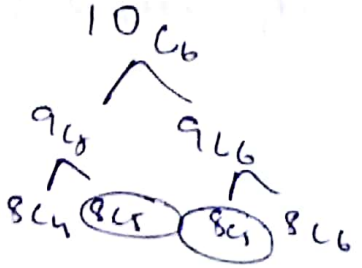
$$\frac{(N-1)!}{(N-R)!\,(R-1)!} + \frac{(N-1)!}{(N-R-1)!\,R!}$$

$$(N-1)!\left[\frac{R}{(N-R)!\,(R+1)!\times R} + \frac{1}{(N-R-1)!\,R(R-1)!}\right]$$

$$(N-1)!\;\frac{N+R-R}{(N-R-1)!\,(N-R)(R-1)!\,R}$$

$$= \frac{N!}{R!\,(N-R)!}$$

① Overlapping sub-problem:

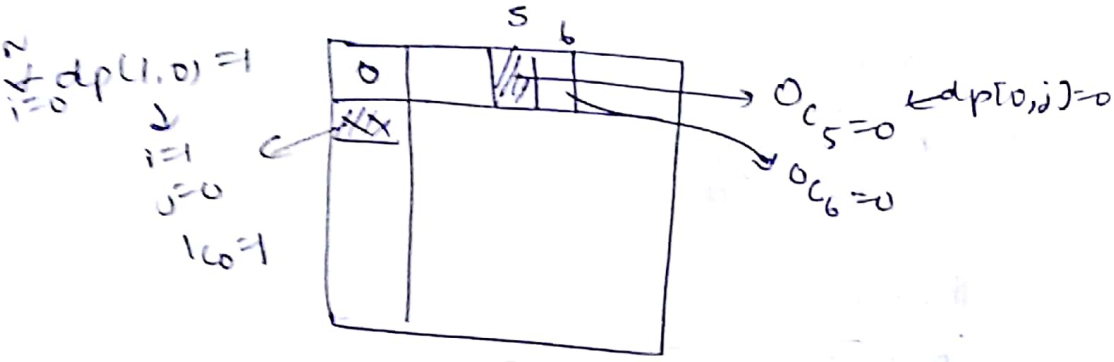


② #dp state :

dp(i,j) → ways to select j items from i items

③ #dp expression :

$$\forall_{i=1}^{N} \forall_{j=1}^{R} \; dp(i,j) = dp(i-1, j-1) + dp(i-1, j)$$



④ #Base conditions:

$$\forall_{j=1}^{R} \; dp[0,j) = 0$$

$$\forall_{i=0}^{N} \; dp(i,0) = 1$$

⑤ # T.C : N^2

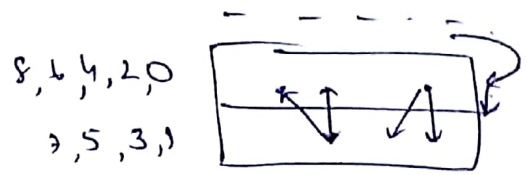
S.C : N+1, R+1

⑥ #Ans. dp[N][R]

⑦ #table space : int dp[N+1][R+1]

Space can be optimised

$dp(i\%2, j) = dp((i-1)\%2, j-1) + dp((i+1)\%2, j)$



by doing %2 we are trying to get the row below or above one.

→ so as we do not need all the values, we only consider on $(i-1, j-1)^{th}$ values and $i-1, j$ values.

iterative code:

```
for(j=1 ; j<R; j++)
  dp[0][j] = 0;
  for(i=0; i<n; i++)
     dp[i][0] = 1

  for( i=1 ; i<=n; i++)
  {
    for(j=1;j<=r;j++)
      {
        dp[i][j] = (dp[i-1][j-1]+dp[i-1][j]) % m;
      }
  }
  return dp[N][R];
```

Recursive:

```
int ncr(int n, int r)
{
  if (r == 0)
    return 1;
  if (n == 0)
    return 0;
  if (dp[n][r] == -1)
    dp[n][r] = ncr(n-1, r-1) + ncr(n-1, r);
  return dp[n][r];
}
```

(9) KNAPSACK

Items: 1 2 3

V_N : 100 60 120 (values)

W_N : 20 10 30 (weights)

K = 50
↳ capacity

Choosen items weight should not go beyond K and value should be max.

knapsack → (GREEDY) Fractional : $R = V/W$ → 2:10 → 60, 1:20 → 100, 3:20 → 80 = 240

knapsack → Integer

Fractional

Create ratio array & sort it

```
while(i<n && k>0)
{
    ans = min(k, wt[i]) * R[i]
    k = k - min(k, wt[i]);
    i++;
}
```

→ Integer: → 0-1 : 220 ← (30+20)

→ Infinite: 300 pick 10 5 items & profit would be 300.

↓

You can pick a weight infinite no. of times.

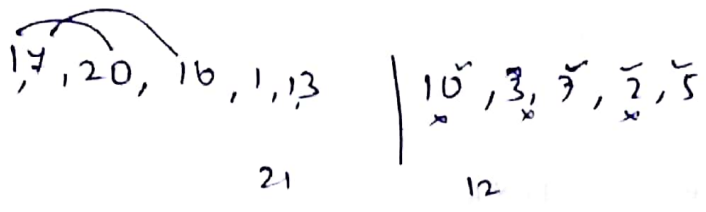
→ Greedy will not work for integer knapsack because the profit you choose will be 150 with weight 50 but the actual ans will be 220.

→ Greedy does not work for infinite knapsack

1) $2^n \times n$,)

2) Using dynamic programming.

(a) #Overlapping sub-problem:



So overlapping sub-problem.

(b) # dp state:

dp[i][j] → choosing items from i to j with max. knapsack size of j.

(c) # dp expression

$$dp[i][j] = \max(dp[i-1][j-wt[i]] + val[i],\ dp[i-1][j])$$

$\forall_{i=1}^{N}$ $\forall_{j=0}^{k}$ if $(j - wt[i] > 0)$ $j = k$

- $dp[i-1][j-wt[i]] + val[i]$ → taking an item (value + remaining weight)
- $dp[i-1][j]$ → not taking an item

$dp(3,20) = \max(dp(2,-10)+120,\ dp(2,20))$

So to avoid negative we put the condition $j - wt[i]$.

(d) #ans : dp[N][K]

(e) table size :- dp[N+1][K+1]

(f) T.C : N x K

S.C : N x K

To reduce space :-

dp[i][j] = max(dp[(i-1)%2, j-wt[i]], dp[(i-1)%2, j]) + v[i]

Note: If you consider the index from 0

dp expression : $\forall_{i=0}^{N-1}$ dp[i][j] = max(dp[i+1, j-wt[i]), dp[i+1, j]) + v[i]

Base condition : $\forall_{j=0}^{wt[0]-1}$ dp(0,j) = 0 $\quad \forall_{j=wt[0]}^{K}$ dp(0,j) = v[0]

For infinite knapsack as you can take one item any no. of times so we do not decrease its value.

dp expression : dp[i][j] = max(dp[i, j-wt[i]] + v[i], dp[i-1, j])

Base conditions for ~~infinite~~ 0 (~~≠~~) $\forall_{j=0}^{wt[0]-1}$ dp(0,j) = 0

$\forall_{j=wt[0]}^{K}$ dp(0,j) = v[0]

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	12	12	12	12	12
2	0	0	0	12	12	12	20	20
3	0	0	0	12	12	15	20	20
4	0	0	6	12	12	18	20	21
5	0	6	6	12	12	18	20	22

→ dp(i-1, b-w(i)), dp(i-1, b)

dp(0,3), dp(0,6)

→ max(dp(i-1, 7-w[i]), dp(0,3))

= 0

→ dp[i-1][3-w[i]] + v[i], dp[i-1][3]

12 0

= 12

(6)

→ COIN-CHANGE

arr: 5 1 8 2 15 18 7 25

K=30

If there exists a subset which would return sum of 30 return true

else return false.

0/1 Knapsack type

Soln> 1) 2^n, 1

2) using dynamic prog.

dp state:

dp(i,j) → can we get a sum of j using 0 to i elements.

dp expression: $\forall_{i=1}^{n} \forall_{j=1}^{s}$ dp(i,j) = dp(i-1, j-arr[i]) || dp(i-1, j)

Base condition: $\forall_{j=0}^{k}$ dp(j,0) = F dp(0, arr[i]) = T

dp(0,0) = T

T.C → N x k

S.C → N x k

(Q)

→ Total number of ways to get sum.

dp expression

dp(i,j) = dp(i-1, j-arr[i]) + dp(i-1, j)

Base condition: dp(0,j) = 0 dp(0, arr[i]) = 1

dp(0,0) = 1

(Q)

→ 0-1: min. no. of elements to get sum = k k = 30

5 1 8 2 15 18 7 25

25+5

2 elements.

① # dpstate: min. no. of elements required to get a sum of j using 0 to i elements.

② # dpexpression:

$$dp(i,j) = \min(dp(i-1, j-arr[i]+1), dp(i-1,j))$$

③ # Base conditions: $dp(0,0)=0$

$dp(0, arr(i))=1$

$\forall_{j=1}^{k} dp(0,j) = \infty$ Intmax

ans: $dp[N-1][K]$

if $dp[N-1][K] = 10^9$ then ans does not exist or else ans exists.

	0	1	2	3	4	5	6	7	8	...	30
0	0	∞	∞	∞	∞	1	∞	∞	∞	...	—
1	0	1	∞	∞	∞	1	2				

⑨

→ Given array of elements using elements of the array sum of some of the elements on the left should be equal to remaining elements.

sum(1) == sum(2)

#dp state:

dp[i] represents the elements to get sum=k

#dp expression:

```
if (S%2 == 1)
  return false
else
  k = S/2 (previous problem)
```

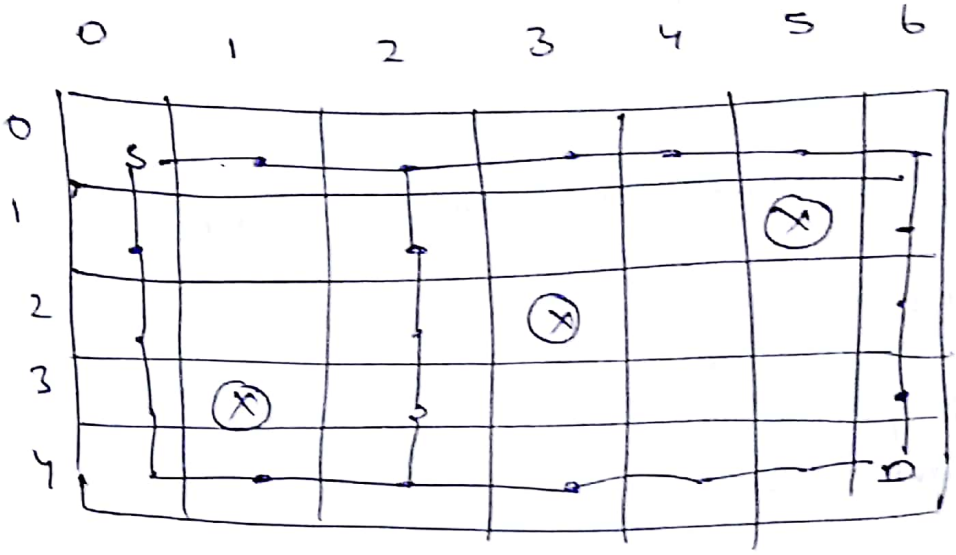
(9) → Find set of numbers S1 & S2 such that sum(S1) − sum(S2) is minimum

Check k = S/2

take array of size N, k+1

then from 40 find true from end.

(9) → Given a 2D matrix with certain points blocked find the no. of ways you can start from 0,0 and reach N-1, M-1



i/p:- N M B-Jy (Blocked positions)

[: : :]

dp state: dp(i,j) → # ways to reach i,j from

dp expression:

$$\forall_{i=1}^{N+1} \forall_{j=1}^{M+1} dp(i,j) = dp(i-1,j) + dp(i,j-1)$$

ans:

dp[N-1][M-1]

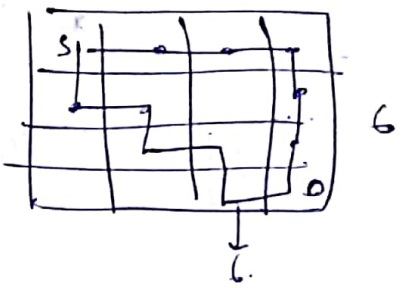
base conditions: if (i,j is blocked)
put 0.

initialise all the blocked points as -1

(8)

→ Find the no of steps to reach a particular point i,j

Give a formula



→ So no matter the way you travel it will take the same amount of time to reach the destination.

Steps :- N-1+M-1

N+M-2

For any matrix if we go from source to destination each path will take N+M-2 steps.

(9)

→ There are 10 candies and no. of ways you can divide them among 2 friend,

$^{N+M-2}C_{M-1}$ or $^{N+M-2}C_{N-1}$

N+M-2

→ N-1 (Friend 1)

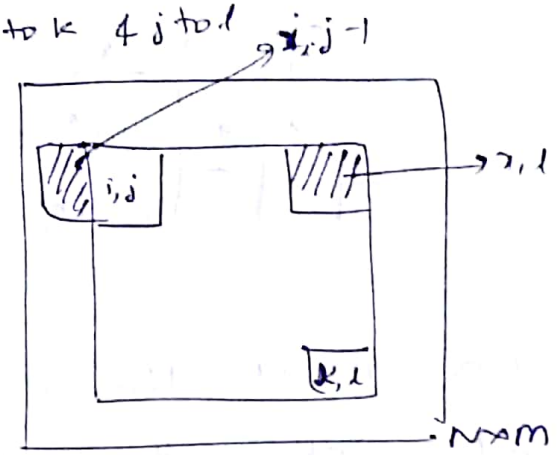
→ M-1 (Friend 2)

So you can share candies with 2 of your friend in N+M-2 ways.

Week-12
Day-1

Dynamic Programming

→ Find the sum of the elements in the sub-matrix from i to k & j to l



i, j, k, l

$0 \le i \le k \le n$

$0 \le j \le l \le m$

$$\sum_{x=i}^{k} \sum_{y=j}^{l} mat[x][y]$$

Soln)

1) Q×N×m, (brute force code)

2) N×m + Q×N, ~~N×m~~, → we can optimise it by storing in the same matrix.

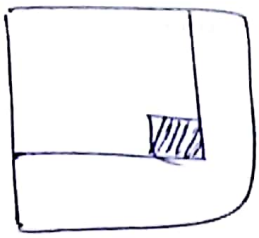
$ps(i,0) = mat(i,0)$

$ps(i,j) = ps(i,j-1) + mat(i,j)$

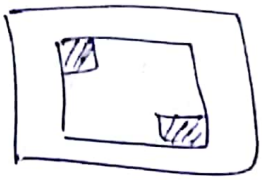
$$ans = \sum_{x=i}^{k} ps(x,l) - ps(x,j-1)$$ $j \neq 0$

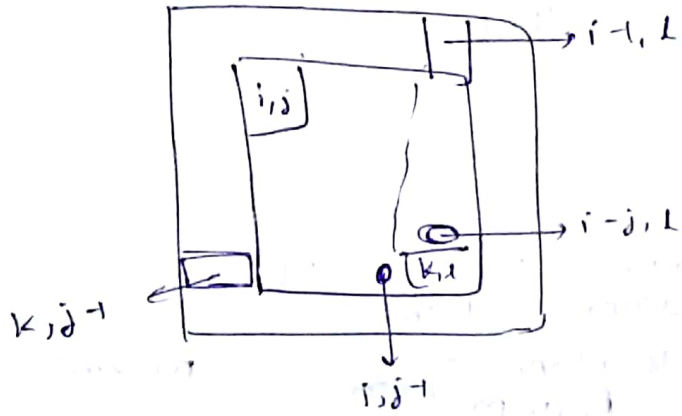
3) N×m + Q×1, ~~N×m~~,

Suppose if a cell can store the sum of all the previous cells



But we need from i,j to k,l





ans :- $ans = dp(k,l) - dp(k,j-1) - dp(i-1,l) + dp(i-1,j-1)$

dp state: $dp(i,j) \rightarrow$ Sum (0,0) to (i,j)

dp expression: $\forall_{i=1}^{N} \forall_{j=1}^{m} dp(i,j) = dp(i-1,j)_{i\neq 0} + dp(i,j-1)_{j\neq 0} - dp(i-1,j-1) + mat(i,j)$

Base: $dp(0,0) = mat(0,0)$

$\forall_{j=1}^{m} dp(0,j) = dp(0,j-1) + mat(0,j)$

$\forall_{i=1}^{N} dp(i,0) = dp(i-1,0) + mat(i,0)$

table size: N+1 × M+1

We shall convert i,j,k,l to i+1, j+1, k+1, l+1 after making the given matrix we will ans this for 1 based index

Q → Find the maximum sub matrix sum.

15	3	-53	25	-2	-4
6	-15	10	-1	-110	5
-1	8	3	12	10	2
10	-4	-25	8	-30	7
-5	1	5	4	4	20

Soln) 1) N^3m^3,

```
i=0 to N   }N×m
  j=0 to m }
    k=i to N   }N×m
      l=j to m }
        x=i to k   }N×m
          y=i to l }
```
} $N^3 \times m^3$

2) $N \times m + N^2 m^2$,

Step ①:- Compute dp matrix $n \times m$

Step ②:- iterate through all matrice, $N^2 \times m^2$

step ③:- Each each find sum $O(1)$
space $O(1)$

3) $N \times m + N \times N \times m$,

2D Kadane's algorithm for finding max sub-matrix sum.

$psl(0,j) = mat(0,j)$

ans: $\sum_{i=1}^{N-1} \sum_{j=0}^{M-1} psl(i,j) = psl(i-1,j) + mat(i,j)$

```
for i=0 to N
    for k=i to N
        s=0
            for j=0 to m  if (i!=0)
                s=s+psl(k,j)-psl(i-1,j)
            ans= max(ans,s)
              if (s<0)
              s=0
```

(9) → 2-player game.

1 5 2 10 3 15 4 20

A → 1 (or) 20

B → 5 20 1 4

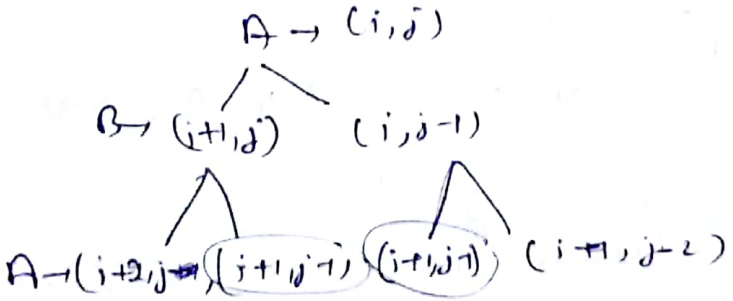
There are 2 players and each of them can pick from any end of the array 1 or 20

the person who picks up max. score will win.

Both play optimally.

Sol

DP



#dpstate:- dp(i,j) → max score a player can get from sub-array i to j

#dp expression:

$$dp(i,j) = \max\begin{cases} ar[i] + \min(dp(i+2,j), dp(i+1,j-1)) \\ ar[j] + \min(dp(i+1,j-1), dp(i,j-2)) \end{cases}$$

↓ every player at point i,j will try to max their score

↓ the other player will try to min. the score of other player

→ we then even add to total ar[i,j] score

#Base condition:

if only one element

dp(i,i) = ar[i]

dp(i,i+1) = max(ar[i], ar[i+1])

```
f(int i, int j){
    if(i==j)
        return ar[i,j]; // if only one ele
    if(i==i+1)
        return max(ar[i], ar[j]); //if only two ele
```

```
if (dp[i][j] == -1)
    dp[i][j] = max(ar[i] + min(f(i+2, j), f(i+1, j-1)),
                   ar[j] + min(f(i+1, j-1), f(i, j-2)));

int A = f(0, N-1)
    B = total sum - A
```

(5) → Given three matrices multiply them such that there are min. no. of iterations.

to reach this you have to perform 10 operations.

Op = 5*3*2 + 2*3*8 = 78

Op = 5*2*8 + 3*5*8 = 200

Depending upon the type ~~the~~ of multiplication the no. of operation will differ.

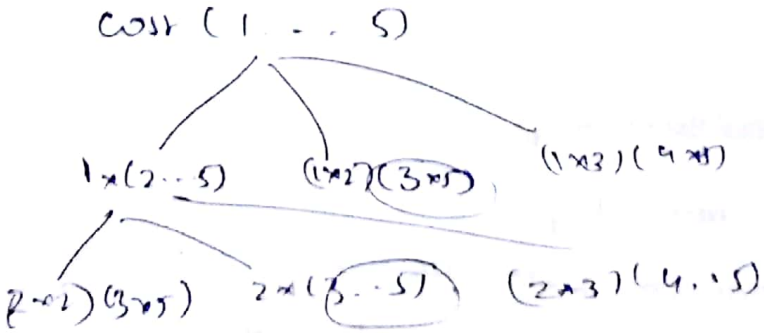
	0	1	2	3	4	5
ar	3	5	2	8	4	7

N=6

R_{3*7}

3*5 5*2 2*8 8*4 4*7

Result will be 3*7.



#dp state:

dp(i,j) → cost to multiply from i to j

#dp expr:

$$dp(i,j) = \min_{k=i}^{j-1} (dp(i,k), dp(k+1,j)) + arr[i] * arr[k+1] * arr[j+1]$$

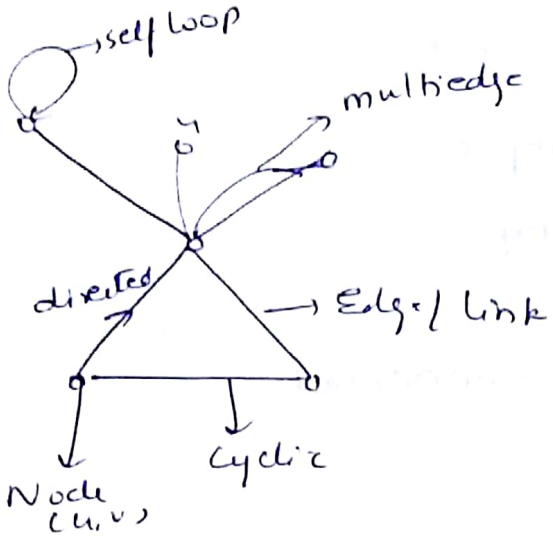
```
ans = f(0, N-1)

f(i, j)
  if (i == j)
    ret 0;

  ans = f(0, N-1);
```

T.C:- $O(N^3), N^2$

Problems on dp

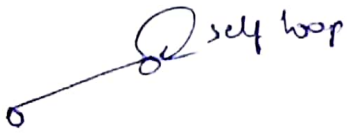
1. Rod cutting
2. Palindrome partitioning
3. Building bridges
4. Box stacking
5. Bitonic subsequences
6. LP substring
7. LP subsequence
8. EDIT Distance
9. LCS
10. Valid boolean exp
11. Max Arithmetic Exp
12. Max submatrix with all 1's
13. " Rect " " " 1's by Histogram
14. Max sequence without

Graph theory



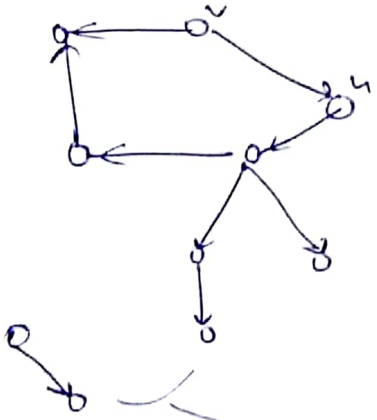
Path :- Traversing from one node to another

Length of path: no. of edges.



loop or multiedges → complex graph

Noloop " " → simple graph



Connected: weakly : undirected path b/w nodes
strongly :- There exists a direct path b/w nodes

Disconnected : no connection

Weighted vs unweighted

Directed vs Undirected

simple vs complex

Cyclic vs acyclic

Connected vs disconnected

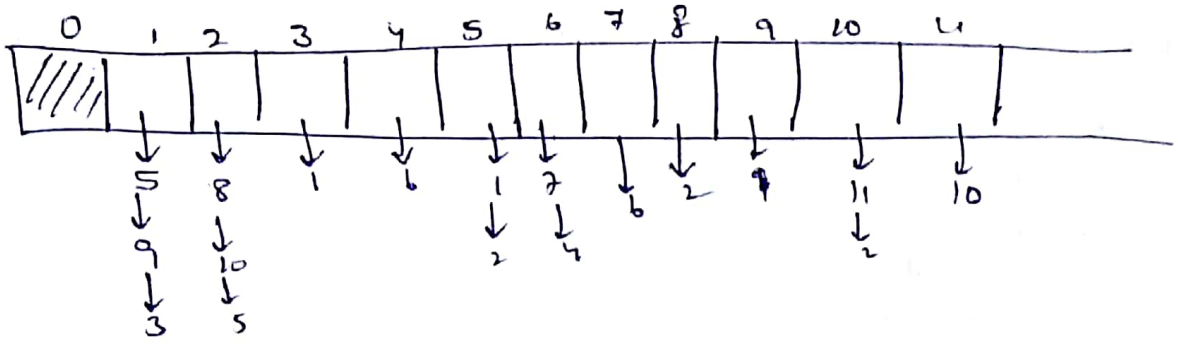
(8)

→ Given source node and destination check if there exists a path.

$G = (V, E)$

u(N)	v(m)
1	5
2	8
10	11
2	10
5	2
6	7
1	9
4	6
1	3
s	p

→ Store graphs using Adjacency matrix of adjacency list.



Traversing a graph

A) BFS

B) DFS

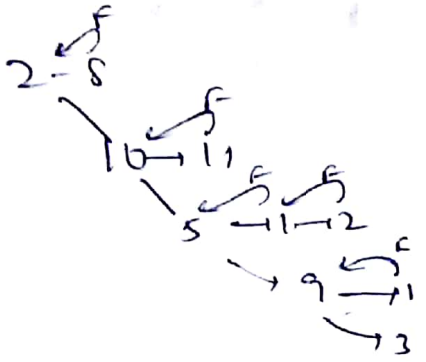
boolean vis[N+1]=F

BFS

```
bool BFS(ins, int d, ___ q){

    queue<Integer> q;
    N = G.size();
    boolean vis[N]={F};

    q.push(s);
    visit[s]=T;

    while(!q.empty()){
        int x= q.pop();
        for(int v: G[x]){
            if(vis[v]==F){
                q.push(v)
                vis[v]=T
            }
        }
    }
    ret
}
```

→ DFS

2 → 3




```
bool    DFS (int s, int d, ArrayList<ArrayList<Integ

{
  if(vis[s]==T)
     ret false;

  if(s==d)
     ret True;

  vis[s]=T;
  for(int v: G.get(s)){

     if(DFS(v,D,G,vis))
          ret true;
     }
     ret false;
  }
```

T.C → BFS → O(|V|+|E|)

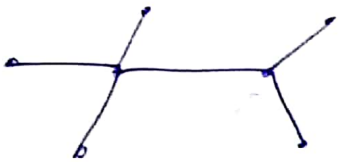
DFS → $\hat{N}$ $\hat{M}$

G(V,E)

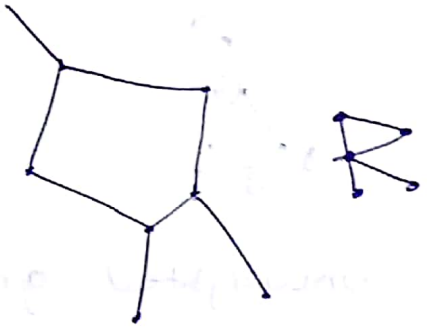
(9) Check if the given graph is a tree or not

(1)

(2)



(3)



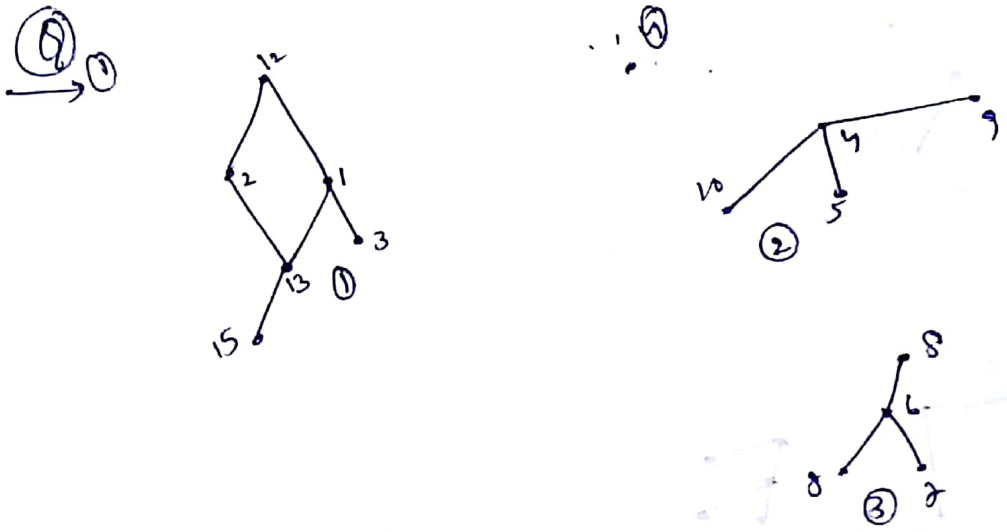
Conditions for a tree or not

a) connected

b) acyclic

```
bool conn() {
    DFS(1, G, vis);
    BFS(1, G, vis)
~
∀ i=1  if (vis[i] == F)
    { return false;
    }
  ret T;
}
```

```
bool acyclic(g)
{
  if (M == N-1)
     return true;
}
```



given an undirected unweighted graph find the no. of connected components.

```
#cc (connected components)
   c=0;
   bool vis[N+1]={F};
   for(int i=1; i<=N; i++)
   {
      if (vis[i]==F)
         c++;
         DFS(i, G, vis);
   }
```

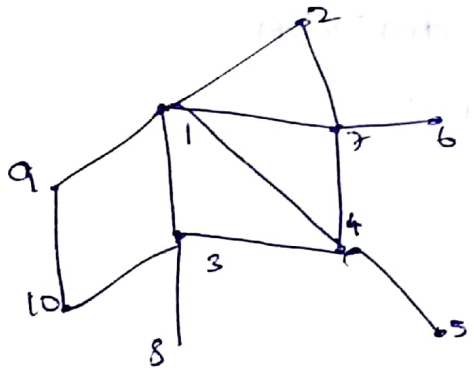
(6) (2)

Check whether the given graph is cyclic or acyclic.

Detect cycle

```
if(e/2==n-1)
    e=e+g[v].size();
```

(9)



1	9
2	4
3	8
1	7
.	.
)	)

→ Find the shortest path from the given source to the destination in an undirected unweighted graph.

S	D
1	5 → 3
9	6 → 3

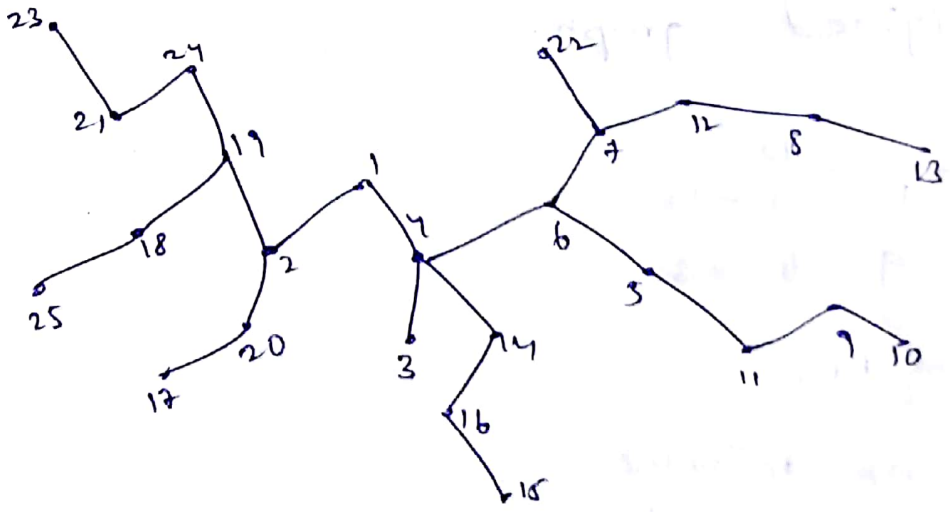
Using BFS:

```
int dis[N+1];
    dis[s]=0;
boolean visit[N+1]={f};
    q.push(s);
    visit[s]=T;
```

```
while ( !q.empty( )){
    int x = q.pop( );
    if ( x == d )
    return dist[v];
    for(int v: G[x]){
        if (vis[v] == F)
            q.push(v);
            dist[v] = dist[v]+1;
            vis[v] = T;
    }
}
```

(8)

→ Given an undirected acyclic graph find the length of the longest path.



O/P:- 23 → 10 → 11

23 - 13 → 11

→ from one node to all other

1) $|V|^2$ $(|V|+|E|)$

2) $|V|$ $(|V|+|E|)$ → BFS

→ for all node at once

3) $2(|V|+|E|)$

running BFS

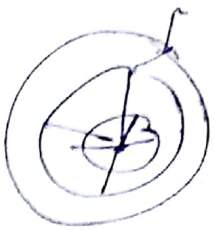
4 14 1 6 3 16 15 1 5 15 20 19 11 12

11 17 21 18 8 7 21 25 13 16 23

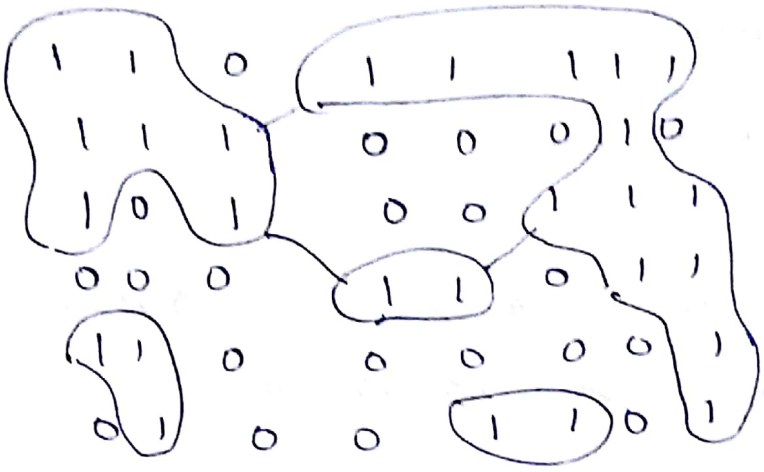
1 → 13
6 → 23
2 → 13
4 → 23
5 → 23

Why does it give the end points of longest path?

A) As we are doing running BFS we are doing a level order traversal & we will end up in last node, last node will be leaf node & even longest node.



Q)



0 → water
1 → land

Count the no. of islands,

N-4 neighbourhood / connectivity
N-8 connectivity

Ans: N-4 → 5
N-8 → 3

```
for i = 0 to N
for j = 0 to m
    if (mat(i,j) == 1)
        cnt++;
        DFS(i,j, mat, N, m);
```

Week-12
Day-2

In DFS mark element as 0 after visited.

↓

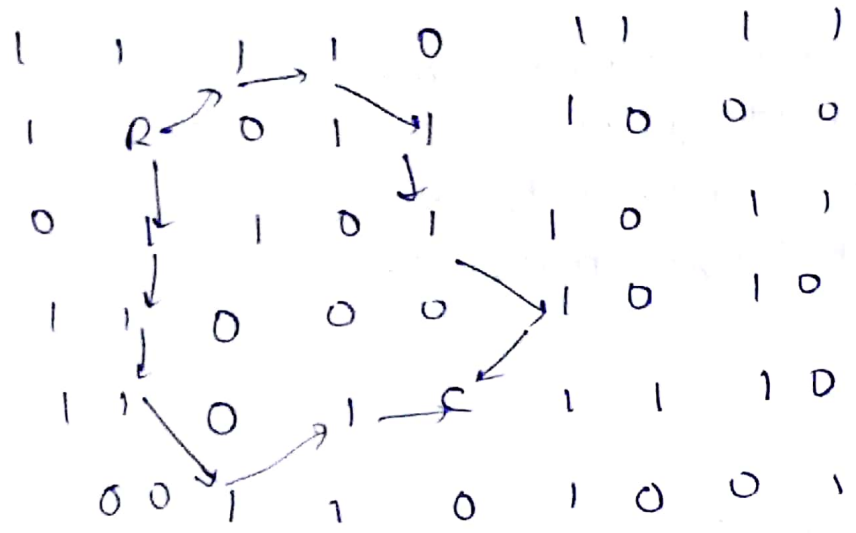
```
void DFS (int i, int j, mat[][], int n, int m)
{
    if (i<0 || i>=n || j<0 || j>=m || mat[i][j]==0)
        return;
    mat[i][j]=0;
    DFS(i-1, j, mat, n, m);
    DFS(i+1, j, mat, ---);
    DFS(i, j-1, mat, ---);
    DFS (i, j+1, mat, ---);
}
```

N-4

N-8

```
dx[8]= {0, 0, -1, 1, -1}
dy[8]= {-1, +1, 0, 0, +1}
c=0;
for i=1 to N
    for j=1 to m
        if (mat[i][j]==1)
            c++;
        DFS(i, j, mat, N, m);

void DFS (int i, int j)
{
    for (k=0; k<8; k++)
        DFS (i+idx[k], j+dy[k], mat, N, m);
}
```

9 → If there is a path b/w Rat(R) & Cheese(C), Find the length of the shortest path Print the path in a lexographical order.



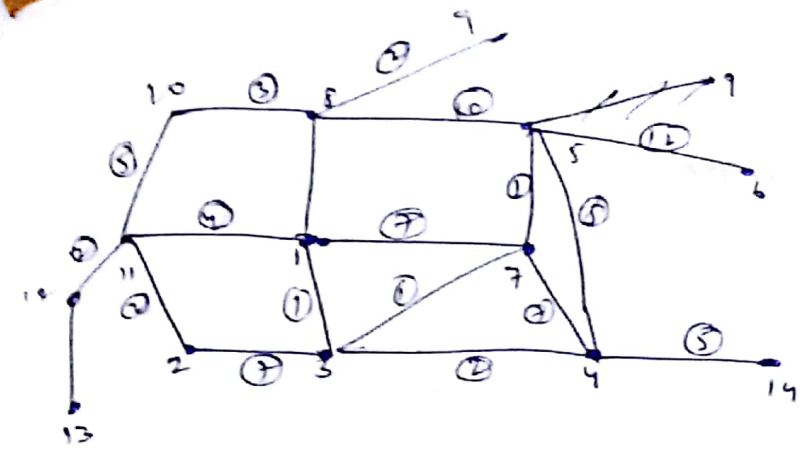
dx[8] = {-1, -1, -1, 0, 0}

dy[8] = {-1, 0, 1, -1, 1}

10 →

Given a source (S) & Destination (D), Find the shortest path from source to destination.

(S, 10 ⟶ (D, 4



1	2	3	4	5	6	7
∞_8	∞_7	∞_{14}	∞_{11}	∞_{13}	∞	∞_{10}

8	9	10	11	12	13	14
∞_3	∞_5	∞_0	∞_5	∞_7	∞_{17}	∞

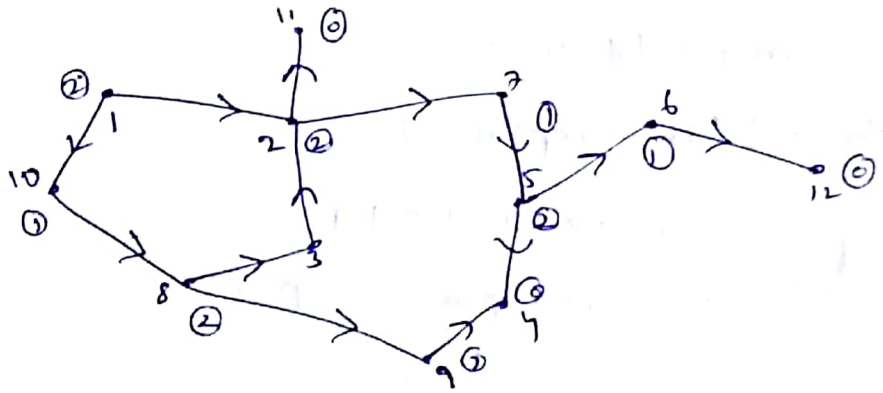
```
int Dijkstra(int s, int d, __ G)
{
  in n = G.sizel();
  PriorityQueue <Integer, Integer> mh;
      int dist[n+1] = {∞}, dist[s]=0
      mh.add(0,s);
  while (!mh.empty()){
      pair <int, int> p = mh.pop();
          int d = p.first ; u = p.second;
          if (u == d)
              return d; //dist[d]
```

```
if (d == dist[u])
for (pair<int,int> q: G.get(u))
{
    int w = q.second;
    int v = q.first;
    int nd = d+w;
        if (nd < dist[v]) {
            dist[v] = nd;
            mh.push(nd,v);
        }
    }
}
```

T.C :- $(|V| + |E|) \log |?|$

Shortest path
- → Unweighted (directed / undirected) → BFS
- → Weighted (directed / undirected) → Dijkstra

(5)



N = 12

There are 12 libraries

1 2 (1 depends on 2)

8 3 (8 depends on 3)

9 4

Indegree → # incoming edges

Outdegree → # outg.

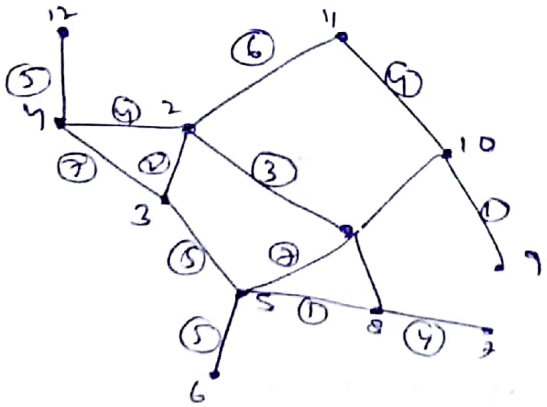
in & outdegree

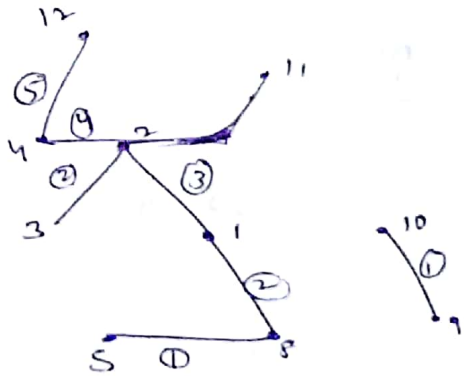
get outdegree = 0 T.C :- (V+E)

TOPOLOGICAL SORT, sorting the graph in a specific order.

└ Kahn's algorithm.

Krushkal's algorithm





N = 12

MST (Minimum spanning Tree)

a) Krushkal's (Greedy)

b) Prim's

Convert the given graph using Prim's or Kruskal's algorithm.

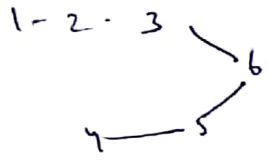
(9)

1) $E \log(E)$

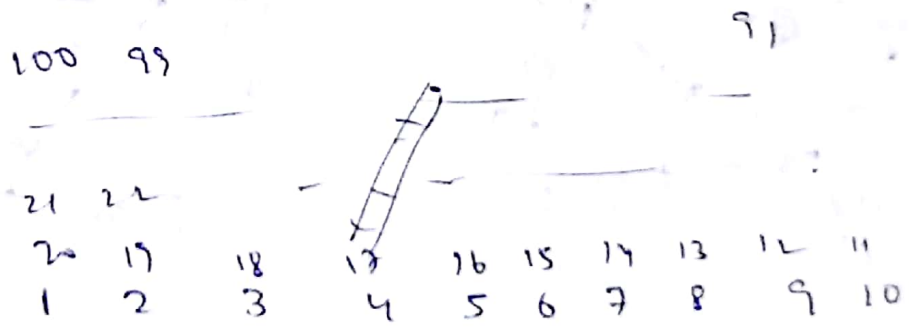
idx :	1	2	3	4	5	6	7	8	9	10	11	12
PT :	1	~~2~~ 1	~~3~~ 2	~~4~~ 1	~~5~~ 1	6	~~7~~ 1	~~8~~ 5	9	~~10~~ 9	~~11~~ 9	12

T.C: $E \log E + (E)(V+E)$

```
pu = f(u)
pv = f(v)
  pu = pv
    p[pv] = pu
```

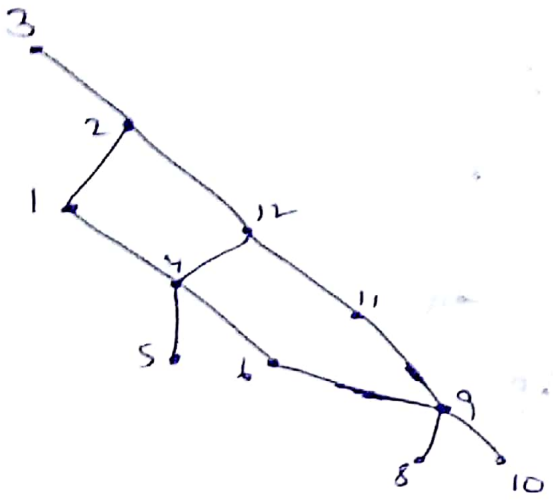


(9)

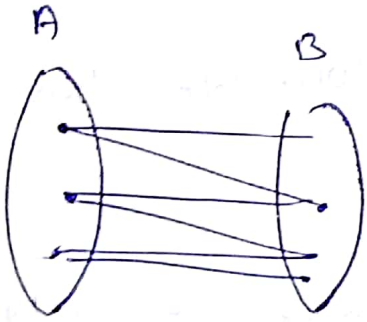


Directed unweighted graph use BFS

(9)



Classify the traders into trader A and trader B such that A-A cannot trade & B-B cannot trade. Only A-B & B-A can trade.



3 —— 2

1 —— 4

12 —— 11

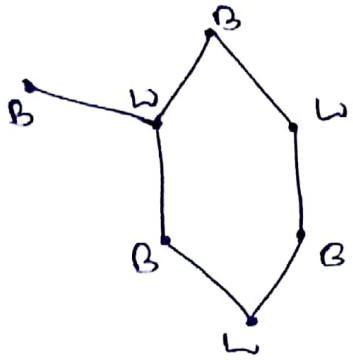
6 —— 7

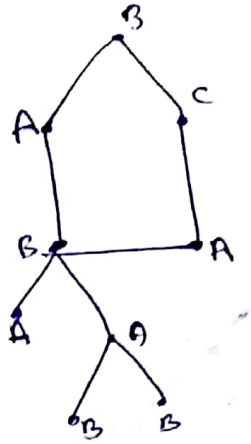
9

5 10

This is like Bipartite graph → 2-colourable.

⑨ Graph colouring





→ Given an ~~or~~ unordered unweighted graph Check if the graph contains cycle of odd length.

Soln) If the graph is 2 colourable then it is not odd length.

If a graph is 3 or more colourable then it is of odd length.

~~ans~~:

⑨ $\downarrow_{0 \le i \le j \le N}^{i\ j}$ (max (arr[i,j]))

Dijstra → SSP

1) Bellman ford (find sp for negative weights)

2) Floyd-warshall (APSP) (OOP-concepts)

3) Articulation points

44)

5) Max flow

Range Queries → segment trees ①
→ Binary Indexed trees ② $\log n$
→ sqrt decomposition ③ $\sqrt{n}$

— X — X — END X — X —