Date
25-05-19

S.No : 90

CLASS-3

Problem

ar_N 5 10 -1 6 -2

k = 12, 7, -4, 13

If Exists a subset of sum = k

sort the nos -1 -2 5 6 10 (K = 12)

$O(N^2)$ — ? Brute force

subsets given for an array of size $N = 2^N$

ar_N	5	10	-1	{ $2^N = 2^3 = 8$ subsets }
0	0	0	0	→ ∅ Null subset
1	0	0	1	→ -1
2	0	1	0	→ 10
3	0	1	1	→ 10, -1
4	1	0	0	→
5	1	0	1	→ 5, -1
6	1	1	0	→
7	1	1	1	→ 5, 10, -1
Bits	2^{nd}	1^{st}	0^{th}	
Index	0	1	2	

calculate different subsets for the given array

& add the elements & check if the sum = K

```
bool findN(int[] a, int K) {
    int length = ar.length();  // int N
    int noofsubsets = Math.pow(2, length);  // 1<<N
    for(int i=0; i<noofsubsets; i++)
    {   int index = 0;
        while(i!=0)
        {   if((i&1)==1)
            {   sum += ar[(N-1)-index];
            }
            index++;
            i = i>>1;
        }
        if(sum==K)
            return true;
    }
    return false;
}
```

Complexity: $2^N * N = O(N \cdot 2^N)$

Problem

```
for(i=0 to N) {
    for(j=0 to N) {
        ans = ans ^ (ar[i] + ar[j]);
    }
}
```

```
int ans = 0;
```

Ex: 1 2 10 5 3

i = 0
j = 0 ans =

i	j	ans
0	0	0 ^ (1+1) = 0 ^ 2 = 2
	1	2 ^ (1+2) = 2 ^ 3 = 1
	2	1 ^ (1+10) = 1 ^ 11 = 10
	3	10 ^ (1+5) = 10 ^ 6 = 10
	4	10 ^ (1+3) = 10 ^ 4 = 12
1	0	

```
010
011
001

1011
0001
1010

1010
0110
1100

1010
0100
1110
```

ar_N a b c

Combinations: $(a+a)^ * (a+b)^ * (a+c)^$
$(b+a)^ * (b+b)^ * (b+c)^$
$(c+a)^ * (c+b)^ * (c+c)^$

As $x^x = 0$, we are left with
$(a+a)^ (b+b)^ (c+c)^$

Complexity reduces from $O(N^2)$ to $O(N)$

Solutions

```
for(int i=0 to N){
    ans = ans ^ (2 * ar[i]);
}
```

Problem

$a^a + a^b + a^c$
$b^a + b^b + b^c$
$c^a + c^b + c^c$

$\Rightarrow 2(a^b) + 2(b^c) + 2(c^a)$

```
for(int i=0 to N)
    for(int j=i+1 to N)
        ans = ans + (ar[i] ^ ar[j]) * 2
```

Complexity: $O(N^2)$

Let say, $x_1 = ar[i] \wedge ar[j]$

$\Rightarrow$ ans $= x_1 + x_2 + x_3 + x_4 + \dots x_n$

$ar[i] \wedge ar[j] \Rightarrow 5 \wedge 10 \Rightarrow 15$.

Binary form = 15

$\Rightarrow 1111$

$\Rightarrow a*8 + b*4 + c*2 + d*1$

$\Rightarrow$ ans $= c_0*2^0 + c_1*2^1 + c_2*2^2 + c_3*2^3 + \dots$

Sub Problem Find the no of bits at each position

ans $= \sum a[i] \wedge a[j] \Rightarrow (a \wedge b) + (b \wedge c) + (c \wedge a)$

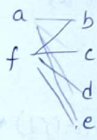
loop : 32 bits

if the count of 1's is odd ans = 1

```
for (i = 0 to 32)
{  while (a != 0 && b != 0)
   {  if ((a&1) != (b&1))
         count++;
```

Solution $\text{ans} = \sum_{i=0}^{31} Ci * (N - Ci) * 2^i * 2$

a b c d e

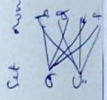


Lets say, there are 2 elements a & f which has 0^{th} bit set as 1 and other elements has 0^{th} bit set as 0

so, all the 8 combinations has 0^{th} bit set as 1.

$a \wedge b$

0^{th}
001
010
011



Calculate the number of elements in the array which has 0^{th} bit set and unset.

$$\sum_{i=0}^{30} Ci * (N - Ci) * 2^{i+1}$$

N = total number of elements

Solution

```
for (int i = 0 to 30)
{  int c = 0
   for (int j = 0 to N)
   {  if (checkBits(ar[j], i))
         c++;
   }
   ans += c * (N-c) * (1<<i) * 2
}
return ans;
```

- $i = 0$ to 30 if it is a signed int
- $i = 0$ to 31 if it is an unsigned int

RECURSION

Problem Triple Trouble

ar_N 1 3 4 3 1 1 3

001 011 100 011 001 001 011

Count the no of bits at each position

if (count % 3 != 0)

ans = (ans << 1) | 1 ;

Problem Repeated numbers except 1

ar_N 5 4 3 5 3 4 6

$x \wedge x = 0$ The remaining no is the result of XOR all elements

Problem Find the repeated no

ar_N 1 3 6 7 4 1 2 5

Range ar_N $1 \le ar[i] \le N$

Assume given i/p has 2 sets of repeated nos

Step 1 : XOR each element with [1 to 7]

$ar_N \wedge [1 \text{ to } 7]$

repeated nos xor = $3 \wedge 6 \Rightarrow 101$ (5)

Step 2 : Find the individual nos by dividing the nos a_N + [1 to 7] to 2 sets

A (set)	B (Unset)
4, 5, 6, 7	3, 2, 3, 1,
6, 4, 5, 6, 7	1, 2, 3

Step 3 : XOR all elements in each set.

A (set) = 6

B (Unset) = 3

* Sum of N numbers

$S(N) = 1+2+3+\cdots\cdots N$

$= S(N-1)+N$

```
int sum(int N)
{
  if (N==0)
    return 0;
  sum1 = sum(N-1) + N;
  return sum1
}
```

```
int fact(int N)
{
  if (N==1)
    return 1;
  res = fact(N-1) * N;
  return res;
}
```

```
int fibonacii(int N)
{
  // N = N elements of series
  // a_N = a_N-1 + a_N-2
  if (N==0) || if (N==1) return 1;
    return 1;
  sum = fibonacii(N-1) + fibonacii(N-2);
  return sum;
}
```

```
int APSum (int a, int d, int N) {
  // sum = 1+2+3+.....N
  // k=2 ⇒ sum = 1+3+5+... N+2
  // sum = a0 + ka1 + ka2 + .... kan ⇒ a, d, N a.p
        // a + n(n+1)/2 d
  if (N==0)
    return 0;
  sum = a + APSum (a+d, d, N-1);
}
  (OR)  return a+(N-1)d + APSum (a, d, N-1);
```

Recurrence Relation

* Sum of N Terms

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

$= T(N-2) + 2$

$\Rightarrow T(N-k) + k$

$\Rightarrow T(0) + N \Rightarrow O(N)$

* Factorial

$T(N) = O(N)$

* Fibonacii

$T(N) = T(N-1) + T(N-2) + 1 \Rightarrow 2T(N-2) + T(N-3)$

$\Rightarrow 2(T(N-2) + T(N-3)) + (T(N-3) + T(N-4))$

$\Rightarrow 2T(N-2) + 2T(N-3) + T(N-4) + 5$

$\Rightarrow N(T(0)) + N-1(T(1)) + \dots$

$T(N) = \sum_{i=2}^{N} k\, T(N-K) \quad K = N \; T(0) + N$

~~$\Rightarrow O(N^2)$~~

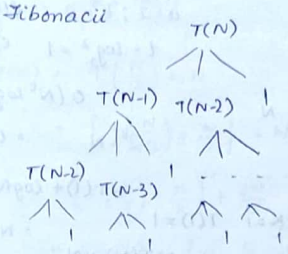
* APSum

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

$\Rightarrow T(N-K) + K$

$\Rightarrow O(N)$

* Fibonacii



At each level, no of operations = 2^i

$T(N) = 1 + 2 + 4 + 8 + 16 + \dots$

$= 2^0 + 2^1 + 2^2 + \dots 2^{N-1}$

$\Rightarrow \frac{1(2^N - 1)}{(2-1)} \Rightarrow (G.P)$

$\Rightarrow O(2^N)$

$T(1) = 1$

① $T(N) = 2T(N/2) + N$ = $O(N \log N)$

② $T(N) = 2T(N/2) + 1$ = $O(N)$

③ $T(N) = T\left(\frac{N}{2}\right) + N$ = $O(N)$

④ $T(N) = T\left(\frac{N}{2}\right) + 1$ = $O(\log N)$

⑤ $T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{4}\right) + N^2$ = $O($

① $T(N) = 2T\left(\frac{N}{2}\right) + N$ $\quad a=2 ; b=2 ; f(N)=N^c$

$\Rightarrow T\left(\frac{N}{2}\right) = 2T\left(\frac{N}{4}\right) + \frac{N}{2}$ $\quad t = \log_2^2 = 1 \quad c=1$

$= 2\left[2T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N$ $\quad O(N^t \log N)$

$= 4T\left(\frac{N}{4}\right) + 2N$ $\quad 2^k = N \quad = O(N \log N)$

$\Rightarrow 2^k T\left(\frac{N}{2^k}\right) + KN$ $\quad K=1 \quad T(1)=1$ $\quad NT(1) + \log N(N)$

$= T(N) + N \quad = N \log N + N$

$\Leftarrow K = N \quad T(N) = 2^N (T(1)) + N^2 \quad O(N \log N)$

$= NT(1) + N^2$

$= N^2 + N \Rightarrow 2^N$

② $T(N) = 2T\left[\frac{N}{2}\right] + 1$

$= 2\left[2T\left[\frac{N}{4}\right] + 1\right] + 1$ $\quad \Rightarrow 2^k T\left[\frac{N}{2^k}\right] + 2^k - 1$

$= 4T\left[\frac{N}{4}\right] + 3$ $\quad = KT\left[\frac{N}{8}\right] + (K-1)$

$= 4\left[2T\left[\frac{N}{8}\right] + 1\right] + 3$ $\quad K = N \quad T(N) = NT(1) + (N-1)$

$= 8T\left[\frac{N}{8}\right] + 7$ $\quad = N + N - 1 = 2N$

③ $T(N) = T\left(\frac{N}{2}\right) + N$ $\quad a=1 \; b=2 \; c=1$

$= \left[T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N$ $\quad t = \log_2^1$

$= T\left(\frac{N}{4}\right) + \frac{3N}{2} \Rightarrow T\left(\frac{N}{2^k}\right) + KN \quad c > t \; O(N^c)$

$= \left[T\left(\frac{N}{8}\right) + \frac{N}{}\right.$ $\quad = O(N)$

$\Rightarrow$ ② $T(K) = 2^K T\left[\frac{N}{2^K}\right] + 2^K - 1$ $\quad 2^K = N$

$2^K = N \Rightarrow T(1) + 2^K$ $\quad NT(1) + N$

$K = \log N$ $\quad$ ~~$O(2^K)$~~ $\quad = O(2N)$

$= O(N)$

③ $T(N) = T\left(\frac{N}{2}\right) + N$

$= \left[T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N = T\left(\frac{N}{4}\right) + \frac{3N}{2}$

$= \left[T\left(\frac{N}{8}\right) + \frac{N}{4}\right] + \frac{3N}{2}$

$= T\left(\frac{N}{8}\right) + \left[N + \frac{N}{2} + \frac{N}{4} + \cdots\right]$

$= T\left(\frac{N}{2^k}\right) + N\left[\sum_{i=0}^{k} \frac{1}{2^i}\right] \Rightarrow T(1) + N\left[1 + \frac{1}{2}\right.$ ~~$\log N + N$~~

$\frac{N}{2^k} = 1 \Rightarrow K = \log N$

④ $T(N) = T\left(\frac{N}{2}\right) + 1$ $\quad \Rightarrow O(N)$

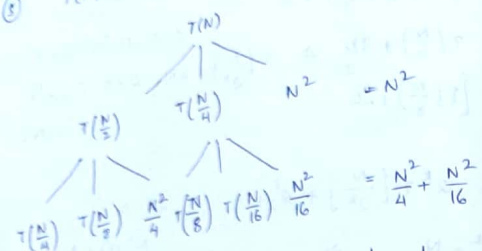
$= \left(T\left(\frac{N}{4}\right) + 1\right) + 1 = T\left(\frac{N}{2^2}\right) + 2$

$= T\left[\frac{N}{2^k}\right] + k \Rightarrow T(1) + \log N$

$\frac{N}{2^k} = 1 \Rightarrow \log N = k$ $\quad \Rightarrow \log N$

$\Rightarrow O(\log N)$

④ $T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{4}\right) + N^2$



$= N^2 + \frac{N^2}{4} + \frac{N^2}{16}$

$\Rightarrow N^2\left[1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{16} + \frac{1}{64} + \frac{1}{64} + \frac{1}{256} + \cdots\right]$

$\Rightarrow N^2\left[1 + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} + \cdots\right]$

3rd level $= \frac{N^2}{4} + \frac{N^2}{16} = \frac{5N^2}{16}$

4th $= \frac{N^2}{16} + \frac{N^2}{64} + \frac{N^2}{64} + \frac{N^2}{256} = \frac{25N^2}{256}$

5th $= \frac{5^3 N}{16^3}$

$\Rightarrow \frac{a(1-r^n)}{1-r} = \frac{\left(1-\frac{5}{16}^K\right)}{1-\frac{5}{16}} = \frac{16N^2}{11}\left[1-\frac{5}{16}^K\right]$

$= \frac{16N^2}{11}\left[1-(0.3)^K\right]$

$= O(N^2)$

⑤ $T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{4}\right) + N^2$

cant apply masteris theoram

☆ ☆ ☆ $T(N) = aT\left[\frac{N}{b}\right] + f(N)$

- Masteris theoram

$t = \log_a^b \quad f(N) = N^c$

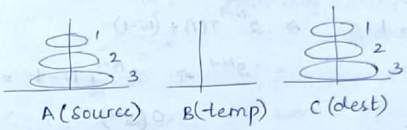
① $c < t : O(N^t)$

② $c = t : O(N^t \log N)$

③ $c > t : O(N^c)$

Problem Towers of Hanoi

N = 3



steps :
1 : A - C (1)
2 : A - B (2)
3 : C - B (1)
4 : A - C (3)
5 : B - A (1)
6 : B → C (2)
7 : A → C (1)

fun (src, dest) "A" "B" "C"

```
void TOH ( int N, char src, char tmp, char dest ) {
// 3 : A C → 2 : A → B
Hoi                ↘ Move nth from A to C
if (N == 0)
return;
                   2 : A B → 1 : B → C
                             ↘ Move (N-1)th from B to C
TOH ( N-1, src, dest, tmp );
Print ( N : src → dest );
TOH ( N-1, tmp, src, dest );
```

$T(N-1, a, b, c)$; (1) $T(N-1, b, c, a)$;

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

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

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

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

$= 4[2T(N-3) + 1] + 2$

$= 8T(N-3) + 3$

$= 2^k T(N-K) + k$

$T(1) = 1$

$N - K = 1$

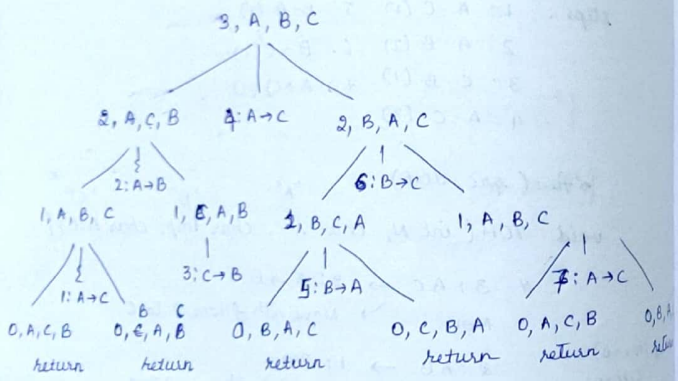
$N = k+1$ $\Rightarrow 2^{N-1} T(1) + (N-1)$

$= 2^{N-1} \Rightarrow 2^{N-1} + N - 1 = 2^N$

$= O(2^N)$

$T(N-1, src, dest, temp)$

$T(N-1, temp, src, dest)$



$= O(2^3) \Rightarrow O(2^N)$

Problem

ar_N 8 2 -3 7 15 -5

$K = 20$

Find the subset sum = k

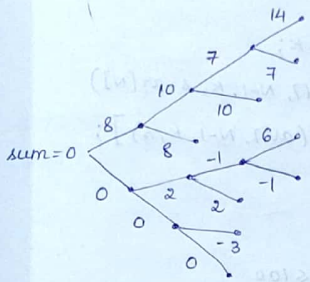
using bitwise operators, complexity = $2^N * N$

Other way to approach the problem,

8 2 -3 7 15 -5

2 2 2 2 2 2ways $= 2^5$

(take / dont)



Path to each leaf node gives the sum of one subset

sum = f(a+b) ⇒ f(a[i], a[i+1])

if (sum == k)

check

sum = f(a+0) = f(a[i], 0)

if (sum == k)

check

```
if (N == 0)
    return ;

bool fun ( int ar[], int N, int k)
{           int sum, int index
    if (index == N)
        return (S == k);
bool r = fun ( ar[], N, K, sum + ar[i], index+1);
bool s = fun ( ar[], N, K, sum, index+1);
        return (r || s);
}
```

T(ar[], N, K, sum + ar[i], ind+1);
T(ar[], N, K, sum, ind+1);

if N^3 , 3 levels of binary tree $= O(2^3)$

$T(N) = 2T(N) + 1$
$= 2[2T(N-1) + 1] + 1$
$= 4T(N-1) + 2.$

Solution

```
bool func (int ar[], int N, int K, int sum) {
    if (N == -1)
        return sum == K;
    return [ f(ar[], N-1, K, sum + ar[N])
           || f(ar[], N-1, K, sum) ];
}
```

$2^N \times T \approx 10^8$
$T \times M \leq 10^8$
$6 \leq 10^{6,7,8}$

$1 \leq T \leq 100$
$1 \leq N \leq 20$

This logic works for small N values

$2^{30} \approx 1$ sec
$2^{60} \approx 31.7$ yrs.

Problem Valid Parenthesis

N = 4 (())
()()

Order of sets : (()) ()()

N = 6 ((()))
(()()) (())() ()(()) ()()()
5 sets

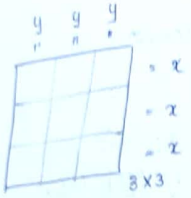
Print all combinations in lexicographical order.

Start with (
count of open & close N/2 N/2

```
func (char[] a, int N, int openCount, int closecount)
                                        int index
if (index == N)
print ar;
    if (openCount > 0) { a[i] = '('
    fun(a, N, oc-1, cc, i+1);
    }
    if (closecount > 0) { // && (openCount > closeCC)
        a[i] = ')'
        fun (a, N, oc, cc-1, i+1);
    }
}
```

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

Problem



Mat [i,j] ∈ [1,9]

sum of all rows should be same

sum of all columns should be same

N = 3X3

Elements [1,9] distinct

Combinations	M1	M2
5 2 8	1 6 8	5 1 9
3 7 1	5 7 3	3 8 4
4 6 9	9 2 4	7 6 2

Given matrix → cost of converting given into magic matrix

(M1) (5-1) + (2-6) + (8-0) + - - - - - = 26.

(M2) 0 + 1 + 1 + - - - - = 16.

```
for (i = 0 to N)
    for (j = 0 to N)
        cost += |a[i] - a[j]|
```

Date 26.05.19

CLASS - 4

Problem Generate a magic matrix of N = 3

[0-2] [3-5] [6-8]

```
func (int[] ar, int N, int elements[])
{
    int index
    if (N == index)
        return; // Validate sum
    ar[index] = elements[0]
    for (i = 0 to N) {
        for (j = 0 to N) {
            if (vis[i][j] == false) {
                a[index] =
                vis[i][j] = true;
            }
        }
    }
```

1 2 3
4 5 6
~~7~~ 8 9 ~~7~~ 8

isVisited = 3X3

Solution

```
for (int i = 1 to 9)
    if (vis[i] == false) {
        ar[index] = i
        vis[i] = true;
        func (ar, index+1, vis);
        vis[i] = false;
    }
```

(N) vis[] = 1 to 9

Break cond:

```
if (index == 9)
    if (valid(ar) == T)
```

```
ans = min (ans);
return (cost(arr, mat))
```

Problem

```
bool valid (int arr[])
{
    return (ar[0][0] + ar[0][1] + ar[0][2] ==
        && (ar[1][0] + ar[1][1] + ar[1][2] ==
        && . . .
        // all rows & columns.
```

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

STRINGS

str: Smart - Interviews

Problem Each partition is a valid word from the list of given words

set : { "smart", "art", "int", "inter", "intern", "view", "views", "s" }

Generate all combinations of words given in set

Approach 1 : { "smart", "art" . . . } combinations

Approach 2 : s/mart Interviews
sm/art Interviews
sma/rt Interviews

```
bool func (int or set, int N, c
          char

bool func (string[] set, int N, string word)
{
    if (index == N)                    string cur_word, int index
        return (cur_word.equals(word));
    bool r = func (set, N, word, cur_word.concat(
                                  set[i], i+1);
    bool s = func (set, N, word, cur_word, i+1);
    return (r || s);
}
```

Partition the given word:

```
bool func (string[] set, int N, string word, int index)
{
    string partition = word.substring (index, word.
                                       length);
    for (int i = 0 to N)
        if (set[i].equals(partition)
            return true;
    return func (set, N, word, index+1);
```

Solution

```
bool fun(string word, List<string> set, int index,
                                              int N)
    if (index == N)
        return true;

    for (i = index to N)
    {   p = str[idx, i]
        if (p ∈ L) &&
            (fun(str, L, i+1, N))
        return true;
    }
```

(OR)

```
for (i = index to N)
{   if (str[idx, i] ∈ L &&
        fun(str, L, i+1, N)) {
            return true;
    }
}return false;
}
```

Problem

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a_{k\,N=20}$	-1	-1	0	0	0	-1	-1	-1	1	1	1	-1	-1	-1	2
								16	17	18	19				
								2	2	2	-1				

	0	1	2
$b_{k\,N=3}$	3	2	5

K = 2

Place the blocks ($b_{k\,N=3}$) so that distance between blocks is ($K=2$) in the array ($a_{k\,N=20}$)

$b_{k\,N=3}$ 3 2 5

3 blocks of 0

2 blocks of 1

5 blocks of 2

Each block is 2 blocks (k) away

Base check: if ($b_k N <= a_k N$)

$+ N * K$

$3+x,\ 2+y+5+z = 20$

$x+y+z = 10$

$x=2, 6$ (2)

$y=4, 6$

$z=2, 6$

$(2, 3, 4, 5, 6)$

$(2, 3, 4, 5, 6)$

$(2, 3, 4, 5, 5)$

```
for (int i = 0 to a_k N)
{
    for (int j = 0 to b_k N)
    {
        for (int k' = 0 to b_k N[j])
            ar[i] = j;
            i++; k'++;
        for (int l' = 0 to K)
            ar[i] = -1;
            i++; l'++;
    }
}
```

```
void func (int ar[], int br[], int K, int aindex,
                    int bindex, int aN, int bN)
{
    for (int i = aindex to br[bindex])
ar[aIndex++] ~~ar[i] = br~~ bindex; i++;
    // func (ar, br, K, i, bindex+1, aN, bN);
```

```
// Add spaces after block only if it's not the last
for (int i=0 to K) // i = aIndex
    ar[i] = -1;  i++;

func (ar, br, K,
      aIndex
      bIndex
      if (bIndex != (bN-1)) {
for (int i=0 to K)
    a[aIndex++] = -1; } // a

fun (ar, br, K, aIndex, bIndex+1, aN, bN);

Break condition: if ((aN - aIndex) < ((bBlocks * K) + bBlocks sum))
                     return;

Solution 1   for (int i=0 to br[ind2])
                 ar[idx1++] = ind2;

             if (ind2 == M-1) {

             } else {
the spaces       for (int i=0 to K)
modification         a[idx1++] = -1;
can be called        func (ar, br, N, M, idx1, idx2);
from Main() if   }
we choose to place -1 before block
        K spaces can range from K to arN
        to generate all combinations
```

```
func (ar, br, N, M, idx1, idx2}, K)
{
    if ((aN - idx1) < ((M-idx2) * K + sum(br, idx2)))
        return;
    for (int i=0 to br[idx2]) {
        ar[idx1++] = idx2;
    }
    if (idx2 != (M-1)) {
        for (int i=0 to K) {
            ar[idx1++] = -1;
        }
TODO!   } // add a while loop to handle placing ki spaces
    func (ar, br, N, M, idx1, idx2+1, K);
}

func sum (br, idx2)
{
    int count;
    for (int i= idx2 to br(M))
        count += br[if * i];
    return count;
}

func main() {
    func (ar, br, N, M, idx1, idx2, K);
    for (int i=0 to N)
        S.O.P (ar[i]);
}
```

SORTING

- Bubble Sort $O(N^2)$
- Selection $O(N^2)$
- Insertion $O(N^2)$
- Merge $O(N \log N)$
- Quick $O(N \log N)$
- Heap
- RS
- Count Sort $O(M+N)$

★ Bubble Sort

ar_N 5 3 -1 2 10 8 15

3 ~~5~~ ~~-1~~ ~~2~~ ~~5~~ 8 10 (15) highest

-1 ~~2~~ 3 5 8 10 15

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

$O(N^2)$ Worst
$O(N)$ Best

★ Selection Sort

find the smallest element & swap

ar_N 5 3 10 8 -1 2 15

i ↑ , min ↑

(-1) | 3 10 8 5 2 15

i ↑ , min ↑

-1 (2) | 10 8 5 3 15

```
func selectionsort (ar, N) {
    int minIndex = -1;
    for (int i=0 to N) {
        for (int j= i+1 to N) {
            if (min > ar[j])
                // min = ar[j];
            }       minIndex = j
        swap (ar, i, /// minIndex);
    }
}
func swap (ar, i, j) {
    if ( i == -1)
        return;
    int temp = a[j];
    a[j] = a[i];
    a[i] = temp;
}
```

$O(N^2)$ // Both worst & best case

★ Insertion Sort

	0	1	2	3	4	5	6
ar_N	5	3	10	8	-1	2	15

pick the next element & place it in its sorted order position

3 5 | 10 8 -1 2 15

i ↑ , j ↑

```
if (a[j] < a[i]) {   // 3 5 8 10 -1 2 15
    x = a[j]                  ↑ ↑
    a[j] = a[i];              i j
}
```

```
for (int i = 0 to N)
    for (int j = i+1 to N)   int j = i+1;
        if (a[j] < a[i])
            x = a[j];
            a[j] = a[i];
            while (a[j] < a[j-1] && j >= 0)
            {   a[j] = a[j-1];
                j--;
            }
            a[j] = x;
        }

func selectionSort (int [] ar, int N)
{   int x = 0;
    for (int i = 0 to N) {
        int j = i+1;
        if (a[j] < a[i]) {
            x = a[j];
            while (j >= 0 && a[j] < a[j-1]) {
                a[j] = a[j-1];
                j--;
            }
            a[j] = x;
        }
    }
}
```

O(N²) Best case complexity : O(N)

★ Count sort

ar_N = 5 1 3 10 3 7 2 2 5 3 10 3

1 ≤ ar(i) ≤ 10

(or) // sort of count
// Maintain hash

```
                                          ((b-a)+1)
func (int[] ar, int N, int noOfCandidates)
{   int[] count = new int [noOfCandidates+1]; max = -1; count
    for (int i = 0 to N) {
        count [a[i]-a] = count [a[i]] + 1;
    }
    for (int i = 0 to noOfCandidates) {
        if (max < count [i])
            max = count [i];
    }
    return max;
}
// sort the given array using count array
func (int[] ar, int N, int M)
{   for (int i = 0 to M) // int j = 0; j ≤ N
        while for (int j = 0 to count [i])
            a[j++] = i + a;
    }
}
```

O(10*N) Time
O(10) space

O(M+N)

O(M*N) ~~worst case~~

Limitations

- Negative nos

$[a,b] = (b-a)+1$

$-10 \le a[i] \le 20 \Rightarrow 31$ elements

Count [31] elements

- Large nos

$10^9 \le a[i] \le 10^9 + 1000$

$1 \le a[i] \le 1000$

$c[0] \to 10^9$

$c[1] \to 10^9+1$ ....

$a \le a[i] \le b$

```
func (a[], N, (b-a)+1)
{  // no of candidates = (b-a)+1;
}
```

- Can't use if no of candidates $> 10^{6 \to 7}$

  Space Constraint

- Shouldn't use when $R >> N$

  Ideal: $R \le N$

Problem

A_N : -1 3 10 12 15 18 20

B_M : -5 -2 12 25 30

Print (M+N) elements in sorted order

```
void fun(int[] a, int N) {
    for (int i=0 to N) { // int j=0 to M
        for(int j=0 to M) {
            if (a[i] ≤ b[j]) {
                S.O.P(a[i]);
                i++;
            } else {
                S.O.P(b[j]);
                j++;
            }
        }
    }
    for (int l=j to M)
        S.O.P(b[l]);
    for (int k=i to N)
        S.O.P(a[k]);
}
```

O(M×N)

→ N+M+(N+M)², N+M

→ N+N, N+M

→ N+M, 1

Problem A_N : -1 3 10 12 15 18 20 - - - - -

B_M : -5 -2 12 25 30

```
func (int[] a, int[] b, int N, int M)
```

// way 1

```
if (a[i] ≥ b[j])
    insert b[j] @ ith location
    shift all other elements in A
```

O(N×M)

O(M×N)

// way 2 copy all elements of B to A

Do in place sort (bubble / selection / insertion)

$M + N^2$

// way 3 Initialize a new array C N+M
Merge the elements to c by comparing A & B
$(N-M)+N$, N

// way 4 Start from the end $(N-1)$ position of A
Place the largest element by comparing A & B
In place sorting without extra space
$(N-M)+M$, 1

TODO Implement way 3 & way 4.

Problem

ar_N : 0 0 1 0 1 1 0 0 1 1
sort 0's & 1's

// way 1 Count no of 0's
Count of 1st = N − count 0's
Override the given array.
$O(N+N)$

// way 2 Two pointers for start & end
Push all 1's to end & 0's to start
$O(N)$

```
int p1=0, p2=N-1
for(int i=0 to N) {
    if(a[i]
```

```
while (p1 ≤ p2) {
    if (a[p1] > a[p2]) {
        a[p2] = a[p1]; // 1
        a[p1] = 0;
        p1++; p2--;
    } else {
        p1++; }
}
```

Ex: 0 0 1 0 1 1 0 0 1
P1 ↑ P2 ↑

$P_1 = 0$
$P_2 = 8$ $0 > 1$
$P_1 = 1$
$P_2 = 8$ $0 > 1$
$P_1 = 2$
$P_2 = 7$ $1 > 0$ replace elements

0 0 0 0 1 1 1 0 1 1
↑→ ↑ ↑ ↑
P_1 P_2

Ex: 0 0 0 0 0
↑ ↑
P_1 P_2
P_1 → search for 1 $P_1 = P_2$ break

Ex: 0 1
↑ ↑
P_1 P_2
P_2 → search for 0
P_1 → search for 1

0 1
↑ ↑ break
P_2 P_1

Ex: 1 0 swap
↑ ↑
P_1 P_2

1 0 0 1 0 1 1 1 1
↑ ↑
P_1 P_2

$P_1 = 0$
$P_2 = 8$ $1 > 1$

```
{
while (a[p2] != 0)
    p2--;
while (a[p1] != 1)
    p1++;
a[p2] = 1;
a[p1] = 0;
} do while (p1 ≤ p2)
```

```
func (int[] ar, int N)
{   int p1 = 0, p2 = N-1;
    while (p1 ≤ p2) {
        while (a[p1] != 1)
            p1++;
        while (a[p2] != 0)
            p2--;
        a[p2] = 0;
        a[p1] = 1;
    }
}
```

Date 01-06-2019

CLASS-5

Problem

	0	1	2	3	4	5	6	7	8
ar_N :	5	10	3	8	12	15	1	7	18

Count the no # (i,j) of pairs such that

$0 \le i < j < N$

$ar(i) > ar(j)$

$\frac{n(n-1)}{2}$ = 8C2

sort :

1	3	5	7	8	10	12	15	18
0	1	2	3	4	5	6	7	8
18	15	12	10	8	7	5	3	1

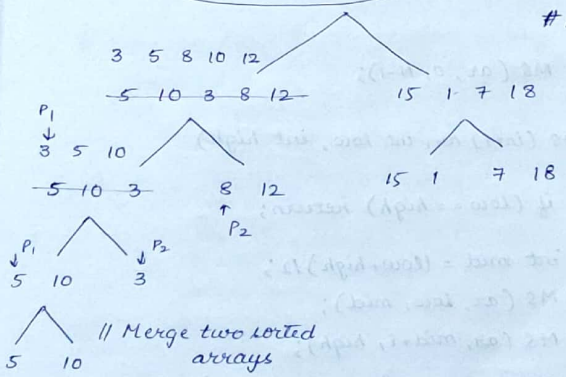
// can't change the array.

$\frac{8(8+1)}{2}$ = No sorting

ar_N :	5	10	3	8	12	15	1	7	18
	2	4	1	2	2	2	0	0	0

O(N2), 1

MergeSort



// Merge two sorted arrays

```
mergeArrays (a, b)
{   for (int i = 0 to a.length)
    {  for (int
    int [] c = new int [ a.length + b.length ]
    while (a.length > 0 && 
```

Steps: Divide the array

low	high	mid
0	n	$\frac{low+high}{2} = n/2$

continue until low = high

merge left & right sub arrays to a temp arr

copy the elements of temp arr to original arr

Solution

```
fun ()
  {   MS (ar, 0, N-1);
  }
void MS (int[] ar, int low, int high)
{
      if (low == high) return;

      int mid = (low+high)/2;
      MS (ar, low, mid);
      MS (ar, mid+1, high);
      MergeArrays (ar, low, mid, high)
  }
Void MergeArrays (int ar, int low, int mid, int high)
{
    int [] temp = new int [high - low + 1];
```

```
for int n = mid - low + 1;   i = low; K = 0
    int m = high - (mid+1);  j = mid + 1
                             +1
while (i <= n && j <= m) {
      if (a[i] > a[j])
            c[K++] = a[j]; a[j++];
      else
            c[K++] = a[i]; a[i++];
}     i++;
while (i < n)
{     c[K++] = a[i++];    // Copy the remaining
}                            elements
while (j < m)
{     c[K++] = a[j++];
}
// copy the temp arr to original array
for (int i = low to high)// K=0 to (high-low+1)   i++ K++
      a[i] = c[K]
```

$O(N)$, N

$T(N) = T(\frac{N}{2}) + T(\frac{N}{2}) + T(N)$ ✓ $O(N \log N)$

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

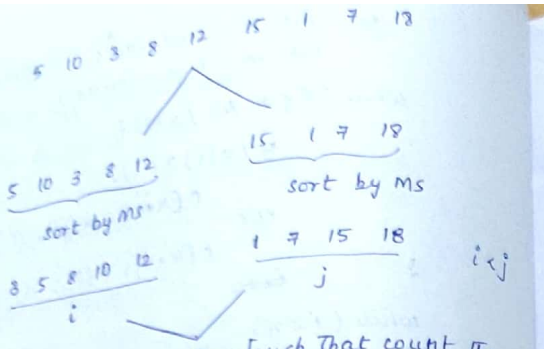
$T(N) = 2T(\frac{N}{2}) + N$

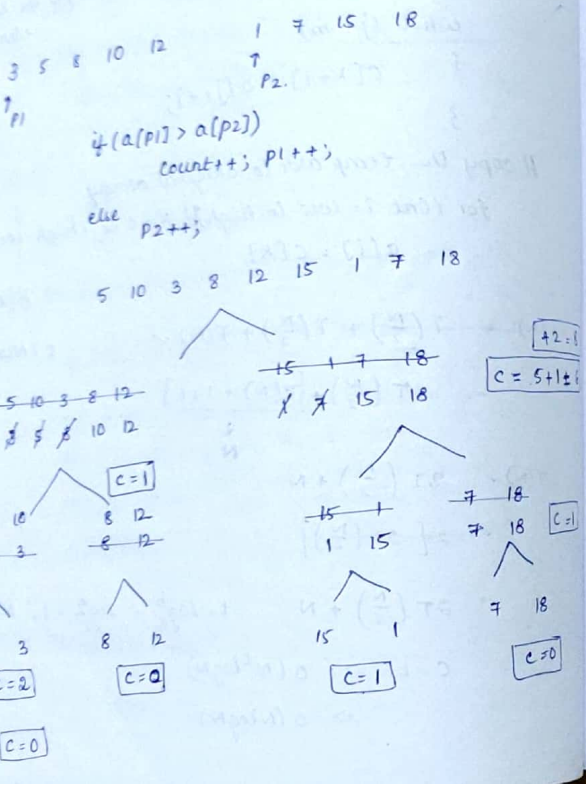
$= 2[2T(\frac{N}{4})]$

$= 2T(\frac{N}{2}) + N$ $t = \log_a^b = \log_2^2 = 1$ $N = N$ $C = 1$

$C = t \Rightarrow O(N^t \log N)$

$\Rightarrow O(N \log N)$





3 5 8 10 12 1 7 15 18

C = 1+1+2+2+2 = 8

★ NOTE $0 \le ans \le \frac{N(N-1)}{2}$

INVERSION COUNT (given problem name)

ar : 3 5 10 12 18 [min]

ar : 18 12 10 5 3 [max]

4
3
2
1

$(n-1)+(n-2)+(n-3)+\dots+1$

$\Rightarrow \frac{(n-1)n}{2}$

Problem

	0	1	2	3	4	5	6	7	8
ar_N :	5	10	-3	8	12	15	1	-7	18

ar(i) + ar(j) = k

i ≠ j K = 23 ✓, 13 ✓, 24 ✗, 20

Brute force : $O(N^2)$

sort : -7 -3 1 5 8 10 12 15 18 (P1 ↓, P2 ↓)

i = 0 to N

if (a[i] + a[N-i-1] == K)
return true;
else
i++; // P1 & P2

ar(P1) + ar(P2)
< K → P1++
= K → return true;
> K → P2--

Space : 1 Sorting : O(NlogN) finding pair : O(N)

2- Pointer Technique

* Merge sort
* Check if str is a palindrome
* Swap to reverse a string
* count = k such that a(i) + a(j) = k
* sort 0's & 1's
* Prev problem // Inversion Count

Problem Statement

- ar(i) + ar(j) = k
- ar(i) - ar(j) = k
- ar(i) + ar(j) + ar(K) = k
  - ar(i) + ar(j) = K - ar(K)

↑ Fix the position of index K for each iteration

K = 0 to N

Compute K - ar(K) ⇒ K'

Now, find ar(i) + ar(j) = K'

ar(i) + ar(j) = k

1) N^2, 1 (BF)
2) NlogN + N, N (MS)

ar_N : -7 -3 1 5 8 10 12 15 18

K = 20

SEARCHING

* Linear Search → N, 1
* Binary Search → NlogN + logN, 1

```
                                    int low, int high
boolean BSR(int[] ar, ~~int n~~, int k)
{   // int low=0, high=N-1, mid=(low+high)/2;
    if (ar(mid) == k)
        return true;
    if (ar(mid) < k)
        return BSR(ar, mid+1, high, k);
    else
        return BSR(ar, low, mid-1, k);

    // boolean res = ar(mid) < k ? BSR(---) : BSR(--)
    // return res;      //return false;
}

boolean BSI(int ar[], int n, int k)
{   while(low ≤ high)
    {   mid = high + low/2;
        if (ar(m) == k)
            return T;
        if (ar(m) < k)
            low = m+1;
        else
            hi = m-1;
    }
    return F;
```

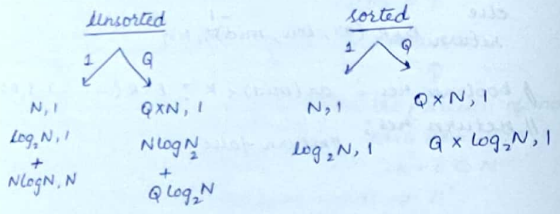
```
boolean BSR(----)
{   if (low > high)
        return false;
    if (a[m] == k)
        return true;
    if (a[m] < k)
        return BSR(a, k, m+1, hi);
    return BSR(a, k, low, m-1);
}
```

$T(N) = 2T\left(\frac{N}{2}\right) + 1$

Master theorem : $T(N) = aT\left(\frac{N}{b}\right) + N^c$

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

$= \log 2$



Problem using binary search, find the pair (i,j) such that

$a(i) + a(j) = k$ & $i \neq j$ & $i < j < N$

way1 Sort & 2 pointers

way2 brute force

way3 Binary search

ar_N : -7 -1 5 10 12 18 $\quad x = 10$

$a(i) + a(j) = k$

$a + b = k$

$b = k - a$

↑ fix this element

search for if $(k == b)$ using binary search

-7 -1 5 10 12 18

i ↑

$a = -7$

$k' = k - a$

$= 27$

search for k' in $(i+1$ to $n-1)$

BSR(ar, k-ar(i), i+1, N-1);

```
for (i = 0 to N)
{   BSR(ar, k-ar(i), i+1, N-1);
}
```

Time complexity : $N\log_2 N + N\log_2 N$, N

Other way : BSR(ar, k-ar(i), 0, i-1);

$$\begin{cases} a+b=k \\ a+b+c=k \\ a-b=k \end{cases}$$

TODO Solve the above problems using binary search

Problem ar_N : 5 -3 12 18 -6 2 15 -1 25

x : $\max_{i=0}^{N-1} (ar(i) \leq x)$

Q ↓ 17 → 15

5 → 5

-10 → $-\infty$ (INT_MIN : -2^{31})

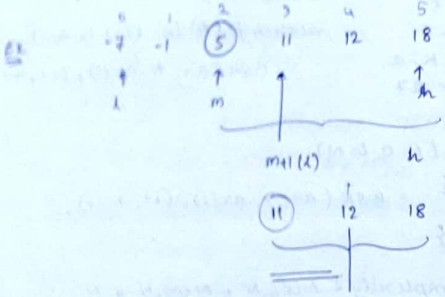
7 → 5

x : $\max_{i=0}^{N-1}(a[i] \le x)$ $\quad Q \times N$, 1

```
int cur_max = 0;
for (int i=0 to N)
    if (a[i] > cur_max && a[i] <= x)
        cur_max = a[i];
```

```
int fun (int ar[], int N, int x)
```

Ex



Solution

```
int ans = -∞;   l = m-1
while (low <= high)
{
    mid = (low + high)/2
    if (a[i] == K)
        return a[i];
    if (a[i] > K)
        hi = m-1;
    elseif (a[i] <= K)
        ans = ar[m];
        low = mid+1;
}
return ans;
```

Complexity : $N\log_2 N + Q\log_2 N$, N

Problem

ar_N : 5 1 8 1 5 5 3 15 5 5 20 18 5 8

10 10 10

Q ↓ x : $f(x)$ frequency of x

way1 : $O(N) \times Q$, 1

```
if (ar[i] == x)
    count++;
```

way2 : Maintain a count array (only if size/range is min)

$N + Q \times 1$, R

way3 : $N\log_2 N + Q(\log_2 N + N)$, N

Sort the array. search for Q (Binary search) & use 2pointers (P1-- && P2++) to count range

1 1 5 5 5 5 5 8 | 8 15 20 | 100 100 100 100 Q = 100

mid

mid

```
if (a[i] == Q)
    P1 = i, P2 = i
while (a(P1) != Q || a(P2) != Q)
{   P1--;
    P2++;
}
count = P2-P1+1;
check for P1 >= 0 && P2 < n
```

Check for range of Q such that

x : $\max_{i=0}^{n-1} Q+1$

$\min_{i=0}^{n-1} Q-1$

1 1 3 5 5 5 5 5 8 8 10 10 10 15 18

ar(m) < k : le = m+1

ar(m) == k : BS1 : ans = m, hi = m-1

ar(m) > k : hi = m-1

way 4 : $N\log_2 N + Q \times 2\log_2 N$, N

BS1 : to find min index

BS2 : to find max index

Way 5 : $N\log_2 N + \cancel{Q \times N}$, $N + Q\log N$, $3N \approx O(N)$; Compress

ar[N] : 1 3 5 8 10 15 18

count[N] : 2 1 5 2 2 1 1

Sort the array. Maintain an array of non-repeating elements. Count the frequency of each element & store it in count array. The Q queries will check in both arrays using 2 pointers.

way 6 Q[N] : 1 2 5 8 10 20

Sort the query array & use 2-pointers to check for frequency.

ar[N] : 1 1 3 5 5 5 5 8 8 10 10 (p_2)

Q[N] : 1 2 5 8 10 20 (p_1)

Ans[N] : 2 0 4 2 2 0

But the given order of query is modified as it is sorted. Try to store the index & print the ans in order given. Maintain another array for the order of Queries & do Binary Search.

TODO Fix the way 6.

$2N\log_2 N$, $2N$

Inbuilt Libraries

Hashing

Sets <K> Maps <K,V>

↑ sorted / unsorted ↑

(K)

→ insert
→ search
→ delete
→ size

Problem Check for 2 repeated nos in a given array

ar_N : 1 5 3 5 2 4 6 2 8

Set <K> : { 1, 5, 3, 2, 4, 6, 8 }

a = ~~1~~ 5

b = ~~1~~ 2

Print a, b in sorted order

Problem Check for element which doesn't occur 3 times

ar_N : 1 5 3 3 6 5 1 1 5 3

Map <K,V> : <1,3> <3,3> <5,3> <6,1>

Iterate over keys & get count

Date 08-06-2019

CLASS - 7

CLASS-6 - Full day lab session

Problem Implement square root of a number (integer)

find the factors a, b
break when a == b $O(T*\sqrt{N})$, 1

Solution

```
int sqrt (int N)
{   if (N == 1) return 1;
    for (int i = 2 to N) {
        if (i*i == N)
            return i;
    }
    return -1;
}
```

way2

low	high	mid	(N = 25)
1	25	13	(1+25)/2 = 13
1	12	6	high = mid - 1
1	5	3	if (mid * mid > N)
4	5	4	low = mid + 1
5	5	5	if (mid * mid < N)

```
int sqrt (int N, int low, int high)
{   int mid = (low+high)/2;
    if (mid * mid == N)
        return mid;
    if (mid * mid > N)
        return sqrt (N, low, mid-1);
    if (mid * mid < N)
        return sqrt (N, mid+1, high);
}
```

Complexity : $O(T * Log_2 N)$, 1

Problem

ar_N : -5 -5 1 1 4 4 10 12 12 15 15 18 18 31 31

Given a sorted array with every element repeated twice except for one. Find the non-repeated number.

way1 : Brute force
Iterate through elements linearly, maintain count
& break if (a(i) != a(i+1)) & count != 2

way2 : skip one element & search for non-repeated

way3 : XOR of elements

way4 :

0	1	2	3	4	5	6	7	8	9	10
-5	-5	1	1	10	4	4	12	12	18	18
					↑					

a[mid] = 4

```
if (a[mid] == a[mid+1]) // (N - mid + 1) % 2 == 0
        high = mid - 1;
if (a[mid] == a[mid-1]) // (N - mid + 1) % 2 != 0
        low = mid + 1;
```

Approach 1 :
use index as condition $O(Log_2 N)$, 1

Approach 2 :
use length as condition $O(Log_2 N)$, 1

```
int findElement(int[] a) {
    int high = n-1, low = 0, mid = high+low/2;
    while (low < high)
    {  int mid = (low+high)/2;
       if (mid%2 == 0) {
            if (a[mid] == a[mid+1]) {
                low = mid+2;
            } else {
                high = mid-2;
            }
       } else {
            if (a[mid] == a[mid+1]) {
                high = mid-2;
            } else {
                low = mid+2;
            }
       }
    }
```

Code: complete the code above

Test cases : 1 1 5
1 5 5

Problem

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

Each task takes given ar_i minutes to complete

Assign each task to workers w_j such that it is completed in min amt of time.

X . 3 (workers)

Edge case: min time / task allocated optimally

K = 3

Total time = 57 min

workers : 3

Optimal time = 57/3 = 19 min each w_j

Find subset of array such that $\sum a_i \approx 19$

~~5~~ 3 10 2 / 7 4 8 1 / 3 5 2 1

w_1 = { 5, 10, 4 } = 19

w_2 = { 2, 8, 1, 3, 5 } = 19

w_3 = { 7, 2, 1, 3 } = 19

K = 4

Total time = 57 min

Workers = 4

Optimal time = 57/4 = 16 ~ 19 min each

Pseudo

Arrays.sort(a);

1	1	2	3	3	5	5	7	8	8	10
↑ P1		sum = 16 ~ 17								↑ P2

Find a subset where sum ≈ 17

```
for (int i = 0 to N)
    isVisited[i] = false;

for (int i=0 to K) {
    int p1 = 0, p2 = N-1; sum = 0, maxsum = N/K;
    while (p1 < p2) {
        sum += 
```

Way1 $^{N-1}C_{K-1}$

generate partitions to calculate sum of each partition

TODO

Way2 Given arr = 35, K = 3, Sum = S

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

31 26

ans = 35, 36, 37, 38, 39

atmax wj takes 31 min

ans = 15 we need 5 workers but we just have 3

= 15, 14, 13, 12, 11

low = S/K (OR) = 0	low	high	mid	ans ∈ [low, high]	
	19, 0	57	28	ans = 28	K = 3
	0	27	13	ans = 28	K = 5+
	14	27	20	ans =	K =

Solution

```
int solve (int [] arr, int N, int K
{
    int low = 0 or S/K,
        high = S,
        mid,
        ans = S;
    while (low <= high) {
        mid = (low+high)/2;
        if (Valid (arr, N, m, K)) {
            ans = m
            high = mid-1;
        } else {
            low = mid+1;
        }
    }
    return ans;
}
```

TODO

Linearly generate partitions in Valid (arr, N, m, K)

Complexity : $N * Log_2(S - S/K+1)$

(OR)

$N * Log_2 S$

NOTE The order of (partitions/) tasks should not be changed

Problem: Name: Aggressive Cows { SPOJ }

ar_N: 5 7 10 11 13 18 20 22 25 30 34 38 42 43 45

N=15

Place k=6 families in 'N' houses such that they are max distant from each other.

Way 1

5 7 10 11 13 18 20 22 25 30 34 38 42 43 45

(45-13) = 22 (max)

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

ans = 5, 4, 3, 2, 1

ans = 10, 11, 12, 13, 14

TODO

distance >= 5

k = 6 $hi = \{5, 10, 12, 25, 30, 32\}$

distance >= 10

k = 6 $hi = \{5, 18, 30, 32, ---\}$ Not possible

low	high	mid	ans ∈ [low, high]
~~40~~ 0	40		

HINT

check for the below pattern to use binary search

① ------ T T T T T F F F F ----

② ------ F F F F F T T T T ----

Problem: Interview Bit : N-Arrays, contains duplicates

ar_N: 3 5 10 12 18 35 42 47 51

br_M: 1 7 9 15 33 40

N+M = ODD

NO DUPLICATES

Find the median of the N+M elements, i.e middle element of the sorted data set

way 1 Use extra space c_{N+M}, sort the elements & find the middle element $\frac{N+M}{2}$ index

for(int i=0 to N) if(-a[i]) O(N+M), N+M

way 2 Maintain count of elements of cur-index while sorting without extra space; i.e use 2 pointers; when $p1+p2 = \frac{N+M}{2}$ return the element $O(\frac{N+M}{2})$, 1

Brute Force Copy all the elements & Then sort

$O(N+M + (N+M)\log_2(N+M))$, N+M

way 4 Use binary search ans ∈ [10, 51]

low	high	mid
$Min(a_0, b_0)$	$Max(a_{N-1}, b_{M-1})$	
1	51	26
1	25	13
14	25	

26 → No of Ele less than 26 : 10

26 → No of Ele greater than 26 : 5

Also, use Binary Search to find the no of element < or > than mid

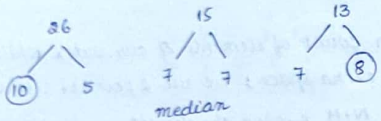
```
x = BS1(A, N, m) + BS1(B, M, m);
y = BS2(A, N, m) + BS2(B, M, m);
```

x = no of elements than m in A & B
greater

y = no of elements lesser than m in A & B

```
if(x>y){
    high = mid-1
}else{
    low = mid+1
}
```

Ex



median

Complexity : $(\log_2 M + \log_2 N) * \log(M-m)$

Sol^N	Insert(x)	Search(x)	Delete(x)
Unsorted array	1	N	N
sorted array	$(N \log_2 N)$ sort N Insertion sort	$\log_2 N + M$ shifts	$\log_2 N + M$ shifts
** BBST (Balanced BST)	$\log_2 N$	$\log_2 N$	$\log_2 N$

Given 10 digit phone nos.

Unsorted single linked list	1	N	N
Sorted single linked list	N	N	N
Binary Search Tree	H $\{\log N, N\}$	H	H → Skewed tree → N Balanced BST → $\log_2 N$ [AVL Trees, Red Black Trees]
[DAT] direct Access Table	1	1	1

NOTE: This can't be implemented due to space constraints

bool cnt[ ] = {F}
↑
10 digit number
space = $10^{10} \Rightarrow 10^9 = 1GB$
$\Rightarrow 10GB$

Alternate: Hash the ph no & store the index

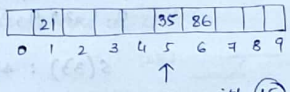
Hash key = h(x) = $\frac{x \% M}{\text{Hash value}}$ ≈ Hash Function

M = size of the Hash Table

x = phone number; i/p

Ex: 35, 21, 86, 24, 15, 63, 45, ...

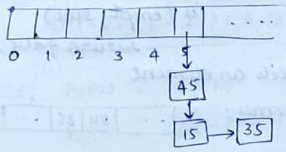
Issue with collisions



Collision with (15)

- Separate chaining

ArrayList < LinkedList >

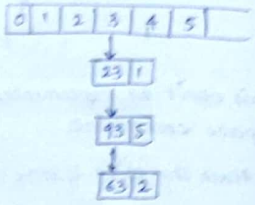


head ptr is enough to insert a node in the list.

	Insert(x)	Search(x)	Delete(x)
Separate chaining	1	L (length of the linked list)	L

NOTE: To handle the duplicate elements

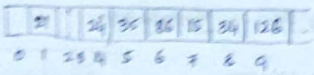
- Maintain count for every node
- search if the node exists before insertion
- Ex:



- Open Addressing: $h(x) = (x+i) \% M$

↳ Linear Probing

i/p: 35, 21, 86, 24, 15, 34, 126

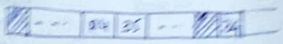


S(33): start from index 3 & continue

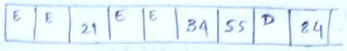
S(94): start from index 4 & continue till M comes back to 0 & search for (empty slot) return false;

D(15): Lets say, delete an element

Then array becomes;

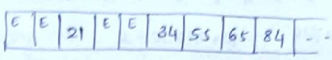


S(34) would give a WA as there is an empty slot in between



To avoid this problem, maintain whether the block is Empty or Deleted while searching

S(65)



↳ Quadratic Probing

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

M = 100

Ex: arr = 103, 1203, 703, 903, 203, 104, 1204, 704, 904, 204

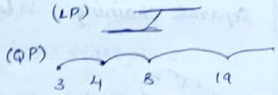
~~LP : 3 4 5 6~~

LP : 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

QP : 3, 4, 7, 12, 19, 5, 8, 13, 20, 29

$(3+4^2)$ $(4+25)$

NOTE: QP avoids forming clusters like in LP



↳ Double Hashing

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

where, $h_1(x) = x \% M$

$h_2(x) =$ (sum of digits) % M (or) $(x/2) \% M$

i/p : 35, 21, 86, 24, 15, 34, 126, . . .

$h_1(x) = x \% M$

$h_2(x) =$ (sum of digits) $\% M$

86	24	35	15	21	34				
0	1	2	3	4	5	6	7	8	9

$(7+5)\% 10 \Rightarrow 12\% 10$ $\quad (6+4)\% 10 = 0$!

$3+1 \% 10 = 4$ $\quad (5+6)\% 10 = 1$

$14+6 \% 10 = 0$ $\quad 7+4 \% 10 = 1$

When we hit collisions, we can go with LP or QP

In the above example, we considered double hashing

ex: LP

Sol^N	Insert(x)	Search(x)	Delete(x)	
Open Addressing	P	P	P	Probing
a) Linear Probing				
b) Quadratic Probing				
c) Double Hashing				
Separate chaining	L	L	L	Length of list

NOTE

Properties of a good Hash Function:

a) Uniformly distribute the hash keys over the hash table

b) simple to compute

How

How to address duplicates in Open Addressing ?

34, 25, 69, 34

Way 1

ar_N : 34, 25, 69

4 5 9

hr_M : 2 1 1 frequency

Way 2

ar_N : 34 25 69

2 1 1

Store the count in a single array

A formula to calculate a N such that it stores the count

To store the state of each cell,

br_N : -1 0 +1 +1 +1

-1 Deleted

0 Empty

+1 Occupied

Ex:

N	M	SC	OA
1000	100	✓	×
		↑ Time comp	
1000	1000	✓	$\geq N$

Break cond for S(x) Empty slot same start place

1000 slots

Empty slot break cond

2000 slots (or) 50% empty

$\Leftarrow$ 10000 slots (or) 90% empty

The complexity $\approx O(1)$ with good hash function & extra space

$\Rightarrow$ similar to DAT

```
class Link  // Separate Chaining

class Node {
    Node next;
    int count;
    int element;
                    Node ( int E, int C, Node p
                    {
                      // initialize vars
                    }
}

class HashMap {
ArrayList<LinkedList> //int [] ar = new int [10000];  // constructor
    public void insert ( int E ) {
        int hash = E % M ;   // M = 10000
        ar[hash] = E ;
    }

bool int search (int E)
        // handle the list
        if ( ar[hash] != -1 )   // Empty condition
        {   ar[hash] = E ;
        if  Node node = new Node ( E , 1 , null);
        else {
    } else {  Node temp = ar[hash]
        wh
```

```
class HashMap {

TODO    ArrayList < Node> ar = new ArrayList <Node> ;
    void insert (int E) {
        int M = 10000 ;
        int hash = E % M ;
        if ( ar.get (hash) != null)
        {
            Node temp = ar.get (hash);
            while ( temp != null) {
                if ( temp.element == E) {
                    temp.count += 1 ; break ;
                } else { is Element
                    return ;
                } temp = temp.next;
            }   } Node head = ar.get (hash);
            Node newElement = new Node (E, 1, head)
            ar.set (hash , newElement) ;
    }

    void delete (int E) {
        int hash = E % M ;
        Node temp = ar.get (hash); Node prevNode;
        while ( temp != null) {
            if ( temp.element == E) {
                if (prevNode == null)
                {  ar.set (hash, temp.next);
                }
                prevNode.next = temp.next;
    }
```

Date
09.06.19

CLASS - 8

Problem Given a list of strings, store them in a hashmap using a hash function, such that there are minimum number of collisions

- length of the string

{abc, cba} $h(x) = strLength \% M$

- sum of all characters

{abc, cba} $h(x) = \left(\sum_{i=0}^{N-1} Ci\right) \% M$

- sum of (Index * ASCII) values

Collisions

{ab, ba} {abcad, abcda} $h(x) = \left[\sum_{i=0}^{N-1} ((i+1) * Ci)\right] \% M$

- sum of ($Index^2$ * ASCII) values

{abxzg, abygx} $h(x) = \left[\sum_{i=0}^{N-1} ((i+1)^2 * Ci)\right] \% M$

Ex: abc
0 1 2

$1*a + 2*b + 3*c$

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

- sum of (Prime No's i * ASCII) values

$$h(x) = \left[\sum_{i=0}^{N-1} (P_i * Ci)\right] \% M$$

$4^2 * 2 \Leftrightarrow 2^2 * 8$

$$h(str) = \left[\sum_{i=0}^{N-1} (str(i) * p^{(i+1)})\right] \% M$$

where, p is a prime number

An exponential function will have less collisions than the remaining hash functions

Inbuilt Hashing Libraries

Map / Set → Sorted or Unsorted

<K,V> <K>

* Sorted Map/Set : BBST { S/I/D - $\log_2 N$; size() - 1

* Unsorted Map/Set : Hashing Techniques { S/I/D - O(1) Avg case ; size() - 1

Problem : Finding Frequency

6 ways are discussed in class 5

way 7 use an unsorted hashMap to store the keys & their frequencies.

N+Q , N

Average case

Problem

arr: 5 -1 2 8 15 -6 10

K = 20, arr(i) + arr(j) = K, i ≠ j

1) N^2, 1

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

3) $N\log_2 N + N\log_2 N$, N

N [binary search] b = k - a — Obtain; Check

NOTE: Space complexity for sorting algos

~ $N\log_2 N$, 1

Space O(1) → 1 (Quick Sort)

→ N (Merge Sort)

Way 4 Quicksort O(1) ⟶ $O(\log_2 N)$

Solution Use Hashing : N+N , N

a + b = K

b = K - a

HashMap <K,V> : support duplicates by checking frequency

{ (5,1) (8,1) (10,1)
(-1,1) (15,1)
(2,1) (-6,1) }

check for 'b' in the hashMap everytime

Way 5 Use sets to solve the problem

N , N

arr: 5 -1 2 8 15 -6 10

15 21

set <K>: { 5, -1, 2, 8 }

a	b
5	15
-1	21
8	18
15	5 Found

Search for 15 in < i-1

K = 20 Insert into the hash set

TODO a + b = K
a - b = K
a + b + c = K

Problem

arr: 5 7 -1 -15 8 2 -5 7 4 -1 10 -45 3 12 -1 2 -3

Find the max sub-array sum.

8 2 -5 7 4 -1 10 = sum = 25

Number of subarrays = ~~2^N~~ $\frac{N(N+1)}{2}$

Ex 1 2 3 4
0 1 2 3

subarrays = { 01, 012, 0123, 0
12, 123, 1
23, 2, 3 } = 10

0 1 2 3

4+3+2+1 = 10

TIP The concept of subsets are applicable on sets but not on arrays. We can't apply it here as it contains duplicates.

TODO subsets Vs subArrays Vs subsequence

NOTE

SubArrays : $\frac{N(N+1)}{2}$

Subsets : 2^N

Subsequences : $2^N - 1$ (-1 if we are not allowed to have empty subsequence)

Problem: Find the number of non-decreasing subsequences in the given array.

i.e curr_Ele <= nextEle

ar_N: 5 7 -2 10 3 15 7

{5, 7} {7, 7}

{5, 7, 10}

{5, 7, 15}

{5, 7, 7}

generate subsets such that $a[i] < a[j]$ & $i < j$

int[] a, int N, int max_sum

Old Sol 1

```
int solve (int[] ar, int N) {
    c = 0;
    for (int i = 1 to 2^N) {
        if (valid(ar, N, i)) {
            c++;
        }
    }
    return c++;
}
```

```
bool func Valid (int[] ar, int N, int i) {
    P = Integer.MIN_VALUE;
    for (int j = 0 to N) {
        if (CheckBit(i, j)) {
            if (ar[j] < P)
                return F;
            else
                P = ar[j];
        }
    }
    return T;
}
```

Solution

```
int solve (int[] ar, int N, int cur_idx, int prevE) {
    if (idx == N) return 1;
    return solve(ar, N, cur_idx+1, prevE)
           +
    (ar[idx] >= P) ? solve(ar, N, idx+1, ar[idx]) : 0;
}

main fun() {
    ans = solve(ar, N, 0, Integer.MIN_VALUE) - 1;
}
```

solve(ar, N, cur_idx+1, prev)

↓

not considering the current Element

ar[idx] >= p ? Then pass the current value & add it to the count; otherwise 0

Problem Cont...

Find the subarrays; max sub-array sum.

```
int solve (int[] ar, int N) {
    int ans = 0; int Integer.MIN_VALUE;
    for (int i=0 to N) {
        int sum = a[i];
        for (int j = i+1 to N) {  sum=0;
            sum += a[j];          for (k = i to j) {
            if (sum > ans) {          sum += a[k]; }
              ans = sum;          ans = Max(sum, ans);
            }}
            sum = sum + a[j]; }
        }
    }
    return ans;
}
```

* Time complexity : $O(N^3)$, 1

way 2

```
int solve (int[] ar, int N) {
    int ans = INT_MIN;
    for (int i=0 to N) {
        int sum = 0;
        for (int j = i to N) {
            s = sum + ar[j];
            ans = Max (sum, ans);
        }
    }
    return ans;
}
```

* Complexity : $O(N^2)$, 1

Problem

Given an array of non-repeated nos, rearrange the subarray such that the length of subarray is maximum.

arr : 5 8 10 12 11 9 -15 -13 -14 6 3 2 1 4 16 19

8 9 10 11 12 -15 -14 -13 1 2 3 4

[5] [3] [4]

Max rearranged subarray length = 5

way 1 Brute force : generate all combinations of subarrays & sort each of them to check if they are contiguous.

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

generate all subsets ↓ sorting ↓ check if they are contiguous

copy the elements to temp array for sorting

way 2 $N^2 (N + N)$, N → space for temporary array

generate all subsets ↓ N shifts ↳ check if they are contiguous

Approach : CARRY FORWARD

Ex : -5 8 12 11 -4 -3

subarray starting with -5

temp = [-5] ✓ 8 insertion [-5, -4, 8, 11, 12] ✓ -3

[-5, 8] ✓ 12 [-5, -4, -3, 8, 11, 12]

[-5, 8, 12] ✓ 11 Now check if temp is contiguous

[-5, 8, 11, 12] ✓ -4

way 3
- Generate all subsets
- For each subset, perform insertion sort in a temp array.
- Let say N = length of subarray which is already sorted & find min (ar_i) & max (ar_i)
- check if (min(ar_i) + N - 1 = max(ar_i))
- then it is a contiguous sub-array

```
for(int i = 0 to N) {
    for(int j = i to N) {
        int Max = INT_MIN or ar(i)
        int Min = INT_MAX or ar(i)
        for(k = i to j) {
            Max = MAX(Max, ar(k))
            Min = MIN(Min, ar(k))
        }
        if (Max - min + 1 == j - i + 1){
            ans = MAX(ans, j-i+1);
        }
}
```

* Complexity : $O(N^3)$, 1

way 4 Optimize the inner loop

```
for(int i = 0 to N) {
    int M = INT_MIN, m = INT_MAX;
    for(int j = i to N) {
        M = MAX(M, ar(j));
        m = MIN(m, ar(j));
        if (M - m + 1 == j - i + 1) {
            ans = MAX(ans, j-i+1);
        }
    }
```

* Complexity : $O(N^2)$, 1

Pseudo code for way 2 :

```
bool checkIfContiguous(int[] temp, int N)
{
    for(int i = 0 to N) {
        if(a[i] != a[i+1] - 1)
            return F;
    }
    return T;
}
```

Problem Given an array of repeated numbers find the max sum of a subarray.

ar_N : -5 8 12 11 8 7 10 12 11 9 8 15 14 17 15 16 -3

[6] [3]

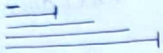
subarray should not consider repeated numbers as it doesn't satisfy the contiguous property

way 1 $N^2(N + N\log_2 N + N)$, N (Refer prev problem)

way 2 $N^2(N + N)$, N (Refer prev problem)

Way 3 Optimized Solution

Ex: -5 8 12 11 8 7 10 12 11 9



Repeated Element

- Stop considering the subarrays generated with starting element -5
- As all the subsets further would contain duplicates

```
Set<k> keys  // extra space
for(int i=0 to N) {
    keys.add(a(i));
    for(int j=i to N) { ----
        ....
        -----
```

(or) linearly iterate all the elements to check if it is repeated.

Now, we can use MIN & MAX logic to find the max sum subarray.

* Complexity : $N^2(N+1)$, 1

```
for(int i=0 to N) {
    int M = INT_MIN, m = INT_MAX;  ⟶ ①
    for(int j=i to N) {
        if(isContains(ar(j)) {  ⟶ ③
            break;
        }
        M = MAX(M, ar(j));  ⟶ ②
        m = MIN(m, ar(j));
        if(M-m+1 == j-i+1)
            ans = MAX(ans, j-i+1);
    }
}
return ans;

bool isContains(int[] ar, int N) {
    for(int i=0 to ar.length)
        if(a(i)==N)
            return T;
    }
    return F;
}
```

* Complexity : $O(N^2)$, N

① ⟶ Define HashSet

② ⟶ Add Elements

③ ⟶ Check if ar(j) ∈ HS

Problem Given an array find the max sum subarray such that the elements can be rearranged in non-decreasing order.

NOTE The non-decreasing order can contain duplicates

TODO

Problem Given an array of repeated elements find the number of distinct elements present in a given window size.

ar_N: 10 -3 8 -3 7 8 -1 -1 7 8 1 8 7 -10 15

K = 5

(4)

(3)

(4)

Way 1 Brute force: Iterate 2 loops; inner loop with window size k = 5
Find distinct elements by maintaining a HashSet<Integer> & count (optimal)
Print the no of elements in Hashset

```
for (int i=0 to N) { N-5) {
    HashSet<Integer> set;
    for (int j= i to i+5(K)) {
        if ( !set.contains (ar(j))
            set.add (ar(j))
    }
    S.O.P (set.size() + " ");
}
```

★ Complexity: (N-K+1) * K , K

No of subarrays of window 'k' ↓ 'K' insertions of elements to HS ↳ extra space for HS

★ Steps for CODING:

- Understand the problem stmt
- Read the i/p & o/p format
- Read the constraints & figure out datatypes & solution approach
- Think of approaches to solve the Q
- Trace the solution & validate the correctness
- Time & Space Complexity
- TC ≤ 10^8; SC ≤ 10^{6-7}; If doesn't work goto S4
- Figure out critical conditions; Revisit S3
- Think about corner/edge cases
- Start coding

Way 2 Try to modify the hashset instead of declaring it everytime

```
for (int i = 0 to N-5) {
    int temp = a(i);
    if ( set.contains (temp))
        delete HS (temp);
    for (int j = i to i+K) {
        if ( !set.contains (ar(j))
```

Use a HashMap to maintain the frequency of each element in a window.

(10, 1)
(-3, 1) → (-3, 2)
(8, 1)
(7, 1)

F(4)

delete (i-K)th element
insert ith element

HashMap<Integer> map =
Int, Int

Problem Given 2 arrays with no duplicate elements in individually, find A ∪ B and A ∩ B

arr N : 10 3 5 6 12

arr M : 3 6 19

Approach : Use Hash Map to maintain frequency.

★ STRINGS ★

ASCII Values

A - 65	a - 97	0 - 48
B - 66	b - 98	1 - 49
:	:	:
Z - 90	z - 122	9 - 57

TODO Print the ascii values from 0 to 127.

Problem Given a string, print the frequency of each char.

Str = a b y z x x c d a e a a b c x

Approach : Use a HashMap To count the frequency

```
int count[26];
Count[ str(i) - 97]++;
        'or'
count[ str(i) - 'a']++;
```

Problem Given a string, find the first repeating character

str N : m x b c a d b y z x

Q → first repeating could be → x, b

Type 1 : First repeating character (x)

Brute force : check for all the characters

```
for (int i = 0 to N) {
    for (int j = i+1 to N) {
        if (str(i) == str(j)) {
            Print str(i)
            break;
        }
    }
}
```

Type 2 : First repeating character (b)

Maintain Hash Map ; if frequency > 1 print the ele

```
int count[26];
for (int i = 0 to N) {
    if (count.contains(str(i))) {
        Print (str(i));
        break;
    }
    count.add(str(i) - 97));
}
```

Soln 1 find b

bool count[26];

Soln 2 find x

int count[26];

Date 15-06-19

CLASS - 9

Problem Given a string of length N, find the length of the longest palindrome substring.

str : a x b c x c b y y b c x 3 3 x c a a c a d

Way 1 : start from first char, move to left & right to check if it is a palindrome. Maintain count for every sub-string.

$O(N^2)$, $O(1)$

Brute Force : Generate all combinations of sub-arrays & check if it a palindrome

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

```
for (int i = 0 to N) {
    ptr1 = i-1; ptr2 = i+1;
    if (str(i) == str(ptr2))  // even length case
        ptr2 = i+2;
    while (ptr1 >= 0 && ptr2 < N &&
            str(ptr1) == str(ptr2)) {
        ptr1--;
        ptr2++;
    }
    count = Math.max(count, ptr2-ptr1+1);
}
```

Complexity : N x N + N x N , 1

Even length centers (0,1) (1,2) (2,3) . . . → N-1

Problem Given two strings an & bm find the substring of 'a' such that it contains all the characters of 'b'

an: a x y q u z e y q e u a c e u q d x e u b d a b

[queue] length: 9 [queue] length: 8

bm: queue

way 1: Maintain hash for all characters in 'b'

Generate all substrings of an & search if contain all characters of bm

~~$O(N^2 * M), M *$~~

generate substrings ↵ ↳ compare count table or replace it with '.' character

way 2: $O(N^2 \times (M \times N)), N$

↳ copy of substring to replace the visited characters with '.'

way 3: cont...

q → 'a' | (index: q-'a')
u → 2
e → 2 CB

$\forall_{i=0}^{25}$ CA(i) ≥ CB(i) [compare count of B chars in A]

Complexity: $O(N^2 * (N+26) + M), O(26+26)$

way 3:

step1: $\forall_{i=0}^{M-1}$ CB[B(i)-97]++;

step2:
```
for(i=0 to N) {
    for(j=i to N) {
        CA[A(j)-97]++;
        if(Match(CA,CB)){
            ans = MIN(ans, j-i+1);
        }
    }
}
```

complexity: $O(N^2 * (1+26)) + M), 26+26$ (CA, CB)

way 4: Use binary search to find the minimum substring

Ex:- x a q c d e u x e d u a b c

min length = 9

Given length = 10 [a to u]

ans = 10✓, 11✓, 12✓

ans = 8✗, 7✗, 6✗

low = M and high = N (possible ans values)

```
int low=M, high=N, ans=-1;
~~while~~( ∀_{i=0}^{M-1} CB[B(i)-97]++;

func isValid(string a, N, countArray of B, mid)
{
    for(int i=0 to mid)
    { // ∀_{i=0}^{25} CA(A(i)-97)++;
      // compare the count arrays of A & B
```

```
for(int j = mid to N){
        CA(A(j)-97)++;
        CA(A(j-m)-97)--;
        25
        ∀ compare the count Arrays
        i=0
}
```

Complexity : $M + N * \log_2(N-M+1)$, $26+26$

```
while(low < high)
{   mid = (low+high)/2;
    if (isValid(a, N, CB, mid))
            ans = m;
            hi = m-1;
    else
        low = mid+1;
}
```

Way 5 Optimize further by considering way 3.

Lets say, str = a x b q u d e y z u k e a b c

generate substrings & compare with CB.

a x b q u d e y z u k e a b c

CA(a) → CB(queue) compare

CA(ax)

CA(axb)

⋮

CA(axbqudeyzuke) → isValid() returns true;

CA(a- - - - - a) → break the loop as the substring is found

Now move to next subarray,

CA(x)
CA(xb)
CA(xbq)
⋮
CA(xbq- - - - -e)

} Not required to generate all these combinations as we already know xbq- - -e is valid.

So, maintain 2 pointers & find the min length

a x b q u d e y z u k e a b c

ptr1 ↑ x ↑ ptr2

ans = 12, 11, 16, 9

for every ptr1++; check for isValid() using count arrays. when ptr1 is at 'u' the isValid() returns false. So break the loop & update the ans as '10'

P1 : N * 1

P2 : N * (1+26)

Complexity : $(N*1) + (N*(1+26)) + M$, $26+26$

$\approx O(N), O(1)$

Problem Given a string 'b' find if it is present in string 'a'.

// substring problem.

strA : a b c q u e u e x y b

strB : q u e u e

Bruteforce : compare strB with strA (substring's of length M)

a b c q u e u e x y b

substrings of window size '5'

Complexity : (N-M+1) * M, 1

★ String Matching Algorithms

- Rubin Karp
- KMP (search on Topcoder Tutorials)

★★ **RUBIN KARP**

h(B) = q+u+e+u+e

h(A) = a+b+c+q+u (first substring)

↳ = b+c+q+u+e

↳ = c+q+u+e+u

↳ = q+u+e+u+e

Compare at each level O(1)

Update the h(A) optimally by avoiding recomputation of every substring

Complexity : N+M, 1

h(B) → M

h(A) → M

Comparision → (N-M) substrings

M+M+N-M = **O(N+M), 1**

★ Good hash function for strings

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

where p is a prime number (11, 31, 47, - -)

$h(B) = q*p + u*p^2 + e*p^3 + u*p^4 + e*p^5$

$1h(A) = a*p + b*p^2 + c*p^3 + q*p^4 + u*p^5$

$2h(A) = b*p + c*p^2 + q*p^3 + u*p^4 + e*p^5$

$$\Rightarrow \frac{1h(A)}{p} - a + (u*p^5)$$

- generate hash value for B : h(B)

```
int p = 11; hashB = 0
for (int i = 0 to M)
    hashB = hashB + (str(i) * Math.pow(p, i));
```

- Generate next sequence of hash value for A : h(A)

```
for (int i = M to N)
    hashA = hash(A)/p - str(i-M) + str(i)
            * Math.pow(P, M);
if (hashA == hashB) return true;
```

Lets say $N = 1000$

$M = 200$

$p = 2$

then p^M i.e 2^{200} will overflow, so update the code accordingly.

$h(B)$: $hB = \left[hB + B[i] * \underbrace{pow(p, i+1)}_{\text{overflow}} \right] \% M$

$h(A) = \left[\frac{h(A)}{p} - A[i-M] + A[i] * pow(P, M) \right] \% M$

As, $\%K$ is not applicable for division, so use inverse modulo or a diff hash function which doesn't involve division.

* updated hash function :

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

$h(A)$: $a*p^{11} + b*p^{10} + c*p^9 + q*p^8 + u*p^7$

↳ $b*p^{11} + c*p^{10} + q*p^9 + u*p^8 + e*p^7$

↳ $c*p^{11} + q*p^{10} + u*p^9 + e*p^8 + u*p^7$

$\Rightarrow h(A) = (h(A) - a*p^5) * p + u*p^1$

```
for(int i = M to N)
    h(A) = [(h(A) - str(i-M) * Math.pow(p, M)) * p
            + str(i) * p ] % M ;
```

$h(A) = \{ [h(A) - A[i-M] * \underbrace{pow(P, M)}_{\text{overflow}}] * p + A[i] * p \} \% K$

consider the % carefully while subtraction

$h(A) = (h(A) + K) \% K$; where $K = 10^9 + 7$

```
int pow[M+1], p = 11, K = 1e9+7;
pow[0] = 1;
```

step1 : $\forall_{i=1}^{M}$ $pow[i] = (pow[i-1] * p) \% K$

step2 : $HB = 0$

$\forall_{i=0}^{M-1}$ $HB = (HB + B[i] * pow[M-i-1]) \% K$

step3 : $HA = 0$

$\forall_{i=0}^{M-1}$ $HA = (HA + A[i] * pow[M-i-1]) \% K$

step4 : if (HA == HB)
    return true;

step5 : $\forall_{i=M}^{N-1}$ $HA = [HA - A[i-M] * pow[M] + A[i]] * p$

$HA = [HA \% K + K] \% K$;

complexity : $M + M + M + 1 + (N-M)$, M ↳ pow Array

$O(N+M)$, $O(M)$

Other version, to avoid collisions, use H1 & H2

Rolling Hash Method

if (h(A)$_1$ == h(B)$_1$, && h(A)$_2$ == h(B)$_2$)
    return true;

compute & pow$_1$[i] using $P_1 = 11$
pow$_2$[i] $P_2 = 31$

Problem Find the sum of the indexed elements between the given indicies.

	0	1	2	3	4	5	6	7
arr :	3	5	-1	8	-7	2	12	4

Q : i, j such that $0 \le i \le j < N$

Ex: 2, 5 → 2
4, 6 → 7

Brute force : $O(Q * N)$, 1

$$\sum_{k=i}^{j} arr(i)$$

way 2 : $b(j) = \sum_{j=0}^{i} arr(j)$ // This is called Prefix sum

b(6) → 0 to 6 elements sum
b(2) → 0 to 2

i j : b[j] - b[i-1] : i ≠ 0
: b[j] : i = 0

⇒ i, j : PS[j] - PS[i-1] ; i ≠ 0
PS[j] ; i = 0

PS(0) = ar(0)

$$\forall_{i=1}^{N-1} \; PS(i) = PS(i-1) + ar(i)$$

complexity : $O(N+Q), O(N)$

⇒ i, j : PS(j) - PS(i) + ar(i)

Problem Find the longest palindrome

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
arr :	a	b	x	y	z	t	u	t	z	y	x	x	y	z	u	z	u	z	a	a

N = 20

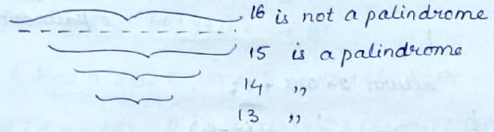
Brute force : $O(N^3)$, 1

way 2 : $O(N^2)$, 1

way 3 : str = reverse (str)
h(str) == h(reverse(str))

generate subarrays & compare

Let say, length = 15 is a palindrome



ans = 20, 21, 22, 23
ans = 19, 18, 17, 16

use binary search to find the longest palindrome

Ex: y a b c a c b y

center,

we can expand to 2 characters on both sides

mid ⇒ length to which can expand chars from the center.

For any center, low = 0, high = MIN (left, right)

we can expand only to 4

left side no of elements ✓ center ↘ right side no of elements

Therefore, high = MIN (10, 4) = 4

```
{
    low = 0, high = MIN(?, ?), ans = 0
    while (low ≤ high) {
        mid =
        if (isValid (str(i-m) to str(i+m))
            ans = mid
            low = mid + 1;      // Look for bigger palindromes
        else
            high = mid - 1;     // Look for smaller palindromes
    }
    return 2 * ans + 1;
}
```

$(c_2 - c_1)$

The above binary search method is called for every char as center in the given string, thus updating the high value;

Use 2ptr technique to implement: isValid (str (i-m : i+m))

Complexity : $O(N * \log_2 N * N)$, 1

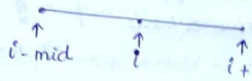
2 ptr technique to compare & check if it is a palindrome

⇒ $O(N^2 \log_2 N)$, $O(1)$

Optimize the isValid (str) method

Pattern : T T T T T . . . (T) F F . . .

min length — max length



Pre compute the hash values of the given string i.e Prefix Sum Hash

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

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

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

```
func isValid (x, y) {   // i-m, i+m
    f = fh(y) - fh(x-1);    ⇒ fh(y) - fh(x) + str(x) * p^x
    fR = bh(x) - bh(y+1);   → (2)
    // Find a way to compute reverse str hash
    if (f == fR)   // i.e h(str) == h(reverse(str))
        return true;
}
```

$$\forall_{i=N-2}^{0} bh(i) = \sum_{j=N-1}^{i} str(i) * p^{N-i}$$

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

$\Rightarrow$ bh(N-1) = str(N-1) * p

Ex: fh = $y*p^4 + 3p^5 + \cdots y*p^{10}$

bh = $y*p^{11} + z*p^{13} + \cdots y*p^{17}$

Multiply fh by 10^7, but can't do $bh/10^7$ as there is a limitation with % in case of handling overflow scenarios

The smallest power of prime in backward hash } = y i.e N-i $\rightarrow$ N-y

The smallest power of prime in forward hash } = x+1

spfh = x+1 ; spbh = N-y

d = |spfh - spbh| ;

if (spfh < spbh)

fh = fh * p^d

else

bh = bh * p^d

if (fh == bh)

return true;

NOTE: Wherever there multiplication or Pow take care of overflow scenarios.

Handle %,K in case of '-' subtraction carefully.

The above discussed method is for odd length palindrome

For even length, update the x, y values based on length

~~if (str(i) == str(i+1))~~

if (mid%.2 == 0)

x = i-m ;

y = i+1+m ;

else

x = i-m ;

y = i+m ;

// if (isValid (x,y)) - - - -

Also, the high value should be updated as

high = min ($\frac{c_1}{}$, $\frac{N-c_2-1}{}$)

int BS (int c1, int c2, - - -) { . . }

low = 0

high = min (c_1, $N-c_2-1$) ; - - -

isValid (c1-m, c2+m) ;

Complexity : N+N + N + $2N * \log_2 N * 1$, N+N+N

bh, Pow(i), fh, Pow(i), fh, bh

$\hookrightarrow$ isValid Method

Binary search

$\hookrightarrow$ odd / even

* Way 3 uses : Rolling Hash

Binary search

Prefix Hash

Modular arithmatic

Precomputation

Way 4 : Manacher's Algorithm → O(N), O(N)

Tutorial on Leetcode

TODO

Problem Given 2 strings & indicies find if strA(i,j) == strB(k,l)

ex : a b c d e f a b c

bx : x y z a b c x y z

Q : $0 \le i \le j \le n$

$0 \le k \le l \le m$

substr(a, i, j) == substr(b, k, l)

Way 1 : Brute force Q x N, 1

Way 2 : Similar to previous problem solution

hA(j) = hfA(i-1)

hB(l) = hB(k-1)

handle for i=0 & k=0 cases separately

spA = i+1

spB = k+1

(sp : smallest power of prime)

Complexity : N+M+Q x 1, N+M

↳ hash of A & B

Date 16.06.19

CLASS - 10

Problem given a string check / find the max length of palindrom

Way 4 : Cont... (Refer to previous class)

EVEN → 0, 100 → 50, 48, 46 ....

Make sure the mid is always an even no

ODD → 1, 99

Here, we are performing binary search on the length of the palindrome.

a b c x c b a z b

Let's say mid = length (4)

search all window sizes of length 4 in the given string.

Then update the low to (mid+1) once we find a palindrome of length mid to continue searching for palindrome of bigger lengths.

Complexity : $N + N + (N-M+1) + \log_2 N$, 1

Problem Given a number N, check if its a prime or not

```
bool isPrime(int N) {
    for(int i=2 to √N)  ⟶ ①
        if(N%i==0)
            return false;
    return true;
}
void main() {
    if(N<=1)
        print "NO"
    S.O.P(isPrime(N));
}
```

① To avoid computation of $\sqrt{N}$ everytime, initialize it before loop or use, for(int i=2 to i*i≤N)

Complexity: way 1 : $\sqrt{N} * \log_2 N$, 1
↳ computation of $\sqrt{N}$ everytime in loop

way 2 : $\sqrt{N}$, 1

Problem Given a number N, compute all the primes ∈ [0, N] count

N = 10 ans = 2, 3, 5, 7 → 4

N = 17 ans = 2, 3, 5, 7, 11, 13, 17 → 7

way 1 : Brute force
```
for(int i=0 to N)
    if(isPrime(i))
        c++;
```

Total no of iterations = $\sqrt{2} + \sqrt{3} + \sqrt{4} + \sqrt{5} + \dots \sqrt{N}$
= Sum of square root of first N nos

TODO Complexity:

way 2 : Initialize a boolean array of size N+1 marked all as true.

ar: ∅ ✗ 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

Start from 2 : ar[2] = true
Cancel / Mark all the multiples of 2 as false
3 : ar[3] = true
Mark all the multiples of 3 as false
5 : ar[5] = true

How How do we know if 5 is a prime or not ?
If it's not, it must have been marked as false by one of numbers ≤ 5.

By continuing, we will be left with only primes marked as true.

```
ar[N+1];
for(int i=0 to N+1)
    if(ar[i]==true) j=i*2;
    while(j<N) ar[j*i]=false; j++;
```

No of iterations = $N/2 + N/3 + N/5 + N/7 + \dots$

$= N\left[1/2 + 1/3 + 1/5 + 1/7 + \dots + 1/x\right]$

x = highest prime less than or equal to N

TODO Complexity :

way 3 : Optimize the above approach. This method is called

SIEVE OF ERATOSTHANESE

```
int primes(int N) {
    bool p[N+1] = {T};
    int c=0;
    p[0] = p[1] = F;
    for (int i=2 ; i*i<=N ; i++)
        if (p[i]==T) {
            ~~c++;~~ → ①
            for (int j = i*i ; j<=N ; j+=i)
                p[j] = F;
    return c;
}
```

Observe the limits of i & j ; The drawback is we miss counting the primes as the $i \le \sqrt{N}$. So, iterate again over p[] & count.

① → `for (int i = 2 to N)`
`if (p[i] == T)`
`c++;`

Problem Given 2 numbers, find the count of primes between them

$1 \le a \le b \le 10^{10}$

$b - a \le 10^5$

Ex: [3, 20] : 7

[54, 200] : —

Method :

SEGMENTED SIEVE

a = 54

b = 200

start marking p[200-54+1] starting primes = 2 to $\sqrt{200}$

P = 2, 3, 5, 7, ..., = 2 to $\sqrt{b}$

L = { 2, 3, 5, 7, 11, 17, 19, ... } compute till $\sqrt{b}$

```
bool p[b-a+1] = {T}
for (int i=0 ; L(i) <= √b ; i++) {
    pr = L(i)
    x =          // First multiple of pr in range [a,b]
    for (int j=x to N)   // j = j+pr for every iteration
        ~~p[j] = F~~
        p[j-a] = F ;   // As the number is 54 & index 0
                       should be marked as false
}
~~for (int i=a to b)~~
    ~~if (p[i]~~
for (int i = 0 to (b-a+1))
    if(p[i] == T)
        c++;
```

TODO Problem in SPOJ, esolve using this approach : PRIME 1

Problem Given an array, find the subset that the sum = k

ar_N : 5 3 -1 8 -5 12

K : 24

Limits : $1 \le N \le 40$

$-10^9 \le ar \le 10^9$

way 1 : As discussed in CLASS 3,

$2^N \times N$, 1

way 2 : 2^N, 1

```
find subsets (int [] a, int N, int idx, int sum) int k)
{
    if (idx == N)
        return sum == K;
    return subsets (a, N, idx+1, sum+ar[idx],k)
        || subsets (a,N, idx+1, sum , K);
}
```

Using iterative, use the logic of generating 2^N combinations by checking bits for each number $< 2^n$

```
for (int i=1 to 2^N) {
    sum = 0;
    for (int j to 32)
        if (checkBit (a[], j, N)
            sum += ar[j]
    if (sum == K)
        return true;
}
return false;
```

way 3 : ar_N : 5 3 -1 | 8 -5 12 ($2^{20} + 2^{20}$)

0 0 0 → 0		0 0 0 → 0
0 0 1 → 5		0 0 1 → 8
0 1 0 → 3		0 1 0 → -5
0 1 1 → 8		0 1 1 → 3
1 0 0 → -1		1 0 0 → 12
1 0 1 → (4) = a		1 0 1 → (20) = b
1 1 0 → 2		1 1 0 → 7
1 1 1 → 7		1 1 1 → 15

Divide the array to 2 halves & generate subsets & calculate sum

Time to generate subsets ⇒ $2^{N/2} + 2^{N/2}$

Problem to solve ⇒ $a + b = K$ (Refer to previous class, there are 5 ways to solve, validate which all ways are applicable

Complexity : $2^{N/2} + 2^{N/2} +$

a) $2^{N/2} * 2^{N/2}$ (Brute force)

b) $2^{N/2} \log_2 2^{N/2} * 2 + 2 * 2^{N/2}$

$\Rightarrow N * 2^{N/2} + 2 * 2^{N/2}$

⇒ sorting + 2ptr technique

c) Binary search ($b = K - a$) after sorting

$2^{N/2} \log_2 2^{N/2} + 2^{N/2} * \log_2 2^{N/2}$

$\Rightarrow N/2 * 2^{N/2} + N/2 * 2^{N/2}$

d) Hash ~~Map~~ Set <key, ~~value~~> = <Sum, ~~Count~~>

$x \in A$, $y \in B$

↓ ↓

HashSet y search for $x = K - y$

$\Rightarrow 2^{N/2} + 2^{N/2} + 2^{N/2}$ → space for Hash set

↓ store sum values in Hashset A

↳ fix each element in B & search in Hashset A

Conclusion: Except a) we can use any of the remaining 3 {b, c, d} approaches, for larger values of N

MEET IN THE MIDDLE

2^N / $2^{N/2}$ $2^{N/2}$ — N^N / $N^{N/2}$ $N^{N/2}$ = N^2

Ex $a+b+c+d = k$

$\frac{a+b}{N^2} = k - \frac{(c+d)}{N^2}$

fix the elements in N^2 & search in $O(1)$ times.

GAME THEORY

- 2 player games
- A Vs B
- Alternate Turns
- WIN Condition

Problem There are N coins, 2 players, pick coins one by one & who ever is left with no coin will loose the game.

N coins: 1, 2, 3

14 — ① C

15 — 13 — ② C

12 — ③ C

10 → 9 → 7 → 4 → ③ ② ①

(A,B) B A B

(1) (2) (3)

A wins the game

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P:	W	W	W	L	W	W	W	L	W	W	W	L	W	W	W	L	W

- If atleast one outcome is 'L' then the current turn outcome is 'W'.
- If all 3 outcomes are 'W' i.e the other player is going to win & we are going to loose. So, the outcome of current turn is 'L'.

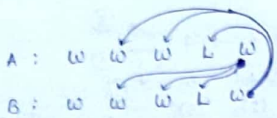
* L : ∀ moves goes to W
* W : ∃ move goes to L

```
if (N%4 == 0)
    A → L
else
    A → W
```

In this case, both A & B are playing optimally.

Problem Assuming that B plays greedily & A plays optimally, find the pattern for above problem.

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A:	W	W	W	L	W	W	W	W	W	W	W	W	W	W	W
B:	W	W	W	L	L	L	W	L	L	L	L	L	L	L	L



- If greedy starts the game, the optimal will loose only when N = 1, 2, 3 and 7

Problem Assuming that both A & B plays the game greedily in the above game, find the winning strategy.

N :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P =	W	W	W	L	L	L	W	W	W	L	L	L	W	W	W

```
if (N%6 == (1,2,3))
    A/B → W
else
    A/B → L
```

a) * Optimal Vs Optimal

b) * Optimal Vs Greedy

c) * Greedy Vs Greedy

Problem Moves = 1, 2, 4, 8, 16, 32, 64, 128 ... 2^x

Find the pattern for all 3 ways (a, b, c)

a)

N:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P =	W	W	L	L	W	W	W	W	W							
P =	W	W	L	W	L	L	L	W	L	L	L	L	L	L	L	W
P =	W	W	L	W	W	L	W	W	L	W	W	L	W	W	L	W

b)

N :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A =	W	W	L	W	L	L	W	W	L	L	W	L	L	L		
B =	W	W	L	W	W	W	W	W	W	W	L	W	W	W		

As there is no specific pattern observed, precompute the outcome for each value of 'N'

- Precomputation → $N \log_2 N$
- Testcases → T

N = 15, 20, ...

$1 \le T \le 10^6$ & $1 \le N \le 10^6$

Complexity = $O(T + N \log_2 N)$, $O(N)$

(T * N) Vs (T + N)

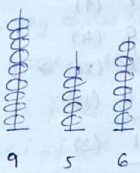
* Importance of precomputation :

when 'N' limits are large, precompute the results of each N value.

can be used for,

- computing factorial
- N sequence of fabonacci numbers
- game theory patterns of winning strategies

Problem N - piles



Ex: 9 ^ 5 ^ 6 : 1001 ^ 0101 ^ 0110 = 1010

A 1 5 6 : 0001 ^ 0101 ^ 0110 = 0010

B → 1 5 4 : 001 ^ 101 ^ 100 = 000

A 1 2 4 : 001 ^ 010 ^ 100 = 111

B → 1 2 3 : 001 ^ 010 ^ 011 = 000

Ex: Turn : 2 4 1

```
  010
→ 100   (4)th pile
  001
  ---
  111   → Make the result '0' by changing
          the bits of same row.
```

```
  0 1 0
  0 ~~1~~ ~~0~~   ⇒ -4 = -(2^2) removed 1
  0 0 1             2 = +(2^1) added 1
  -----             1 = +(2^0) added 1
  0 0 0
```

Total ⇒ -4+2+1 = -1

⇒ Remove 1 pebble from 2nd pile where they are 4 pebbles in last turn.

Ex: Turn

```
  0 1 0 0 1   (9)
  1 0 1 1 1   (23)
  0 1 0 0 1   (9)
  0 0 1 0 0   (4)
  1 1 0 ~~1~~ ~~0~~   (26)
  0 1 0 1 0   (10)
  -----------
  0 0 0 1 1   (3)
```

$+(2^0)$ ↑ $-(2^1)$

⇒ 1 1 0 0 1 ⇒ -2+1 = -1

Remove 1 pebble from 5th pile containing 26 pebbles in last turn.

NOTE: After 1's to 0's starting from most significant bit (MSB) Fix to that row & alter remaining bits such that the resultant XOR is zero.

XOR = 0 : ∀ moves goes XOR ≠ 0

XOR ≠ 0 : ∃ move goes XOR = 0

XOR = 0 → L

XOR ≠ 0 → W

If the XOR of initial 'n' piles is 0, then the person who starts the game ~~wins~~ loses. And if initial XOR ≠ 0, the first player wins.

Other Versions: Stragtegy of picking pebbles can be in form of fabonacii series = 1, 1, 2, 3, 5, 8, ...

NIMS GAME

0 Vs 0 → Above condition

0 Vs G → TODO

G Vs G → if (N%2 == 1) Player1 wins else Player2 wins

Problem: Given an array of size N & elements values '0' find the sum of all elements ~~sum~~ after computing given criteria

Q i j x : $\forall_{l=i}^{j} \; ar(l) = ar(l) + x$, $0 \le i \le j \le N$

Way 1: Brute force : O(Q * N) + N), N

	0	1	2	3	4	5	6	7	8	9	10	11
ar_N :	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
	4	4	~~7~~	~~7~~	~~2~~	~~2~~	-1	~~4~~		~~-2~~	~~-2~~	-2
			7	7	2	2		4	5	3	3	

ar_N : 4 4 7 7 2 2 -1 4 5 3 3 -2

Find the sum of elements = $\sum_{i=0}^{N-1} ar(i) = 38$

way 2 : Q : i j x sum [optimized version]

2	5	3	= 12	$(j-i+1) * x$
4	7	-1	= -4	
0	3	4	= 16	
9	11	-2	= -6	
7	10	5	= 20	
			38	

Complexity : $O(Q)$, $O(1)$

TODO Print the array after computing the query results.

Brute force : $Q \times N$

Date 27-06-2019

CLASS - 11

Problem Contd... Print the array elements after completing the query results.

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

way 3 : Consider 1st query & add the x value at index i & -x

~~x+1~~ at j+1

	0	1	2	3	4	5	6	7	8	9	10	11
	~~0~~	0	~~0~~	0	~~0~~	0	0	0	0	0	0	0
	4		3		~~4~~							
					-5							

$ar(i) += x$

$ar(j+1) -= x \quad j \neq N-1$

Complexity : $Q+N$, 1

* HackerRank - Practise Section

Problem Given an array find the product of all elements except that such that, $\forall_{i=0}^{N-1} B(i) = \prod_{j=0}^{N-1} A(j)$ and $i \neq j$

way 1 : Brute force Complexity : N^2, 1

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

↓

$(5 * -1 * 3 * 10 * -2 * 8 * -6)$

Way 2 : → $P = \prod_{i=0}^{N-1} A(i)$

→ $\forall_{i=0}^{N-1} B(i) = \frac{P}{A(i)}$

This condition fails when $A(i) = 0$, the product itself (P) becomes zero.

Complexity : O(N), 1

Way 3 : If there are multiple zeros in the given array, skip them while computing product P & maintain a count of number of zeros.

→ If there is only one zero, remember its index.

→ If there are multiple zeros, the product would be always zero for every element of B(i)

Way 4 : Avoid the use of division operator by using prefix product & suffix product

ar_N :	5	-1	2	3	10	-2	8	-6	
PP(i)	5	5*-1		5*-1*2*3				-6*8	SP(i)
PP(i+1)	5*-1*2							-6	SP(i-1)

Prefix Product $PP(i) = \prod_{j=0}^{i} A(j)$

$\forall_{i=1}^{N-1} PP(i) = PP(i-1) * A(i)$

$PP(0) = A(0)$

Sufix Product $SP(i) = \prod_{j=i}^{N-1} A(j)$

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

$SP(N-1) = A(N-1)$

compute B(i), $\forall_{i=1}^{N-2} B(i) = PP(i-1) * SP(i+1)$

$B(0) = SP(1)$

$B(N-1) = PP(N-2)$

Complexity : 3N, 2N

↳ prefix product + Sufix product

The space complexity can be further reduced by using one array The prefix product can be computed on the fly.

$\forall_{i=1}^{N-2} B(i) = PP * SP(i+1)$ where $PP = A(0)$

$PP = PP * A(i)$

→ $B(0) = SP(1)$

→ $B(N-1) = PP$

Problem Given an array, rotate it by 'd' positions in clockwise direction. (unsorted array)

ar_N :	-1	8	3	5	-2	10	18	-4
d=1 :	-4	-1	8	3	5	-2	10	18
d=3 :	10	18	-4	-1	8	3	5	-2

If d = 100 then d' = 100% N rotations

solution : check on SI youtube channel

Problem Given a sorted array, perform 'd' clockwise rotations & search for a given key 'K'.

way1 : ar_N : 15 18 20 23 -5 -2 3 5 6 8 12

Given, d = 5 and original ar_N sorted

→ BS (0, d-1)
→ BS (d, N-1)
} Brute force using BS

Complexity : $\log_2 N$, 1

way2 : Perform linear search for 'k'

Complexity : N, 1

Problem In the above problem, say 'd' is not given as i/p There are no duplicates in the given array.

way 1 : Find the position (d)

ar_N : 15 18 20 23 -5 -2 3 5 6 8 12

↑ start of the original array

Perform BS to find 'd'

```
if (a(mid-1) > a(mid) &&
        a(mid) < a(mid+1)
    return mid ;
```

```
if ( a(mid) < a(mid+1) &&
     a(mid) > a(mid-1)) {
        high = mid-1 ;
} else {
        low =
```

Then perform, BS (0, d-1)
BS (d, N-1) to search for key 'K'

2 ways of finding index of 'd'

→ Find the highest element index ; last index of left part
→ Find the smallest element index ; start index of right part

① d_1 = ans + 1

ar_N : 15 18 20 23 -5 -2 3 5 6 8

d_1 ↓ d_2 ↓

L R

if (m ∈ L) : ans = m
low = mid+1

else // (m ∈ R) : high = mid-1

* Initialize ans = -1

ar_N: 1 2 3 4 5
d = 0

② How do we know if m ∈ L or m ∈ R ?

if (ar(mid) > ar(N-1))
then mid ∈ L
else mid ∈ R

we can either compare with ar(0) or ar(N-1)

if (ar(0) > ar(mid))
mid ∈ R
else mid ∈ L

② d_2 = ans

if (m ∈ R) : ans = mid
high = mid - 1

else (m ∈ L) : low = mid + 1

complexity: $2\log_2 N$, 1

* Initialize,

ans = N - 1

arr: 2 3 4 5 1

d = N-1

way2 : Directly perform BS for 'K'

	0	1	2	3	4	5	6	7	8	9	10
arr:	15	18	20	23	-5	-2	3	5	6	8	12

K = 10 — L — R — mid — high

we can say that the elements [mid, high] are always sorted if the mid ∈ R

if (m ∈ R) : K ∈ [ar(mid), ar(high)] : low = mid + 1
else : high = mid - 1;

else // (m ∈ L) : K ∈ [ar(low), ar(mid)] : high = mid - 1
else : low = mid + 1;

TODO Find the logic, if the given array contains duplicate elements.

complexity : $\log_2 N$, 1

Problem given a sorted array, find the first missing integer on absolute scale.

arr: 1 2 3 4 5 7 19 23 54 ans = 6

arr: 1 2 3 4 ans = 5

arr: 3 4 5 7 10 11 13 ans = 1

way1: Brute force

```
for (int i=0 to N-1)
    if (ar(i) ≠ i+1)
        return i+1;
```

Complexity : O(N), 1

way2 : Use BS to find the index,

Approach 1 : last index of L part

Approach 2 : first index of R part

→ LI of L : ans = 4 { ans + 2 (or) ar[ans] + 1

→ FI of R : ans = 5 { ans + 1

if (mid + 1 == ar(mid)) : low = mid + 1; ans = mid;
else : high = mid - 1;

End of while loop, return ar(ans) + 1 (or) ans + 2

* Initialize ans = -1 for (ans + 2)

Problem Given a sorted array containing negative elements, find the first missing integer.

ar_N : -5 -2 -1 1 2 3 5 6 7 ans: 4

TODO

Problem Given 3 arrays of size N, choose 3 elements such that the difference of $max(i,j,k)$ & $min(i,j,k)$ is minimum

a_N : 12 6 5 18 29

b_N : 13 10 18 7 30

C_N : 9 1 25 2 35

$$MIN\ (\ max(a(i), b(j), c(k)) - min\ (a(i), b(j), c(k))\ )$$

Ex: c - 25 b - 10 a - 18

max = 25 min = 10 res = 25 - 10 = 15

a - 12 b - 13 c - 9

max = 13 min = 9 res = 4

way1 : Brute Force

```
ans = INT_MAX;
for (int i=0 to N)
    for (int j=0 to N)
        for (int k=0 to N)
            // compute result
```

complexity : $O(N^3)$, $O(1)$

way2 : Ex: 6, 13, ? [6, 13] ⇒ 9

12, 13, ? 9 → [12, 13] ← 25 (✗)

(✓) closest element

a_N : 12 (6) 5 18 29

b_N : (13) 10 18 7 30

C_N : 1 2 (9) 25 35 // Sorted array

(6, 13) → find ceil(6) or floor(13) in C_N

$x = min\ (A(i), B(j))$

$y = max\ (A(i), B(j))$

ceil(x) || floor(y)

ans = min (ans, y - x);

Complexity : $N\log_2 N + N^2 * \log_2 N$, 1

↓ sorting C_N ↓ fixing 2 elements ↳ find the 3rd element

way3 : Try to fix one element & find out the other 2 elements

a_N : 12 6 5 18 29

b_N : 7 10 13 18 30

C_N : 1 2 9 25 35

a	b	c	ans
(12)	7	✗	11
		✗	10
	9	=	5
10	9	=	3

use 2 pointers for B_N & C_N & calculate ans,
move to next element in A_N only when ans starts increasing

complexity : $2N\log_2 N + N(N+N)$, $O(1)$

sort 2 arrays B & C ↵

↳ 2 pointers for B & C
↳ Fix element in A

way 4 : Sort all the 3 arrays & find the result.

a_N : 5 6 12 18 29
B_N : 7 10 13 18 30
C_N : 1 2 9 25 35

a	b	c	ans
5	7	1	6
		2	5
		9	

complexity : $3N\log_2 N + 3N$, $O(1)$

sort all 3 arrays ↵ ↳

TODO code

```
p1 = p2 = p3 = 0
while (p1 < N && p2 < N && p3 < N)
    -
    -
    -
```

problem Find the length of the longest subarray with equal no of 0's & 1's.

a_N : 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0

way1 : $N^2 * N$, 1 → check for all sub-arrays

way2 : N^2, 1

a_N : 0 0 0 0 1 0 1 1 0 1 0 0 1

$c_1 = 4$ $c_2 = 1$

using carry forward technique & update c_1 & c_2 for all subarrays

way3 : using binary search,

Does it ? Does it follow the pattern of BS

→ T T T (T) F F F F
8 16

→ F F F F (T) T T T

mid = 8, 6, 4 ✓ ✓ ✓
= 16, 18, 19 ✗ ✗ ✗ (length of the subarray)

Ex : 0 0 0 0 1 1 1 1 follows pattern ✓ ✓

Ex : 0 1 1 1 1 0 0 0 doesn't follow the pattern ✗ ✗

Therefore, BS can't be used here.

way 4 : Optimized Approach

- Replace all the zero's with -1 and maintain a prefix sum

arN : 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1 1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

PSN : -1 -2 -3 (-4) -3 -4 -5 (-4) -5 (-4) -5 -6 -7 -8 -7 -6 -7 -6 -5

last -4 to get the max length

x — — — x
equal no of 0's & 1's

- For every unique element in the PSN, find the first & last occurance & calculate length.
- store the prefix sum along with its index

① - Sort the prefix sum values, like N + N log2N , 1

(-4, 3) (-4, 5) (-4, 8) (-4, 9)
↑ ↑
first occurance last occurance

(FI, LI] ⇒ LI - FI = (9-3) = 6

② - HashMap<int, int> ans

<K, V>	
-1, 0	~~2~~
-2 1	~~4~~
-3 2	6 (max_length)
-4 3	
-5 6	

(-3, 4) (-4, 5) (-4, 7) (-4, 9) (-5, 8) ⇒ 2

Complexity : N + N , N

O(N) , O(N)

ans = 0 , HashMap = hs ;
for (int i=0 to N)
if (hs.contains(PS(i)))
ans = MAX(ans, i - hs.get(PS(i)));
else
hs.put(PS(i), i);

JMP Check

arN : 0 1 0 1

PSN : -1 0 -1 0
0 1 2 3

** To fix this problem, insert <0,-1> into the HM

Problem Given a row-wise sorted matrix, find if a given key 'K' is present or not. (column-wise sorted)

$a_{N \times M}$ =
-10 1 6 9
-2 6 13 20
15 21 32 99
32 43 66 101

way1 : N * M , 1 (Traverse all elements)

way2 : N log2M , 1 or M log2N , 1
↳ Perform BS for each row (N * log2M)

way3 :
-10 1 6 9 ↙ 42 > 9 ✓
-2 6 13 20 ↙ 42 > 20 ✓ K = 42
15 21 32 99 ↙ 42 > 99 ✗
32 (42) 66 101 42 > 32 ✓
found 42 > 66 ✗

Skip either row or column
(N) (M)

Complexity: $O(N+M)$, 1

- we started searching for K = 42 from top-right corner. we can't do this search from the remaining corners except bottom-left corner & top-right corners.

Start: ar[0, N-1] or ar[N-1, 0]

Problem Given a N*M completely sorted matrix, find element 'K'

ar_{N*M} :

-10	-8	-5	-4	-3
-2	0	3	8	9
10	15	17	18	19
20	24	27	28	30
33	40	42	99	100

way1: $O(N*M)$, 1

way2: $N\log_2 M$, 1 or $M\log_2 N$, 1

way3: $N+M$, 1 (Top-right or Bottom-left) approach

way4: Find the row in which 'K' might be present

```
for (int i=0 to N)
    if (ar(i)(0) < K && K < ar(i)(i))
        rowId = i
Perform BS in that column 'i'
```

$O(N + \log_2 M)$, 1

Dec 1st week, 2019

CLASS - 12

Problem Contd... Find element 'K' in completely sorted array

way5: $O(\log_2 N + \log_2 M)$, $O(1)$

way6: Assume the 2D array as 1D array by computing indicies & perform Binary search

How? Figure out a way to convert 2D to 1D index

low	high	mid		
0	29	14	→ (2,2)	K = 23
0	13	6	→ (1,0)	
6	13			

- ★ row = mid / M
- ★ col = mid % M

ar_{N*M} :

	0	1	2
0	1	2	3
1	3	4	5
2	4	15	16
3	6	7	8
4	17	18	19

mat(i,j)

Complexity: $O(\log_2(N*M))$, $O(1)$

$\Rightarrow O(\log_2 N + \log_2 M)$

Optimized solutions are way5 & way6

Problem Given an array of 0's and 1's find the max length of contiguous 1's.

way1 : ar_{N*M} :

0 0 0 0 0 0
0 0 0 1 1 1
0 0 0 0 0 1
0 0 1 1 1 1
0 0 0 0 1 1

Brute force : $O(N*M)$, $O(1)$

way2 : Perform binary search on each row & find for the index of the first occuring '1'.

complexity : $O(N\log_2 M)$, $O(1)$

way3 : $(N*M) \rightarrow N\log_2 M$, 1

way4 : Apply the previously discussed optimized approach by iterating from TOP-RIGHT or BOTTOM-RIGHT counting the number of contiguous 1's.

ar_{N*M} :

0 0 0 0 0 ↓
0 0 0 ← ← 1 1 c = 2
0 0 0 ↓ 0 1 c = 2
← ← ↓
0 1 1 1 1 c = ~~2~~ 4
↓
0 0 0 1 1 c = 4

Complexity : $O(N+M)$, $O(1)$

Problem given an array, find the first missing integer.

way1 : Answer should lie in range $[1, N+1]$

ar_N :

5	1	3	2	8	1	-6	15	10	-4
0	1	2	3	4	5	6	7	8	9

Brute force : $O(N^2)$, $O(1)$

way2 : Sort the array. Assuming there would be duplicate integers, iterate through array to find out missing no.

Complexity : $O(N\log_2 N + N)$, $O(1)$

way3 : Using Hashset of size N+1, check for keys (Unsorted set)

Complexity : $O(N+N)$, $O(N)$

Insert into Hashset → Hash set
Iterate over Hashset keys

way4 : Using count array of size N+2, ignore elements that are $a_i < 0$ & $a_i > N+1$, and insert the remaining bool values into it.

```
bool cnt[N+2] = {F}
N-1
∀  if(ar(i) ∈ [1, N+1])
i=0
        cnt[ar(i)] = T;
```

The size of the count array can be further reduced to N

```
bool cnt[N] = {F}
N-1
∀  if(ar(i) ∈ [1, N])
i=0
        cnt[ar(i)] = T;
```

$\forall_{i=0}^{N-1}$ if (cnt(i) == F)
return i+1;

Ex

ar(i)	idx
1 →	0
2	1
3	2
:	:
N+1	N

Complexity : O(N+N), O(N)

- store / Mark elements in cnt[]
- Count array
- Find the missing integer

way 5 : Optimize further, without using the count array.

Mark the values

	0	1	2	3	4	5	6	7	8	9
ar_N :	5	1	3	2	8	1	6	18	10	4
	-5	-1	-3		-8		100	100		100
								-100		-100

- Remove the irrelevant elements by replacing by 100
- Mark the visited indicies by making it negative
- Iterate over the array & find the first non-negative integer which is the ans = (index+1)

Step 1 :
```
for (int i=0 to N)
    if (ar(i) < 1 || ar(i) > N)
        ar(i) = INT_MAX;
```

Step 2 :
```
for (int i=0 to N) {
    int x = a(i);
    if (x < N+1)
        a(abs(x-1)) = -a(abs(x-1))
}
```

Step 3 : for (int i=0 to N) { if (a(i) > 0) return i+1; }

Complexity : O(N), O(1)

Problem : Given an array containing N elements, find the frequency of each element in the array

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

N = 10 — $1 \le ar(i) \le N$

Frequency :

1 → 2
2 → 0
3 → 2
4 → 1

way 1 : $O(N^2)$, 1 Brute force : Search every [1, N] elements count

way 2 : Optimized approach.

	0	1	2	3	4	5	6	7	8	9
ar_N :	1	5	10	3	5	8	1	10	4	3
	+100		+100	+100	+100			+100		+100
	101%100 = 1 (index)		+100		105%100 = 5 (index)					
	+100				+100					

- Visit the ar(i)th index & add '100' to its array value
- Iterate again & find the count by ar(i)/100

$\forall_{i=0}^{N-1}$ ar[ar[i]%100 - 1] += 100

$\forall_{i=1}^{N}$ print i → ar[i-1]/100

Problem Given the height of the buildings, find the amount of rainfall that can be accumulated between the buildings

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

ans = 2+3+4+2+5+7+6+3+4+5+1+2 = 43

Way1 : Find Left Max & Right Max for every building & calculate w(i)

$w(i) = (MIN(LM, RM) - ar(i))$

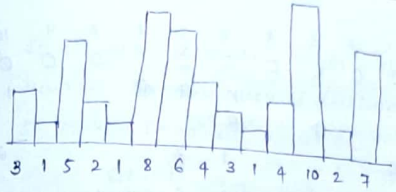
(or)

ans = ans + min(LM, RM) - ar(i);

Complexity : $O(N^2)$, $O(1)$

Ex Building 8 ⇒ LM = 8 and RM = 10

Way2 : Find the max buildings on left & right for each building



Left Max : 3 3 5 5 5 8 8 8 8 8 8 10 10 10

Right Max : _ _ _ _ 10 10 10 10 7 7

step1 : $\forall_{i=1}^{N-1}$ LM(i) = max(LM(i-1), ar(i))

LM(0) = ar(0)

step2 : $\forall_{i=N-2}^{0}$ RM(i) = max(RM(i+1), ar(i))

RM(N-1) = ar(N-1)

step3 : ans = ans + min(LM, RM) - ar(i);

Complexity : O(N), O(2N)

↳ 2 arrays for LM & RM

We can further optimize the space by computing LM on the fly.

Complexity : **O(N), O(N)**

Problem Given array of size N, initially filled with 0's. Compute the following & print the answer for each query

① i : ar(i) = 1 - ar(i)

② i : $\min_{j=0}^{N-1}$ (abs(j-i)) ar(j) = 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar_N :	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
				↑ 1		↑ 0	↑	↑			1				

Q : 1 10 1 3 2 3 → 0

2 7 → 3 2 6 → 1

1 5 1 5

2 7 → 2 2 6 → 3

* Initially if no queries of type '1' are given then ans = -1

for ex : 2 12 → -1

if there no 1 in the current state of array

way1 : O(Q*N), O(1)

Q1	Q2
O(1)	O(N) → Find the nearest '1'

Either iterate from the start of the array or use 2 ptr to expand on both sides to visit the first occurance of 1 element.

way2 : Maintain a list of elements that are given in Q1 & for each calculate the difference of Q2 with every element in the list.

L = { 10, 5, 3 . . }

Q1 : 1 5 → check for key 5

if it exists, remove from the list

otherwise, add it to the list

* HashSet can be used here (alt)

complexity : Q1 : N

Q2 : N

O(Q*N), O(N)

way3 : In the above approach, replace the list with a HashSet. Then insertion & deletion would be done in O(1) So, overall complexity would be

Q1 : 1

Q2 : N O(Q*N), O(N)

way4 : Maintain a sorted list such that

Q1 : 1 8 → perform insertion sort

Q2 : 2 16 → Find the ceil(16) & floor(16) & compute the difference.

L = { 3, 7, 8, 10, 18, 25, . . }

Q1 : 1 7 → Delete the element from list by shifting all the elements

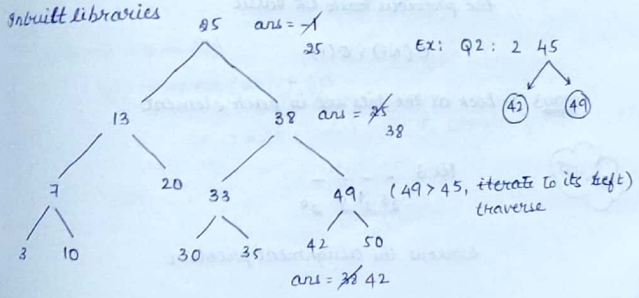
complexity : Q1 : N

Q2 : $2 \log_2 N$ O(Q*N), O(N)

way5 : In the above approach, replace list with a sorted HashSet

Sorted HashSet : Balanced BST { S/I/D - $\log_2 N$; size() - 1

Inbuilt libraries



* Sorted HashSet has inbuilt methods to get
floor(x) & ceil(x)

complexity : Q1 : $\log_2 N$ | Q2 : $2\log_2 N$ → $O(\log_2 N)$, $O(N)$

To search for ceil & floor using inbuilt methods supported in Java

Problem Given an array of elements, find the Bitwise OR of all the subarrays. sum of

arr : 3 7 25 18 1 14 12 7 2

ans = 3 + 3|7 + 3|7|25 + 3|7|25|18 + - -
7 + 7|25 + 7|25|18 + - - -
25 + 25|18 + - - -

way1 : Generate all the subarrays & compute bitwise OR
$O(N^2 * N)$, $O(1)$

way2 : Using forward hash technique, compute OR by storing the previous hash OR value
$O(N^2)$, $O(1)$

way3 : Look at the bits set in each element

REVISIT

bit 3 - - - -
2^3 2^2 2^1 2^0

Workout the assignment problem

arr : 3 7 25 18 1 14 12 7 2 6 43 37 19
x x ✓ x x ✓ ✓ x x x ✓ x x
↑ c=1 c=2 c=3 ↓ c=1

subarrays starting with 14 = (N-5)

$(c+1)(N-i) * 2^b$

c=0)

way4 : Find the subarrays ending with a particular element

3 7 25 18 1 14 12 7 2 6 43 37 19
x x ↓ x x ↓ ↓ x x
✓ ✓ ✗
(7,2) (2) ② count subarrays with no bit set ending with 2

7(c=1) 2(c=2)

c=0
for (int i=0 to N)
if (checkBit(arr(i), b))
$(i+1) * 2^b$
=
else

* Complexity : O(N), O(1)

Stacks & Queues

 # STACK

- LIFO / FILO (ar) → O(1), O(N)
- push(x) (ar[++t] = x)
- pop() (t--)
- top() / peek() (ar[t])
- size() (t+1)

 # QUEUE

- FIFO / LILO (ar) → O(1), O(N)
- enqueue(x) (ar[++r] = x)
- dequeue() (f++)
- front() / rear() ar(f+1) ar(r)
- size() (r-f)

Implemented using arrays or linked lists

ar_{100} : 30 push → t=69 | 30 pop → t=39 | 40 push → t=79

ar_{100} : 30 enqueue → r=69, f=-1 | 30 dequeue → r=69, f=29 | 40 enqueue → r=? → overflow (if, rc)

TODO

Try to use the starting empty array blocks to enqueue elements using modulus operator

ar_N ; f=-1 ; r=-1

enqueue() : Check for overflow ; ar((r++)%N) = x

dequeue() : check for underflow ;
if (r==0) f++
else (f++)%N

size() : // r-f

Date 29/10/19

CLASS - 13

Contd...

After updating the queue implementation using % operator check for the below ex: for N=10

→ Enq(3) → deq(3)

→ Enq(3) → deq(3) → Enq(10)

→ Enq(10)

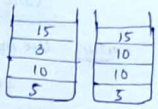
Problem Implement getMax() using stacks

5, 10, 3, 15, gM(), ↑, gM(), 7, size()

→ getMax()

→ getMin()

→ getAverage()



Implement using two stacks; S1 & S2

for getAvg() Maintain the sum of elements in a variable (Σ a(i)) & call size() to get avg (sum/size())

Problem Implement 2 stacks using one array of size N.

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

S1 (t1=-1) (t2=N) S2

	push(x)	pop()	top()	size()
S1	if (t1+1==t2) overflow	if (t1==-1) underflow	return t1--;	(t1+1)
S2	else ar(t1++)=x	else t1--;		(N-t2)

S1 ar[++t1] = x t1--; ar[t1] t1+1

S2 ar[--t2] = x t2++; ar[t2] N-t2

Problem: Implement M stacks in an array of size N

Assume M << N Ex: M=5 N=1000

way1: Divide → N/M partitions & implement

0 → 200 t1 = -1 t2 = 200

201

401

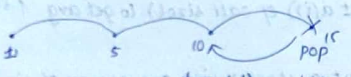
:

801 → 1000 t5 = 800

can't use space effectively

way2: Use indexes to push elements

- S1 = { 1, 5, 10, 15 ... }
- S2 = { 2, 6, 11, 16, ... }
- S3 = { 3, 7, 12, 17, ... }



top = (x) (x)

pop() = return { ar[top]; top = top - M; }

way3: Initialize all pointers to -1

prev(5) = t2

ar(5) = x

t2 = 5

0	1	2	3	4	5	
s1a	s1b	s2a	s4a	s1c	s2b	Elements pushed
-1	0	-1	-1	1	2	Index of previous elements

t1 = ~~4~~ ~~0~~ ~~1~~ 4 t4 = ~~1~~ 3

t2 = ~~1~~ 2 t5 = -1

t3 = -1

Maintain one more array to maintain the list of empty indicies

L = {0, 1, 2, ... N-1}

→ pop() : S2

t[i] = prev[t[i]]

t[i] → E

→ push()

idx ← E

prev[idx] = t

ar[idx] = x

t[t] = idx

→ size()

- maintain one more array [M] to store size of each stack
- Iterate over prev[i] to count the elements of that stack

```
int size(int t) {
    c = 0;
    while (t != -1) {
        t = prev[t]
        c++;
    }
}
```

Complexity: O(3N + 2M) space O(1) time

Way 4: If we are constrained to use only one array, then divide the given array for each of the previously discussed extra space

Max size limit of a stack = $\frac{N-2M}{3}$

TODO: with this approach, try to find out a way to use the extra space left behind in each partition

* Inbuilt Libraries

→ Stacks : push(x), pop(), top(), size()

→ Queues : enqueue(x), dequeue(), front(), rear() size()

Problem: Implement queue using 2 stacks

Way1: Maintain 2 stacks

→ s1.push() to s2 - enqueue(x)

→ s1.pop() & insert elements to s2 - dequeue()

Enqueue()

→ if (t2 != -1)
copy elements of s2 to s1

→ ar(t1++) = x;

Dequeue()

→ if (t1 != -1) if (t2 == -1)
copy elements of s1 to s2

→ return ar(t2--);

enq() : ~~O(n)~~ O(1) size() : s1.size() + s2.size() O(1)

deq() : ~~O(n)~~ O(1) front() : top2() O(1)

rear() : top1() O(1)

Why? Why does dequeue() takes O(1)?

Asymptotic Notation / Analysis : O(N)

-Amortised Analysis should be considered here : O(1)

where we ignore the time taken for copying elements s1 → s2

Ignore the rare operations though they are ~~rare~~ costly.

front() :

s1	s2
~~7~~	(7) → front top of s2
~~2~~	2
~~5~~	5

rear() : top of s1

s1	s2	
rear element ← (5)	~~2~~	Both the operations involve copying of elements
2	~~5~~	

This can be optimized by maintaining a rear variable.

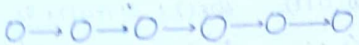
r = ~~1~~ ~~7~~ ~~2~~ 5 (last inserted element)

f = ? cant store in a variable

* Dynamic Lists

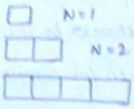
vector → v[i]	· push_back(x)	O(1)
ArrayList → ar.get(i)	ar.add(x)	
[] → l[i]	· append(x)	

★ Linked Lists



How do you implement list.get() & list.add() using LL?

To expand the size of an array, double its size whenever it overflows. $N \approx 10^6 \approx 2^{20}$



To insert 2^{20} elements

$2^0 + 2^1 + 2^2 + \dots + 2^{19} = 2^{20} - 1$

$+ 2^{20}$ (copying elements)

$10^6 \simeq 20$ (copying elements)

Apply amortized analysis here & ignore this constant.

Therefore, insertion can be done in O(1)

The same can't be applied for LL.

★ Hashing

LoadFactor = 0.5

Re-Hashing takes time but ignored

size(HT) = 100

insert at 51 → expand size to 200

Re-hash all the keys 0-50 & copy to new HT

100 times → Insert random E

→ Delete random E

The search becomes slow if size is 200 when the elements are 50.

HashTables uses the concept of doubling the size.

The size expands & contracts depending on the load factor

Load factor { I = 0.5 (Expand size when 50% slots filled)
D = 0.75 (contract the size when 75% slots are empty)

Problem Given an array, print the first smaller element on its right side.

way1 Brute force $O(N^2)$, O(1)

Iterate through every element & find its right-min

ar_N:	5	12	7	10	15	8	2	4	7	6	15
ans:	2	7	2	8	8	2	-1	-1	6	-1	15

way1

```
if (a(i) > a(i-1))
    print prev-MIN;
else
    while (a(i) < a(i+1))
        i++;
    print a(i); prev-Min = a(i);
```

way2 Take an array B_N.

br_N:	~~8~~	~~12~~	~~10~~	~~15~~	4	~~7~~	15
	2	~~7~~	~~8~~	~~8~~		6	
		2	2	2		⊗	

[Do not update if we get a min element already]

store index in stack instead of elements

st: (indicies) 4, 3, 5, 1, 2, 0

st: (elements) 15 8 / 16 8 2 / 12 7 2 / 8 2

```
stack<int> st;
for(i = 0 to N)  i=j;
    while(___)(a(i) < st.top())
        int x = st.pop(); st.top();
        if(x < a(i))
            b(i) = x;
        st.pop();  i++;
    }
    st.push(ar(j)) → push index
```

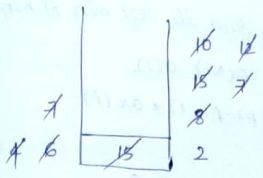
Solution

Processing from left to right

```
for(int i=0 to N)
    while(!stack.isEmpty() && a(i) < ar(stack.top()))
        int index = stack.top();
        br(index) = ar(i);
        stack.pop();
        i++;
    }
    stack.push(i);
}
```

Processing from right to left

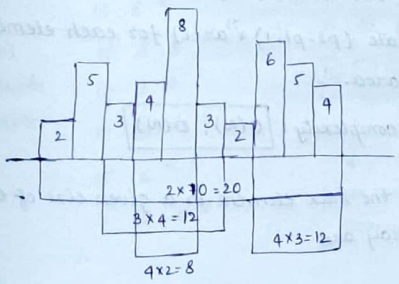
ar_N :	5	12	7	10	15	8	2	4	7	6	15
br_N :								4	7	6	-1



```
int br[N] = {-1};
stack<int> st;
for(int i = N-1 to 0) {
    while(!st.isEmpty && ar(i) < st.top()) {
        st.pop(); }
    if(stack.isEmpty())
        br(i) = -1;
    else
        br(i) = ar(i)
    st.push(ar(i));
}
```

Complexity: O(N), O(N)

Problem Calculate the maximum histogram area of the given graph.



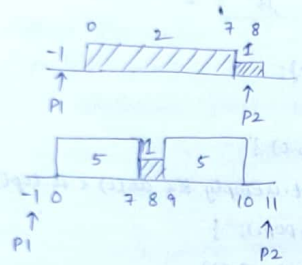
Max histogram area = 2×10 = 20

At each building find its left min & right min

way1 Brute force : $O(N^2)$, $O(1)$

ans = $(p2 - p1 - 1) * ar(i)$

way2



$(P_2 - P_1 - 1) * ar(i)$

$(8+1-1) * 2 = 16$ units

$(P_2 - P_1 - 1) * ar(i)$

$(11+1-1) * 1 = 11$ units

Use prefix & sufix min to find out p1 & p2

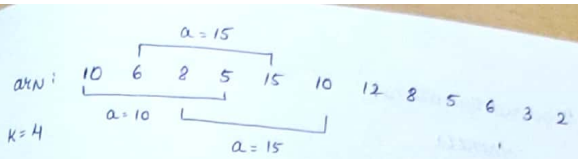
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar_N :	2	5	3	4	8	3	2	6	5	4	10	5	1	2	3
p2 :	12	2	6	5	7	6	12	8	9	12	11	12	N	N	N
p1 :	-1	0	0	2	3	0	-1	6	6	6	6	6	-1	12	13

→ Find out the min of each building on left & right of it using two stacks (Refer to prev problem)

→ calculate $(p2 - p1 - 1) * ar(i)$ for each element & find max area.

complexity : $O(N)$, $O(N)$

Problem Find the max element in a given size of window 'K' of array ar_N



way1 $(N+1-K)$ windows

Iterate over each window & getMax()

$O((N-K+1) * K)$, $O(1)$

way2 Using carry forward technique? word work

Use sorted Map (BBST)

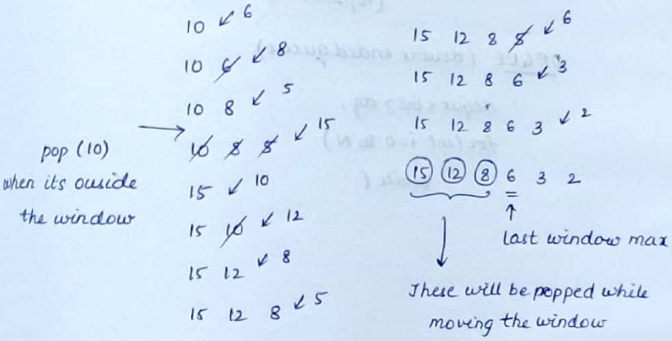
M = {5, 6, 8, 10}

frequency : 1 1 1 1↑ last element is max

complexity : $O(N \log_2 K)$, $O(K)$

way3 ar_N : 10 6 8 5 15 10 12 8 5 6 3 2

K = 4



pop (10) when its ouside the window

These will be popped while moving the window

Required operations:

front()
pop_back()
back()
push_back()
pop_front()

Doubly Ended Queue

push() : O(1) pop() : O(1) space : O(K)

complexity : O(N), O(K)

print (i > K-1) indicies

10 6 ↙ (push back())

10 6 ↙ 8 (push back())

↑ (pop_back()) (8 > back())

10 8 7 ↖ 15 (window next E)

↗ (pop_front()) ↑ (Pop_back()) (15 > back())

DEQUE (double ended queue)

deque <int> dq;

for (int i=0 to N)

while (

Date
21/06/19

CLASS - 14

Linked lists
- Singly linked lists
- Doubly linked lists

Circular linked lists

```
void createNode(int x) {
        Node n = new Node();
        n.data = x;
        n.next = null; // n.prev = null;
        return n;
}
public class Node {
        int data;
        Node next; // Node prev;
}
```

Problem: Create a list of 'N' numbers as a single linked list

N = 4; 1 → 2 → 3 → 4 → NULL

```
Node createList(int N) { // N >= 1
        Node head = createNode(1);
        Node temp = head;
        for(int i = 2 to N) {
                Node newNode = new Node(i);
                temp.next = newNode;
```

```
        temp = temp.next;
    }
    return head;
}
```

Problem

(i)
```
void print (Node head) {
    Node tail = head;
    while (tail.next != null) {
        print (tail.data);
        tail = tail.next;
    }
}
```

(ii)
```
int size (Node head) {
    Node tail = head;
    int length = 0;
    while (tail != null) {
        length++;
        tail = tail.next;
    }
    return length;
}
```

(iii)
```
void printR (Node head) {
    Node tail = head;
    int size = size (head);
    int[] values = new int[size];
    while (tail != null) {
        values[size-1-i] = tail.data;
        tail = tail.next;
        i++;
    }
    for (int i=0 to size)
        Print (values[i];
} ~~ret~~
```

(iv)
```
Node insert (Node head, int x, int pos) {
    Node tail = head; int count = 0;
    if (pos < 0 || pos > size(head))   if (pos == 0) {
        return ~~null~~; head;              Node n = new Node(x);
                                            n.next = head
while(--pos != 0) while (tail.next != null) {   head = n;
  tail = tail.next      count++;          }
                        tail = tail.next;  if (count == pos) break;
    }
    Node newNode = new Node (x);
    if (tail != null)
        ~~tail~~ Node nextNode = tail.next;
        tail.next = newNode;
        newNode.next = nextNode;
    ~~} else {~~
        ~~tail~~ = newNode;   // check not required
        head
    }
    return head;
}
```

(v) Node insert(Node head, int x) {

Node tail = head;

// ③ while (tail.next != null) {

if (x < temp.data) {

tail.next.data) {

Node nextNode = tail.next;

Node newNode = new Node(x);

tail.next = newNode;

newNode.next = nextNode; break;

}

tail = tail.next;

}

// → ①

(if x ≤ tail.data) if (tail == null) {

head = new Node(x);

}

return head;

}

// ② if (temp.next == null)

tail.next = newNode

(iii) void printR(Node head) {

if (head == null)

return;

printR(head → next);

S.O.Pln(head → data);

}

complexity: O(N), O(1)

(v) while (head.next != null && head.next.data < x)

head = head.next;

(vi) Node deleteAll(Node head, int x) { // USLL

if (head == null)

return head;

Node tail = head;

while (tail.next != null) {

if (tail.data == x) {

Node temp = tail;

tail = tail.next;

temp = null;

}

if (head == null) {

head = tail; tail = tail.next;

}

}

return head;

}

(vi) Node delete(Node head, int x) {// SSLL

// ① if (head == null)

return head;

// ③ while (tail.next != null && tail.next.data != x)

head = head.next;

tail tail

}

Node temp = tail.next.next;

tail.next = temp;

// ② if (head.data == x)

head = head.next;

```
        return head;
    }
```

(vii)
```
Node deleteAll ( Node head, int x ) {   // USLL
    if (head == null)
        return head;
    while (head.data == x)
        head = head.next;
    Node tail = head;  Node prevNode = tail;
    while (tail.next != null) {
        Node temp = tail;
        while ( temp.data == x)
            temp = temp.next;
        prevNode.next = temp;
        prevNode = tail = temp;
        tail = tail.next;
    }
    if (prevNode != tail) {
        prevNode.next = null;
    }
    return head;
}
```

(viii)
```
void distinct ( Node head) {
    if (head == null)
        return;
    while (head.next != NULL) {
        if (head.data == head.next.data)
            head.next = head.next.next;
        } head = head.next;
    }
    if (head
}
```

(ix) Node distinct (Node head) { } TODO InterviewBit
unique

(x) Node reverse (Node head) {

i/p : 7 → 2 → 8 → 5 → 1
NULL (prevNode), h, t

o/p : 1 → 5 → 8 → 7 → 2

Solution
```
Node prevNode = null;
Node temp;
while ( head != null) {
    temp = head.next;
    h.next = prevNode;
    prevNode = head;
    head = temp
}
return prevNode;
```

```
        return head;
    }
```

(vii)
```
Node deleteAll ( Node head, int x ) {   // USLL
    if (head == null)
        return head;
    while (head.data == x)
        head = head.next;
    Node tail = head; Node prevNode = tail;
    while (tail.next != null) {
        Node temp = tail;
        while ( temp.data == x)
            temp = temp.next;
        prevNode.next = temp;
        prevNode = tail = temp;
        tail = tail.next;
    }
    if (prevNode != tail) {
        prevNode.next = null;
    }
    return head;
}
```

(viii)
```
void distinct ( Node head) {
    if (head == null)
        return;
    while (head.next != NULL) {
        if (head.data == head.next.data)
            head.next = head.next.next;
        } head = head.next;
    }
    if (head
}
```

(ix) Node distinct (Node head) { } TODO InterviewBit
unique

(x) Node reverse (Node head) {

i/p : 7 → 2 → 8 → 5 → 1
NULL (prevNode), h, t

o/p : 1 → 5 → 8 → 7 → 2

Solution
```
Node prevNode = null;
Node temp;
while ( head != null) {
    temp = head.next;
    h.next = prevNode;
    prevNode = head;
    head = temp
}
return prevNode;
```

(vii) Solution

```
if (h == NULL)
    return h;
Node th = h;
while (h.next != NULL) {
    if (h.next.data == h.data)
        temp = h.next;
        h.next = h.next.next;
    else
        h = h.next;
}
if (th.data == x) {
    th = th.next;
}
return th;
}
```

(viii) Solution

```
if (h == NULL)
    return h;
while (h.next != NULL) {
    if (head.next.data == head.data)
        h.next = h.next.next;
    else
        h = h.next;
}
```

(x) Recursive Implementation of reversing a linked list

Solution

```
Node reverse (Node head) {
    if ((head.next == null) || head == null)
        return head;
    Node newhead = reverse (head.next);
    head.next.next = head;
    head.next = null;
    return newhead;
}
```

Problem Given 2 singly LL sorted, merge them into a singly LL

H1 : 3 → 10 → 12 → 13 → 20 ↵

H2 : -1 → 2 → 7 → 9 → 18 → 23 → 28 → 30 ↵

way1 using two ptrs in each one of the lists & compare data

```
Node h = new Node(); Node th = h;
while (h1.next != null && h2.next != null) {
    if (h1.data < h2.data) {
        h.next = h1;
        h = h1;
        h1 = h1.next;
    } else {
        h.next = h2;
        h = h2;
        h2 = h2.next;
    }
}
```

```
if (h1 != null)
    h.next = h1;
if (h2 != null)
    h.next = h2;
return th.next;
```

complexity : O(N), O(1)

Problem given a SLL return the mid of the list.

h1 : -5 → 8 → 3 → 2 → 6

h1 : -5 → 8 → 3 → 6

T F

```
Node findMid (Node head, bool flag) {
    Node t1 = head;
    Node t2 = head;
    while (t2 != null) {   // odd length lists
        t1 = t1.next
        t2 = t2.next.next;
    }
    while (t2.next != null) {   // Even length lists
        t1 = t1.next
        t2 = t2.next.next;
    }
    if (flag)  if (h == null) return null;
```

solution

```
    while (f.next != null && f.next.next != null)
        f = f.next.next;
        s = s.next;
    }
    if (f.next == null || flag == true) {
        return s; }
    return s.next;
}
```

Problem Sort a SLL. Evaluate the details of sorting using the below sorting techniques. Implement merge sort.

BS / SS / IS / QS / MS / CS.

solution

```
Node MS (Node h) {
    if (h == null || h.next == null)          → 1
        return h;
    Node m = findMid (h);                     → N
    Node sh = mid.next;                       → 1
    mid.next = null;                          → 1
    return Merge (MS(h), MS(sh));             → T(N/2) + T(N/2)
}
```

$T(N) = 2T(N/2) + N$ No extra space required

Problem given a singly linked list, modify in such a way that

i/p : $L_1 \rightarrow L_N \rightarrow L_2 \rightarrow L_{N-1} \rightarrow L_3 \rightarrow L_{N-2} \rightarrow \cdots$

solution. Split the list & find mid

Reverse the second half of list

Iteratively ~~call join~~ add nodes one after other

```
void solve (Node h) {
    Node m = findMid (h, 7);
    Node sh = m.next;
    m.next = null;
    join (h, reverse (sh));
}
void join (Node h1, Node h2) {
    while (h2 != null) {
        Node temp = h1.next;
        h1.next = h2;
        h2 = h2.next;
        h1 = h1.next;
        h1.next = temp;
    }
}
```

Problem Given a singly linked list, modify it in such a way that odd numbers are aggregated in the first followed by the even numbers in order in which they appear in the given input.

i/p : 5 → 10 → 3 → 8 → 12 → 6 → 7 → 9 → 2 ⊥

o/p : 5 → 3 → 7 → 9 → 10 → 8 → 12 → 6 → 2 ⊥

way1 Maintain 2 head ptrs : lists for odd & even elements & then join in the end.

```
void solve (Node h) {
    Node h1 = new Node (); th = h1;
    Node h2 = new Node (); temp = h2;
    // h1 → odd elements
    // h2 → even elements
    while (head != null) {
        if (head.data%2 == 0) {
            h2.next = head; h2 = head;
            head = head.next;
            h2.next = null;
        } else {
            h1.next = head; h1 = head;
            head = head.next;
            h1.next = null;
        }
    }
    h1.next = temp.next;
    // th.next → head ptr of complete list
}
```

Problem Find if the given SLL is a palindrome or not.

i/p : a → b → c → b → a

way1 find the mid node & reverse the second half of the list. Now compare node data one by one.

NOTE In linked lists, the following pts help:

- Defining a dummy node
- Reverse a list whenever we need to traverse in reverse order.

TODO Implement all the (10) functions to insert, delete, reverse, .. etc for a doubly linked list.

- reverse
- deleteAll
- unique / distinct
- $l_1 \to l_N \to l_2 \to l_{N-1} \to \cdots$
- ODD - EVEN aggregation

If we have to print $\frac{3N}{4}$ th element, maintain 2 pointers;

slow = s.n.n.n

fast = f.n.n.n.n

Problem Given an address of a node, delete the node from list

i/p : 5 → 8 → 3 → 16 → (18) → 1

delete Node : 18

Constraints : The given node is neither head nor tail.

Solution Replace the data of given node with next node data & delete the next node.

```
void delete(Node t) {
    t.data = t.next.data;
    t.next = t.next.next;
}
```

Problem Given 2 SLL's add them & find the result in form of SLL

i/p 1 : 3 → 5 → 8

i/p 2 : 9 → 8 → 2 → 1

Ans : 1 → 0 → 1 → 7 → 9 (358 + 9821)

way1 : Reverse both the list and start adding each node data by maintaining carry_bit

O(N), O(1)

way2 : Convert each list to a number & add them to a variable & again split the digits to get resultant linked list

Limitation : If the no of nodes are very large that even cant fit in datatypes.

TIP Use this approach for problem STRING MODULO in HR/SI

[((0*10+3)*10+5)*10+8]

solution

```
void solve(Node h1, Node h2) {
    h1 = reverse(h1);  Node h3;  h3 = th;
    h2 = reverse(h2);  int carry = 0;
    while (h1 != null & h2 != null) {
        int sum = h1.data + h2.data + carry;
        if (sum > 9) {
            carry = 1;
        }                          h1 = h1.n; h2 = h2.n;
        sum = sum % 10;            h3 = new Node();
        h3.data = sum;             h3 = h3.next;
    }                              h3 = next = h;
```

```
    return (reverse(th));
}

// c = sum/10   data = sum%10
```

06/07/19

1 P → SLL Given two linked lists

h1: 7 → 1 → 5 → 3 → 8 → 2 → -1 → 2 → 12 → NULL

h2: 8 → 6 → 2

Check whether there is an intersection?
If yes, find point of intersection.

BF: Check for each & every node in LL1, whether it is there in LL2

N×M, 1

2 → Using HashSet<Node>

h1 → HashSet<Node>

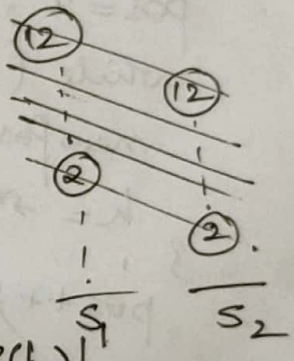
N+M, N

3 → Reverse Order: Cannot reverse these two LL's
We can use two Stacks

N+M+(1), N+M

S_1 S_2 Pop Operations

LIFO



4 → Find Size(h_1) → N
Size(h_2) → M

Skip, $d = |S_2(h_1) - S_2(h_2)|$ nodes
(N) (M)

And compare each node thereafter

N+M, 1

7 → 1 → 5 → 3 → 8 → 2 → -1 → 2 →
Skip 8 → 6 → 1

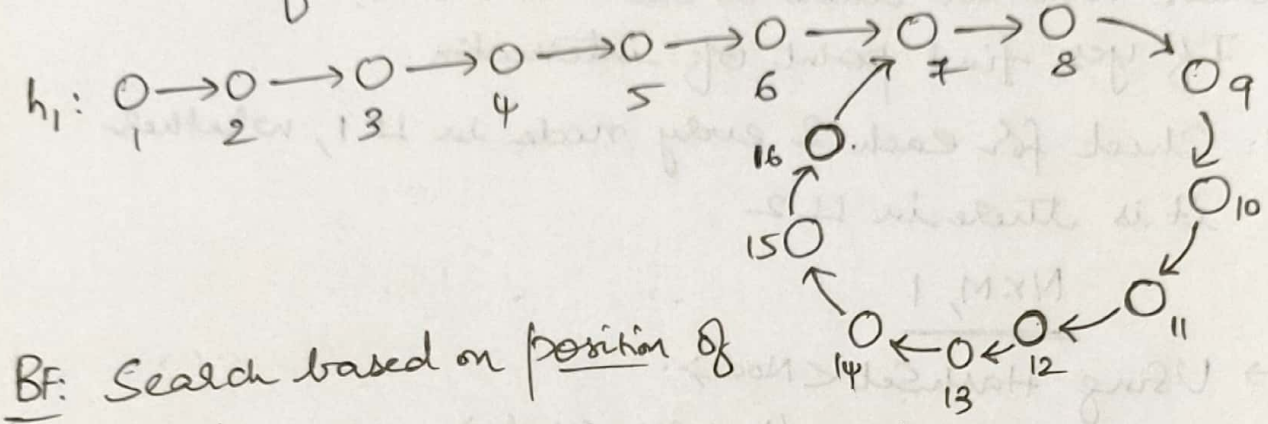
d = |9 - 7|
= 2 Nodes

After skipping d nodes, go for a two pointer solution

2 P → Given two linked lists,

a) Check if a cycle exists in the LL

b) If there is a cycle, break the cycle



BF: Search based on position of node,

Node 17 position is not there, instead it is at position 7

```
pos = x;
while (h != NULL) {
  newpos = find (th, h);
  h = next;
}
pos++;
```

$N^2, 1$

2 → Using Hashset, Search and insert, Repeat

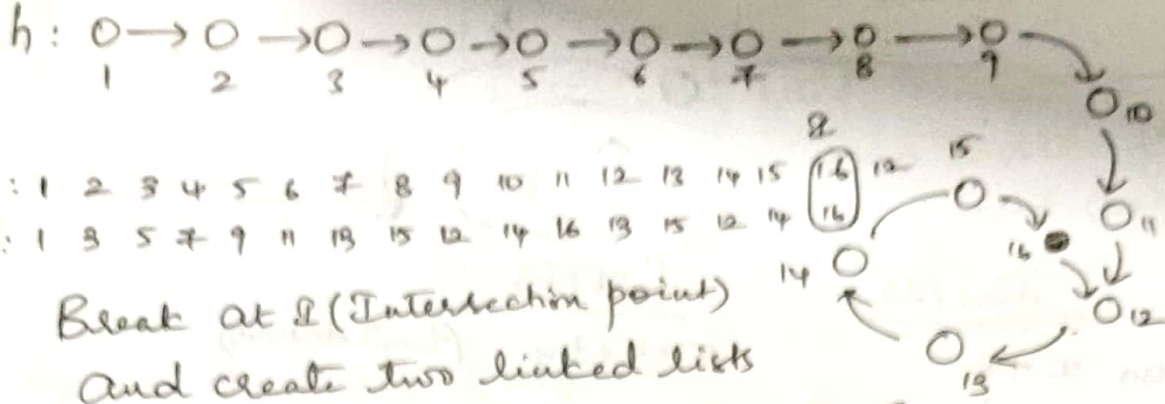
N, N

3 → Using Slow and fast pointers,

S: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13

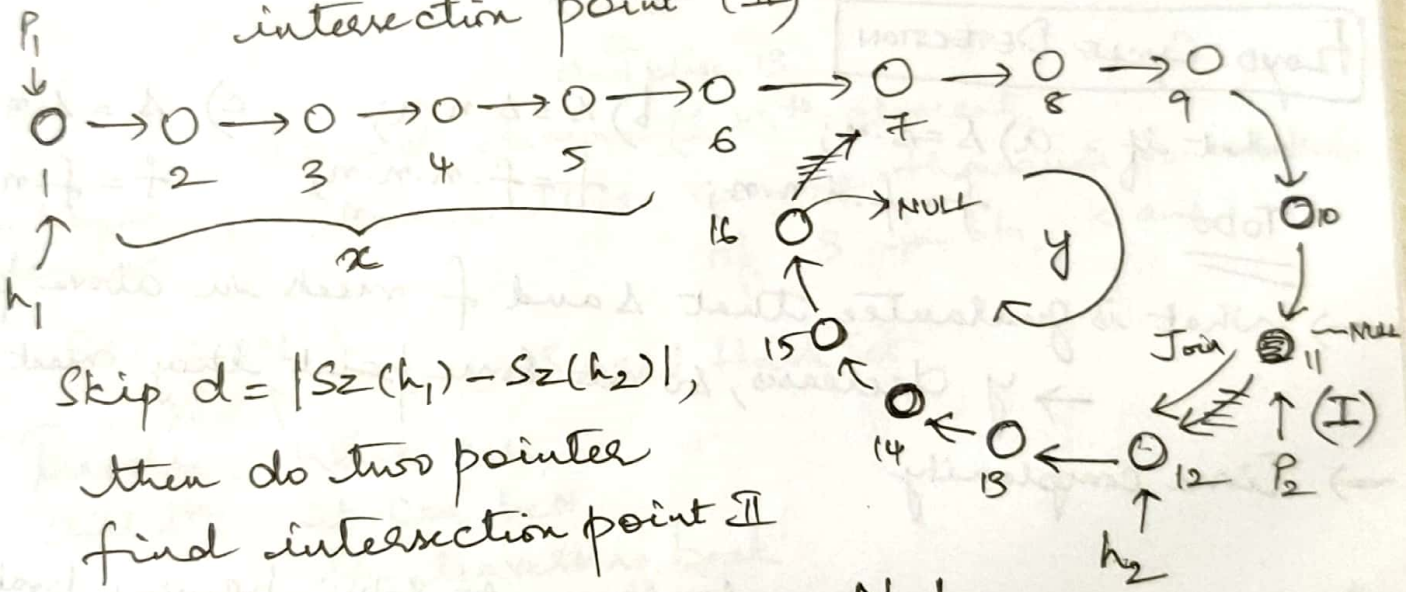
F: 1 → 3 → 5 → 7 → 9 → 11 → 13 → 15 → 7 → 9 → 11 → 13

Meets at



Break at I (Intersection point)
and create two linked lists
Before breaking, store location of
Break point

Use 1.4 d technique, and find the intersection point (II)
Join the linked list at previous intersection
point (I) and break the linked list at
intersection point (II)



Skip $d = |Sz(h_1) - Sz(h_2)|$,
then do two pointer
find intersection point II
at 7

N, 1

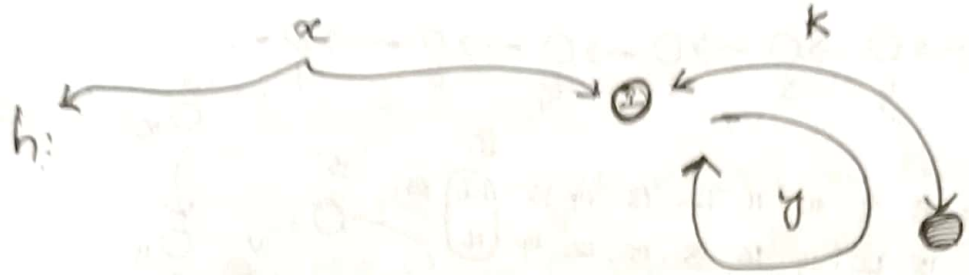
3.2 → With out Breaking the link,
Use two pointer approach Using P_1 and P_2

$\therefore s = s \cdot n$
$f = f \cdot n \cdot n$

$y = 10, 9, 8, 7, \ldots$

$\therefore N\ 1$
$x + y$

a) $s/f \rightarrow N, 1$
b) → N, 1 break and Join
→ N, 1
2-pointer



slow (s) — fast (f)

Slow travels $x+k$ dist for sure

$x+k+ty$ (t = rotations)

we know, $f = 2s$

$\therefore x+k+ty = 2(x+k)$

$x+k+ty = 2x+2k$

$ty = x+k$

$x = ty-k$

SLOW FAST POINTER TECHNIQUE IS CALLED

FLOYD CYCLE DETECTION

What if : a) $s = s.n;$ $f = f.n.n.n;$

b) $s = s.n.n;$ $f = f.n.n.n;$

c) $s = s.n.n$ $f = f.n.n.$ $.n$

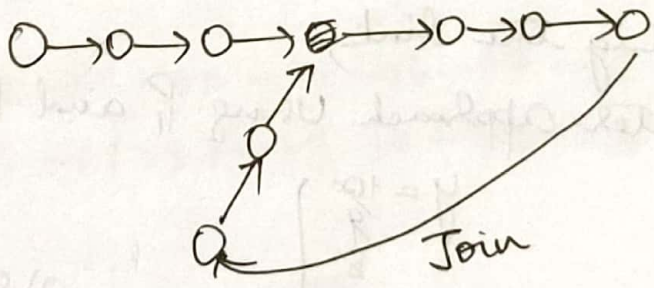
TODO

⇒ What is guarantee that s and f meets in above prob

→ y decreases, so at some point they meet

⇒ Time complexity

* ⇒ Can we use the above findings, to solve previous prob



and use s/f pointer approach

3 P → Implement LRU Cache → like int primary data

Least Recently Used Cache

Cache: FIFO, LIFO, LFU
↓
Frequently

→ To store frequently used data
→ To improve response time

Given, an array of size N
And cache size K

Ar_N : 10 3 15 3 12 18 15 3 13 18 ≠ Cache Miss
k i Cache Hit

K=4

~~10~~ ~~18~~
3
~~15~~ ≠
~~12~~ 13

10 3 15 3 12 18 15 3 13

Remove 12 and place 13

Cannot remove (i-k)th element,
Because there can be thousands & hundred of 3 in b/n k and i

Take Unique elements,

BF: N×N, K ← Unordered HashSet

Traverse till ith

Worst case, it can be N for traversing back

Cases:

Not Present → Not Full (NP → NF) : k
Present (P) : k
Not Present → Full (NP → F) : K+N ⇒ N (N for each elem

∴ N×N, K

2 →

arr : 10 3 15 3 12 18 15 3 13 18 7

0 1 2 3 4 5 6 7 8 9 10

Unsorted

~~10 → 0~~ ~~18 → 5~~ 9 HashMap <arr[i], i>

3 → ~~1~~ ~~3~~ 7

~~15 → 2~~ ~~6~~ 7 → 10

~~12 → 4~~ 13 → 8

Here, Np → NF : 1

P : 1

Np → F : k

k for each element

For N, N × K, K

3 → Use Sorted HashMap

0, 10

1, 3

2, 15

Unsorted Map <arr[i], i>

~~10, 0~~

3, ~~1~~ 3

15, 2

12, 4

18, 5

Sorted Map <i, arr[i]>

~~0, 10~~

~~1, 3~~

2, 15

3, 13

4, 12

5, 18

Here, Np → NF : $\log_2 k$

P : $\log_2 k$

Np → F : $\log_2 k$

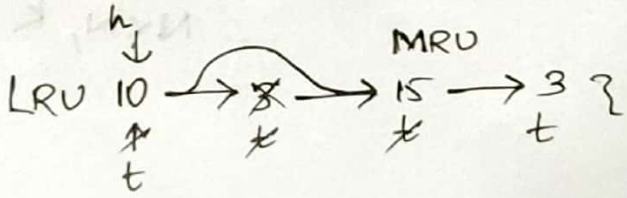
$\log_2 k$ for each element

For N, $N \log_2 k$, $2k$ — 2 HashMaps.

4 → Use linked list, where

head stores LRU

tail stores MRU



Here, Np → NF : k

P : k

Np → F : k

N × K, K

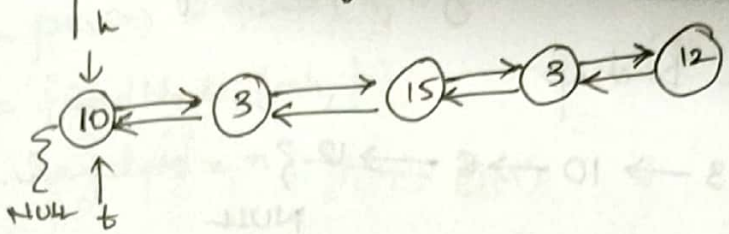
↓

Linked list

5 → HashSet {10, 3, 15} and Linked list

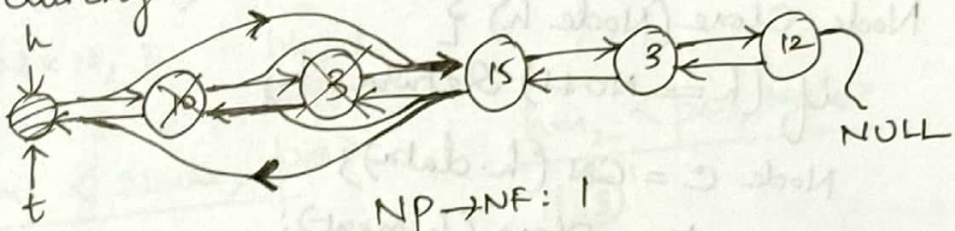
NP → NF : 1
P : K
NP → F : 1
NxK, K

6 → HashMap and Doubly Linked List



Use dummy node

HashMap <int, Node>




```
void LRU(int arr[], int N, int K){
  Node d = CN(-1);
  Node h = d, t = d;
  HashMap<int, Node> hm;
  for(int i=0; i<N; i++){
      Consider all 3 cases
  }
```

d : dummy Node

NP → NF : 1
P : K
NP → F : 1
1, 2K → N, K

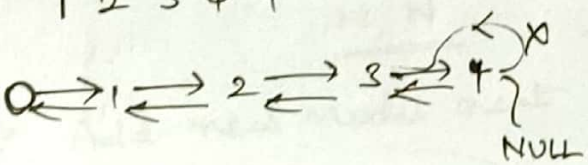
HashMap DLL

HashMap<int, Node> hm
= new HashMap<int, Node>();
hm.put

Edge cases:

k = 1

1 2 3 4 4



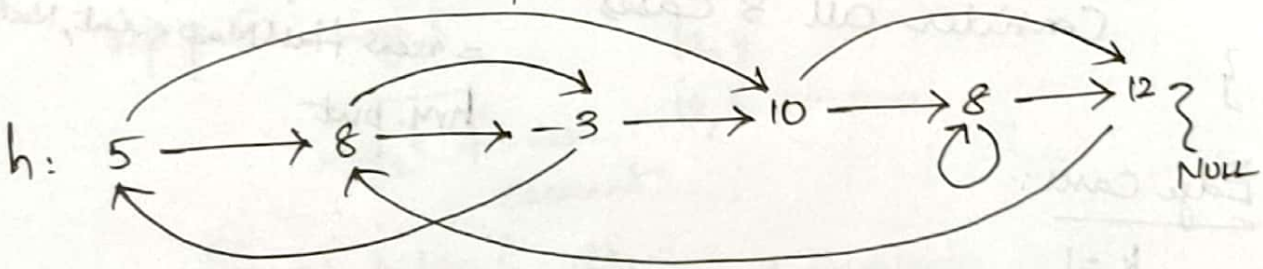
4 / P → CLONE A SLL

* Passing an array to a function, we use reference and in function if array is modified it is reflected in original array ⇒ clone and pass
* Take a backup

h: 5 → 8 → -3 → 10 → 8 → 12 } NULL

```
Node Clone(Node h) {
    if (h == NULL) return h;
    Node c = CN(h.data);
    c.next = Clone(h.next);
    return c;
}
```

5 / P → Clone a linked with two pointers
a. Normal next pointer
b. Random pointer which can point to any node




```
Class Node {
    int data;
    Node next, random;
}
```

BF: First created SLL and then assign random pointers of each node

nh: 5 (1) → 8 (2) → -3 (3) → 10 (4) → 8 (5) → 12 (6) → NULL

n = h.random; — O(1)
p = pos(n) = find(h, n); — O(N)
n = findNode(nh, p); — O(N)
} Optimize these
nh.random = n; — O(1)

∴ N × (N + N), 1
⇒ N × N, 1 ⇒ N^2, 1

2 → hm1 <Node, pos>

Node	pos
5	1
8	2
-3	3
10	4

hm2 <pos, Node>

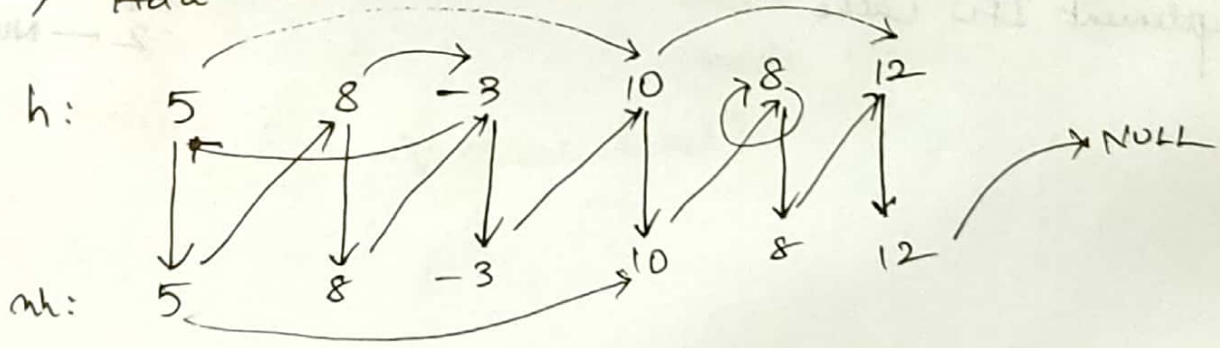
pos	Node
1	5
2	8
3	-3
4	10

nh.random = hm2.get(hm1.get(h.rand))

N · 2N
hm1, hm2

3 → hm <old node, New Node>
nh.rand = hm.get(h.rand)
N, N

4 → Add new nodes next to old nodes,



```
nh.next = h.next.next;
h = h.n.n;
nh = nh.n.n;
```

```
h.n = h.n.n;
n.h.n = n.h.n.n;
h = h.n
n.h = n.h.n;
```

N + N + N ⇒ N, 1

ToDos:

h: 1 → 3 → 8 → 2 → 5 → 10 → 6 → 8 → 12 → 4 → 7 → 10 → 2 → 5 → NULL

K = 3 Chunks

① Reverse the Chunks

8 → 3 → 1 → 10 → 5 → 2 → 12 → 8 → 6 → 10 → 7 → 4 → 5 → 2 → NULL

② Reverse alternate chunks

8 → 3 → 1 → 2 → 5 → 10 → 12 → 8 → 6 → 4 → 7 → 10 → 5 → 2 → NULL

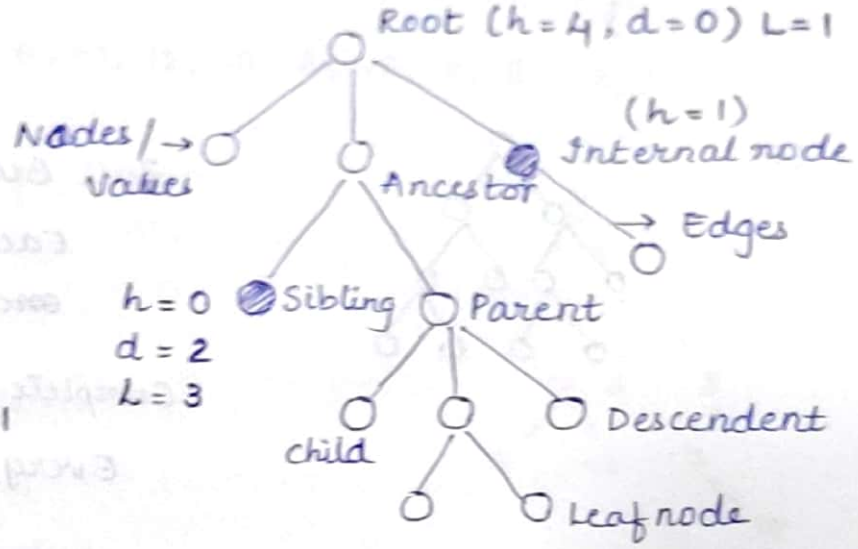
* Implement LFU Cache

Date: 07-02-2019

CLASS - 16

☆ TREES

N- Ary Trees



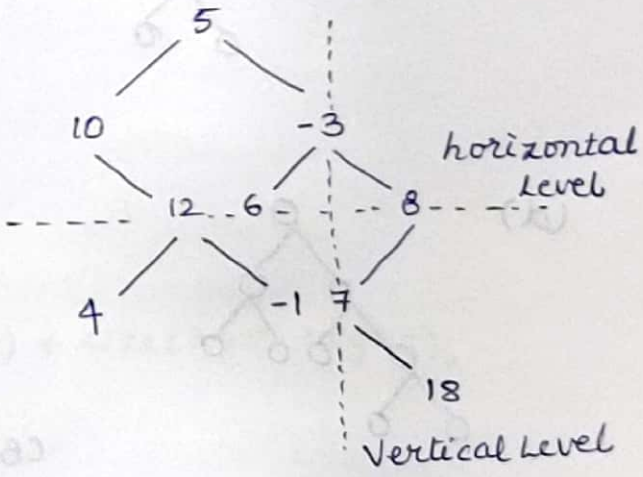
- Length of the path between the shaded nodes = 3
- Height (Tree) = 4
- Height (Shaded nodes) = 0, 1
- Depth (RootNode) = 0
- Depth (Shaded Node) = 2

Height (Tree) = Height (Root)
= Depth of the deepest leaf node

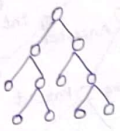
- Level (Shaded Node) = 3
- Level (RootNode) = 1

☆ Binary Trees

```
Class Node {
    int data;
    Node left, right;
}

Node CN (int x) {
    Node node = new Node();
    node.data = x;
```

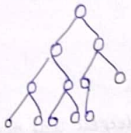


```
node.left = null;
node.right = null;
return node;
}
```

(a) 

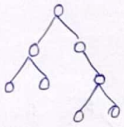
Full Binary tree = YES
Each node should have atleast ~~one~~ zero or two child nodes

Complete BT = NO
Every level should have all the nodes till that point / node

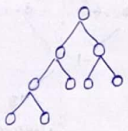
(b) 

FBT = NO
CBT = YES

FBT { Height (min) = ~~2^h+1~~ $2h+1$
Height (max) = $2^{h+1}-1$

(c) 

FBT = NO
CBT = NO
No of nodes (min) = $2^n - 1 = 1$
No of nodes (max) = $2^n - 1 = 8$ n = level

(d) 

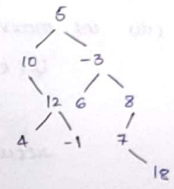
FBT = YES
CBT = YES

CBT { Height (min) = 2^h
Height (max) = $2^{h+1} - 1$

✱ Tree Traversals

- Preorder (DLR) : 5, 10, 12, 4, -1, ~~10, 5~~ -3, 6, 8, 7, 18
- Inorder (LDR) : 10, 4, 12, -1, 5, 6, -3, 7, 18, 8
- Postorder (LRD) : 4, -1, 12, 10, 6, 18, 7, 8, -3, 5

```
void inorder (Node root) {
    if (root == null)
        return;
    // inorder (root.left);
    Print (root.data);
    inorder (root.right);
}
```



(i)
```
int size (Node root) { return count; }
int count = 0;
void inorder (Node root) {
    if (root == null)
        return;
    inorder (root.left);
    count++;
    inorder (root.right);
}
```

(ii)
```
int size (Node root) {
    return 1 + size (root.left) + size (root.right);
}
```

(ii) int sum(Node R) {

```
    if(R == null)
        return 0;
    return R.data + sum(R.left) + sum(R.right);
}
```

(a (iii) int maxValue(Node R) {

```
    if(R == null)
        return ~~0~~ INT_MIN;
    return Math.max(R.data, Math.max(maxValue(R.left), maxValue(R.right)));
}
```

(b (iv) int height(Node R) {

```
    if(R == null)
        return ~~0~~ -1;
    return Math.max((R.left).height, (R.right).height) + 1;
}
```

(// Bottom-up approach

(v) class Node {

```
    int data;
    int depth;
    Node left, right;
}
```

```
void fillDepth(Node R) {
}
```

fillDepth(R, 0);

```
void fillDepth(Node R, int depth) {
    if(R == null)
        return;
    R.depth = depth;
    fillDepth(R.left, depth+1);
    fillDepth(R.right, depth+1);
} // Top-down approach
```

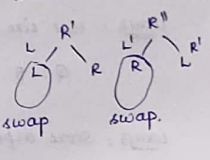
Problem Convert a given BST to its mirror image.

```
void mirror(Node R) {
    if(R == null)
        return;
    Node temp = R.left;
    R.left = R.right;
    R.right = temp
    mirror(R.left);
    mirror(R.right);
}
```

L R D ✓
L D R ✗
D L R ✓
L D L ~~✗~~ ✓

R L D ✓
R D L ✗
D R L ✓
R D R ~~✗~~ ✓

L D L



Problem Given a binary tree, print the node values in level order.

Breadth first search

```
void printLevelOrder (Node R) {
    if (R == null)
        return;
    q.enqueue(R);
while (q.isEmpty()) {
    Queue<Node> queue = new Queue<Node>();
    queue.enqueue(R.left);
    queue.enqueue(R.right);
    int size = queue.size();
    while (size > 0) {
        R' = print (queue.dequeue());
        size--;
    if (R'.left != null) q.enqueue (R'.left);
    if (R'.right != null) q.enqueue (R'.right);
    }
    print (newLine);
}
```

(a

(b

(c

way1 : Add a delimiter after adding elements of each level

Q: 5 ⊗ 10 -3 ⊗ 12 7 8 ⊗ 18 ⊗

way2 : Use size of the current queue to check if it's a new level

Q: 5 ↑ 10 -3 ↑ 12 7 8 ↑

way3 : Store depth of each node in the queue & maintain previous depth (pd = 0)

Q: 5,0 ↑ 10,1 ; -3,1 ↑ 12,2 ; 7,2; 8,2 ↑

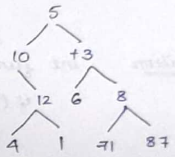
* Solution : Technique + Traversal

- Bottom-up
- Top-down

- In
- Pre
- Post
- Level

Problem Find the sum of all elements from root node to it each left leaf node for a given BT.

ans : 5,10,12,4 + 5,10,12,1 + 5,+3,6 + 5,3,8,71 + 5,3,8,87



Problem Form a number by appending digits of each node of a path

ans : 510124 + 510121 + 536 + 53871 + 53887

- Numbers are computed top-down
- Sum is computed bottom up

```
class Node {
    int data;
    int sum;
    Node left, right;
}

int f(Node root, int v) {
    if (root == null)
        return;
    Queue<int> q;
    q.enqueue (root);
```

```
(8) while (q.isEmpty()) {
        int size = q.size();
        while (size > 0) {
            v = v * pow(10, digits(r.data) + r.data);
            root.sum
            r = q.dequeue();
            r.sum = v;
```

solution

```
int fun(Node r, int v) {
    if (r == null)
        return 0;
    v = v * pow(10, digits(r.data) + r.data);
    if (isLeaf(r))
        return v;
    return fun(r.left, v) + func(r.right, v);
}
```

★ Binary Search Trees

- all(L) < D < all(R) ⇒ BST
- Insert a node in BST

```
if (r.data > x)
    r.left = insert(r.left, x);
if (r.data < x)
    r.right = insert(r.right, x);
```

```
Node insert(Node r, int x) {
    if (r == null) {
        t = createNode(x);
        return t;
    }
    if (r.data < x)
        r.right = insert(r.right, x);
    else{// if (r.data > x)
        r.left = insert(r.left, x);
    }
    return r;
}
```

Complexity : O(H) , O(1)

- N → Skewed → Left / Right
- log N → BBST → AVL Trees / Red Black Trees

Problem Given a BST search for a given node value & return true if it is present/found

```
bool search(Node r, int k) {
    if (r == null)
        return false;
    if (r.data == k)
        return true;
    if (r.data < k)
        return search(r.right, k);
    else
        return search(r.left, k);
}
```

* Insert array in BST

- Read (T)
- loop (T) for elements

```
Node createBST() {
        Read TV
        Node root = null;
        loop N
            Read x;
            root = insert(root, x);
        return root;
    }
```

* Delete a node in BST

```
Node deleteNode(Node r, int K) {
    // Case 1: For all leaf nodes
        if(r.data == K) r = null;
    // Case 2: For single child nodes  r = r.left;
    // Case 3: For nodes with 2 child nodes
        Replace it with MIN (Right side)
                    (or)  MAX (Left side)
    if(r == null)
        return;
    if(r.data == K) {
        // Case 1
        if(r.left == null && r.right == null)
            return null;
        // Case 2
        if(r.left == null || r.right != null)
            return r.right;
        if(r.left != null && r.right == null)
            return r.left;
        // case 3
        if(r.left != null && r.right != null)
            r.data = return findMin(r.right);
            r.right = deleteNode(r.right, r.data);
        }
        if(r.data < K)
            r.right = deleteNode(r.right, K);
        else
            r.left = deleteNode(r.left, K);
    return r;
}
Node findMin(Node r){
    if(root.right == null)
        return root.left;
    root.   while(root.left != null)
            root = root.left;
        }
        return root;
}
```

Complexity: O(H), O(1)

Problem Given a root node of a tree, return true if it is BST, else false.

```
boolean ~~void~~ isBST (Node root) {
        if (root == null)

        root' = root.right;   // findMin (r.right)
        while (root'.left != null)
            root' = root'.left;

        root'' = root.left;   // findMax (r.left)
        while (root''.right != null)
            root'' = root''.right;

O(N²), O(1)   if (root''.data < root.data &&
                  root.data < root'.data)
                return true;

        return isBST (root.left) && isBST (root.right);
    }
```

way2 store the inordered traversal numbers & check if they are sorted or not.

```
bool isBST (Node r) {
    return fun (r, @INT_MIN);
}

bool func (Node r, int prevValue) {
    if (r == null)
        return F;
    if fun (r.left, r.data);
    if (p > r.data)
        return F;
    else
        p = r.data
    fun (r.right, p);
```

O(N+N), O(N) → extra array

O(1) → without extra space, compare directly

way3 Pass the valid range for each node & check

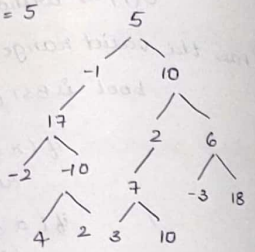
```
bool isBST (Node r, int a, int b) {
                    INT_MIN    INT_MAX
    if (r == null)
        return T;
    if (a < r.data && r.data < b)
        return T;
return isBST (r.left, a, root.data) && isBST (r.right, root.data, b);
                  ↓
              ~~INT~~
}
```

Date: 13.09.2019

CLASS - 17

Problem Given a binary tree & 2 nodes, find the length of the path between the given two nodes.

Length between -10 and 6 = 5




```
int path(Node root, Node x,
         Node y)
```

- Find path of node x from root.
  x = -10 { 17, -1, 5 }
- Find path of node y from root.
  y = 6 { 10, 5 }
- Store the last previous common node & delete the previous common nodes before it.

Way1 Store the path; List of the nodes in the path from x to root

```
boolean findPath(Node root, Node x, List lx) {
    if (root == null)
        return F;
    if (root.data == x)
        lx.add(root);
        return T;
    if (findPath(root.left, x, lx) ||
        findPath(root.right, x, lx)) {
        lx.add(root);
        return T;
    return F;
}
int Path(Node root, Node x, Node y) {
    List lx;
    List ly;
    findPath(root, x, lx);
    findPath(root, y, ly);
    // Remove the common nodes, leaving the previous or
       last common ancestor
    // Return the number of sum(lx, ly)
}
```

Complexity: O(N), O(N)

Problem Given a binary tree, find the maximum sum of node values. The tree contains negative integers too.

Brute force: $O(N^2 * N)$, $O(N)$ → space to store node values

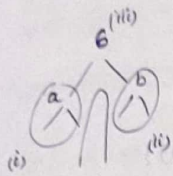
↳ store all the node values by preorder traversal

↓

Find max sum between every combination of two nodes

way2 Either use top-down or bottom-up approach to pick the max path

The path/max sum can be left subtree alone (or) right subtree alone (or) can include root in it




```
0          0          (global var)
//         //
void func(Node root, int cur, int m) {
    if(root == null)
        return;
    // TOP-DOWN APPROACH
    cur = cur + root.data;
    m = max(m, cur);
    func(root.left, cur, m);
    func(root.right, cur, m);
}
void solve(Node root) {
    if(root == null)
        return;
    int l=0, r=0;
    fun(root.left, 0, l);
    fun(root.right, 0, r);
    ans = Math.max(ans, root.data+l+r);
    solve(root.left);
    solve(root.right);
}
```

The above snippet is for Top-down approach, pre-order.

way3 Use bottom-up approach, by using pre-order traversal / post

TODO Complexity: O(N*N), O(1)

The possible paths where max-sum can be;

→ This still needed to update the ans as it can return the max sum

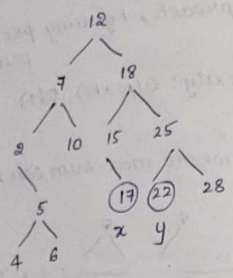
return Math.max(l, r) to its previous node to get the max sum using Bottom-up

```
int ans =
int func(Node root) {
    if(root == null) return 0;
    l = func(root.left);
    r = func(root.right);
    ans = Math.max(ans, root.data+l+r);
    return Math.max(root.data + Math.max(l,r), 0);
}
```

Complexity: O(N), O(1)

How what if all the elements in the BT are negative?
Is '0' the expected ans? NO
Update the logic, such that it returns the max (2 negative node values)

Problem Find the least common ancestor for a given BST



Node(x < y)

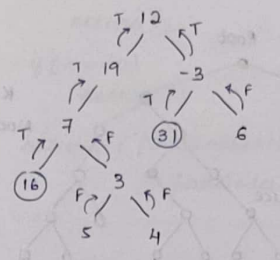
way1 Find the path / list of nodes from node x to root similarly find the path from node y to root store the path nodes in lx & ly. Remove the common nodes except the least common node.

Complexity : O(N), O(N)

way2

```
Node LCA(Node root, int x, int y) {

    if (x < r.data && y > r.data)
        return root;
    if (x < root.data)
        return LCA(r.left, x, y);
    return LCA(r.right, x, y);
}
```

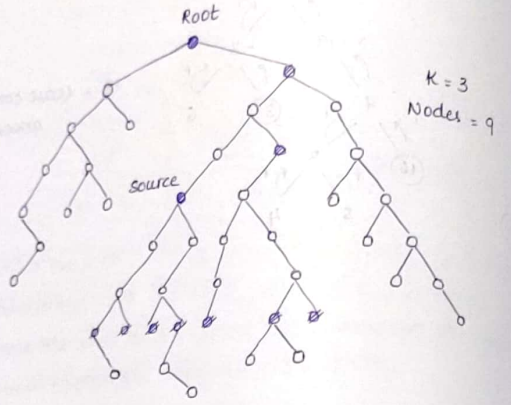
way3 Assume that the given tree is a binary tree.
Then use bottom-up approach using post-order traversal to check if x, y are present in its left & right sub-trees respectively.



12 = least common ancestor.

TODO

Problem Given a Binary tree, find the no of nodes from a source node at a distance K



Way1 O(N×N), O(N)

↓ Iterate to save all nodes

↳ store all nodes to traverse from source to x_i

↳ Find distance from each node to S

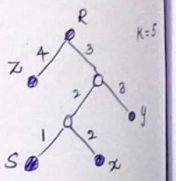
Way2 Choose the four different paths from the source node

⇒ func (S, 5) = 8

func (z, 3) = 3

func (y, 2) = 4

func (x, 1) = 2



Level order traversal at K^{th} level given root (OR) use

The below method returns the no of nodes below it at K^{th} level.

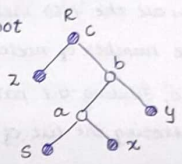
STEP1

```
int findNodesAtLevelK (Node root, int K) {
    if (root == 0)
        return 0;
    if (K == 0)
        return 1;
    return ( findNodesAtLevelK (root.left, K-1) +
             findNodesAtLevelK (root.right, K-1));
}
```

STEP2 The path of Source node from Root

L: [S, a, b, c]

Refer prev prob to create this list



L: { S, a, b, c }
0 1 2 3

f(S,5) ↓ f(b.right, 2) ↓ f(c.left, 1)

f(a.right, 3)

if (L(i) == L(i-1) → left) then call L(i).right

[K-index-1]

How How do you know which side of the subtree to be called?

- Consider the list of nodes in order
- As it denotes the path from R to S. Therefore

  if (a.left == S)

  then search for nodes in a.right

  else search for nodes in a.left
- The index / K' is calculated by [K-index-1]

STEP 3

```
ans = f(S, K);
for (int i=0 to L.size()) {
    if (L[i].left == L[i-1])
        ans += f(L[i].right, K-i-1);
    else
        ans += f(L[i].left, K-i-1);
}
```

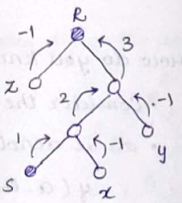
The function f(a.right, k) traverses different part of a tree. So, all the 'f'() methods combinely take O(N) times to find the number of nodes for a given level 'K'.

Also, finding the path from source S to root R takes O(H) time & storing the list of nodes.

complexity: $\boxed{O(N+N),\ O(H)}$

Way4 We can avoid taking extra space, by optimizing the space needed to store the list of nodes in path (S to R)

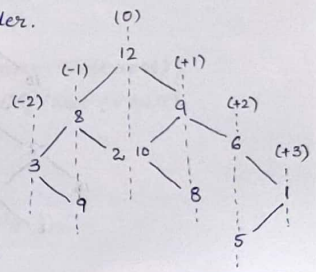
- Check if the node S is present in its left (or) right subtrees for every node in the tree.
- Return -1 if it is not found
- Return L+1 if it is found
- When found, make a call to other subtree to compute no of nodes at level K' where K' = K-L-1




```
int solve(Node root, Node s, int K) {
    if (root == null) return 0;
    l = solve(root.left, s, K);
    if (l != -1) {
        ans += func(root.right, K-l-1);
        return l+1;
    }
    r = solve(root.right, s, K);
    if (r != -1) {
        ans += func(root.left, K-r-1);
        return r+1;
    }
    return -1; // if l==-1 && r==-1
}
```

★ complexity: $\boxed{O(N+N),\ O(1)}$

Problem Given a binary tree, print the node values in vertical order from left to right. The elements in a vertical can be in any order.



ans: [3]
[8, 9]
[12, 2, 10]
[9, 8]
[6, 5] [1]

Way1 Compute the index of each node as shown in prev. Store the index & list of nodes of that index in a HashMap. Then iterate the HashMap to print nodes from left to right.

```
void func(Node root, int v, tt Map map){
    if(root == null)
        return;
    map[v].add(root);
    func(root.left, v-1, m);
    func(root.right, v+1, m);
}
```

TODO

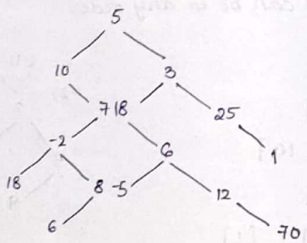
Complexity : $O(N * \log_2 N)$, $O(N)$

↳ Sorted HashMap

Way2 Avoid sorted HashMap by maintaining MIN & MAX vertical indexes, to iterate over hashkeys

Complexity : $O(N)$, $O(N)$

Problem Given a binary tree, print its LEFT VIEW & right view of the tree.



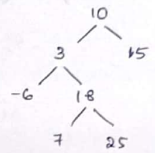
Maintain vertical order index & traverse using level-order

[5,0]
[10,-1] [3,1]
[7,0] [18,0] [25,2]
[-2,-1] [6,1] [1,3]

TODO Implement the logic for all 4 views of a tree.

Problem Implement pre-order traversal without using recursion

ans : 10, 3, -6, 18, 7, 25, 15



Use a stack, push root node
push right child
push left child
pop & print the value

```
main:
    IterativePreOrder o = new IPO(root);
    while(o.hasNext()){
        print(o.getNext().data);
    }

class IterativePreOrder {
    Stack<Node> s;
    public IterativePreOrder(Node root){
        s.push(root); // if(root != null)
    }
    bool hasNext(){
        return !s.isEmply();
    }
```

```
Node getNode() {
    Node temp = s.pop();
    s.push(temp.right);   // Add null check.
    s.push(temp.left);
    return temp;
}
}
```

Complexity : O(N), O(H)

↓

stack

At any point only one element of each level is stored

Problem Implement inorder iteratively.

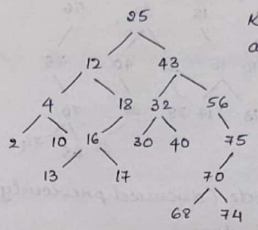
```
void Inorder(Node root) {
    if (root == null)
        return;
    while (!s.isEmpty()) {
        while (root != null) {
            s.push(root);
            root = root.left;
        }
        Node t = s.pop();
        Print(t.data);
        root = t.right;
    }
}
```

Date 20.07.2019

CLASS - 19

CLASS-18 Full day Lab Session

Problem given a BST, find the ceil & floor of a given number k. k can/cant be part of the tree.



K = 15
ans : floor : 13
ceil : 16

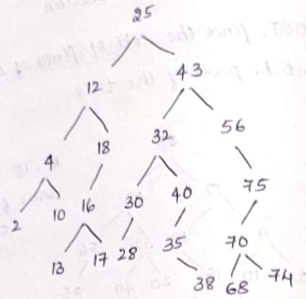
way1 Check for floor of the given number

Ex: K < 95 → left
12 < K → right (update ans)
K < 18 → left
K < 16 → left
13 < K → right (check & update ans)

```
void findfloor(Node root) {   // global var
                              ans = INT_MIN
    if (root == null)
        return;
    if (k < root.data)
        findfloor(root.left);
    if (k >= root.data) {
        ans = Math.max(ans, root.data)
    } findfloor(root.right);
}
```

Problem Given a BST, trim it such that it contains data b/w the given range.

[15, 70]
a b



Brute force: Delete a node (discussed previously)
Traverse the tree
Delete a node outside the range
Start traversing using the new/updated root node

Complexity: O(N(N+H)), O(1)

Way!:

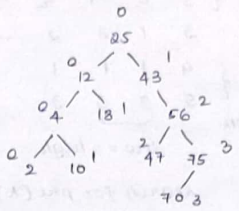
```
if (r.data < a)  check1 : trim (r.right)
if (r.data > b)  check2 : trim (r.left)
r.left = trim (r.left);
r.right = trim (r.right);
return r;
```

TODO

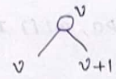
Complexity:

Problem Given a BST, print the nodes in diagonal order

ans: 25, 12, 4, 2
43, 18, 10
56, 47
75, 70



Way! Calculate the vertical order index for each node

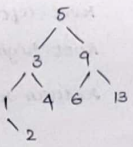


TODO

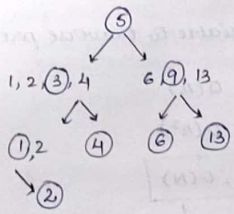
Similar to vertical order printing nodes problem

Problem Given preorder and inorder of a tree, find its postorder

Pre O: 5 3 1 2 4 9 6 13
In O: 1 2 3 4 5 6 9 13
ans: Post O: 2 1 4 3 6 13 9 5



Way!



Create a BST and do postorder traversal

k	low	high	idx	
0	0	11	6	→ createNode(a(k))
1	0	5	4	→ createNode(a(k))

	2	0	3	0	→ createNode (12)
base condition	3	0	-1	2	→ null
	3	1	3	2	→ createNode (18)
This can be the base condition also	4	1	1	1	
	5	3	3	3	

low == high

search for pre(K) in In[]
then increment K;

```
Node createBST (int[] PO, int[] IO, int low, int high){
    if (low > high)
        return null;
    int idx = search (PO[K], low, high);  // IOMap.get(PO[K])
    Node root = CreateNode (PO[K++]);
    root.left = CBST (PO, IO, low, idx-1);
    root.right = CBST (PO, IO, idx+1, high);
    return root;
}
int K=0; // Global variable to traverse preorder
```

Right skewed - $O(n)$

Left skewed - $O(n^2)$

Complexity : $O(N^2), O(N)$

↓ ↳ Create BST

depends if its a left skewed or right skewed or a normal BT

way2 If we store the inorder values in a HashMap, the search for a key can be optimized

Complexity : $O(N), O(N+N)$

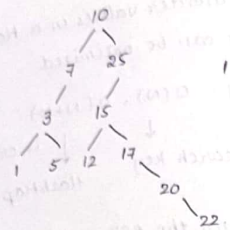
↓ search key ↓ HashMap ↳ create BST

way3 Instead of creating the BST, optimize further to print the elements in post order

```
void printPostOrder (int[] PO, int[] IO, int low, int high){
    if (low < high) return;
    int idx = IOMap.get(PO[K]);
    K++;
    printPostOrder (PO, IO, low, idx-1);
    printPostOrder (PO, IO, idx+1, high);
    S.O.P (in[idx])
}
main(){
    printPostOrder (PO, IO, 0, N-1);
    // Maintain int K=0 as global variable
}
```

Complexity : $O(N), O(N)$

Problem Given a BST, convert it into sorted doubly linked list and return the head node.



1 ⇄ 3 ⇄ 5 ⇄ 7 ⇄ 10 ...

way1
Brute force: Store the nodes in an array by traversing inorder
& then create a doubly linked list
Complexity : O(N), O(N)

way2 perform inorder traversal
Create a dummy node.

d.right = root
root.left = d
d = root;

d ⇄ ①

After creating the list, make ① as head & ①.left = null
The same is not needed for the last node, as it would be always/obviously NULL (25).right = null

```
void inOrder (Node root) {
    if(root == null)
        return;
    inOrder (root.left);
    d.right = root;
    root.left = d;
    d = root;
    inOrder (root.right);
}
```

complexity : O(N), O(1)

problem Given a sorted doubly linked list, convert it into a BBST

SDLL → BBST

```
Node func (Node head) {
    if (head.next == null)
        return null; head;
    Node midNode = findMid (head);
    Node temp = midNode.next;
    midNode.left = fun(head);
    midNode.right = fun (temp);
}
```

complexity : O(n log n), O(1)

$T(N) = 2T\left(\frac{N}{2}\right) + N \longrightarrow N \log_2 N, 1$

Master's Theorem

⭐⭐ This is done in TOP-DOWN Approach

IMP TODO Try to solve the same using BOTTOM-UP Approach using O(N) $T(N) = 2T\left(\frac{N}{2}\right) + 1 \longrightarrow N, 1$

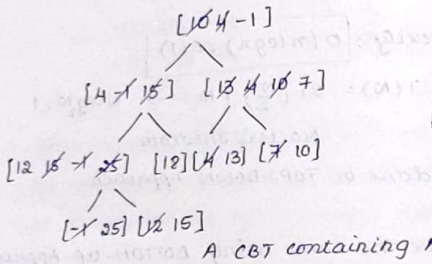
Solution	Insert(x)	getMin()	delMin()
Unsorted Array	1	N	N
Sorted Array	N	1	N
HashMap	1	N	1

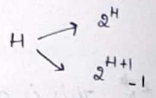
HashSet	1	N	1
Sorted Hashset (BBST)	log N	log N	log N
Singly LinkedList	1	N	N
Sorted SLL	N	1	1

Problem Create a BT such that it satisfies the below conds

- CBT (Complete BT)
- Node < ChildNodes

Ex: 10, 15, 13, 25, 18, 4, 7, -1, 12



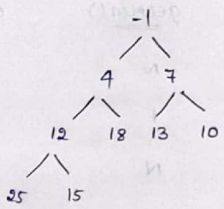


A CBT containing N nodes will have

height = $\log_2 N$

($2^H = N$)

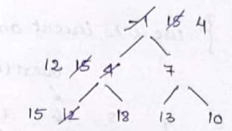
Final CBT looks like,



Now, delete Node -1

Solution	Insert (x)	getMin()	deleteMin()
Complete BT priority Queue	$\log_2 N$	1	$\log_2 N$

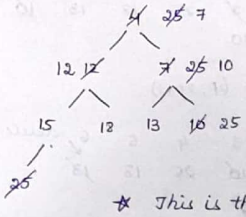
deleteMin() ⇒ -1



[The last element would be the new root]

[Perform same steps to satisfy the cond]

deleteMin() ⇒ 4



If (root > childNodes)

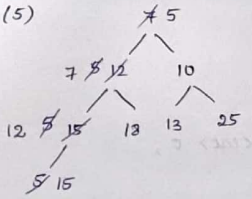
findMin (r.left, r.right)

then swap (r, minValR)

☆ This is the priority queue

MIN-HEAP

insert (5)



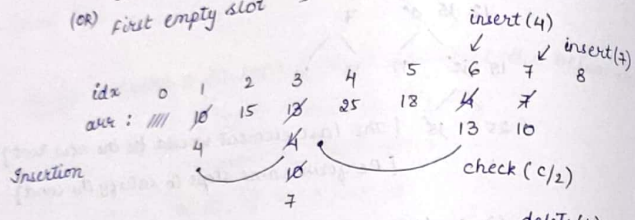
- How do we find the position of new node to be inserted
- Replace root & child nodes

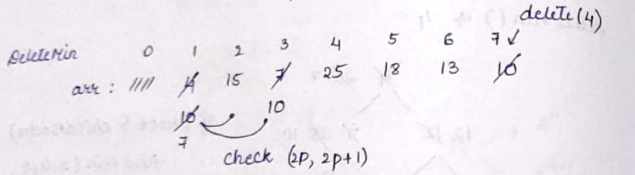
Store the elements of the Heap in an array starting from index 1

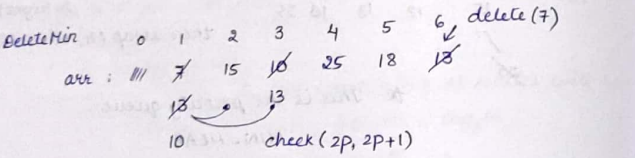
$p \rightarrow 2p, 2p+1$ child nodes of parent p

$c \rightarrow c/2$ parent of a child c

last occupied slot (OR) first empty slot } use it to insert an element







Pseudo code :

```
Class MinHeap {
    int ArrayList<int> v;
    MinHeap() {
        v = new ArrayList<int>();
    }
```

```
void insert (int x) {
    v.add(x);
    int idx = size() - 1;
    while ( idx != 1 && v[idx] < v[idx/2]) {
        swap ( v[idx], v[idx/2])
        idx = idx/2 ;
    }
}
```

★ PERCOLATE_UP

```
void deleteMin() {
    v[1] = v[size()-1];
    int idx = 1;
    while (idx < size()) {
        int temp = v[2*idx] > v[2*idx+1] ?
                   2*idx+1 : 2*idx ;
        swap ( v[idx], v[temp]);
        idx = temp;
    }
}
```

★ PERCOLATE_DOWN

★ JAVA : Inbuilt library Priority Queue (MIN_HEAP)
If we need to implement MAX_HEAP, we have to override the comparator method.

Problem Given an array of N elements, print the first K number of minimum elements (in any order)

Way 0 Brute force : $O(\cancel{N \log N})$, $(N * K)$

	0	1	2	3	4	5	6	7	8
ar_N:	10	3	12	-1	8	15	9	13	2

way1 Sort the array and return first K elements

Complexity: $O(N\log N + K)$, $O(1)$

way2 using Min-heap,

Complexity: $O(N\log N + K(1+\log_2 N))$, $O(N)$

way3 use Max-Heap to get the 'k' min elements

Complexity: $O(K\log_2 K + (N-K)(1+2\log_2 K))$, $O(K)$

Insert K elements to the max heap,

	0	1	2	3	4	
ar_K:	////	~~12~~	10	3	-1	← Insert (8)
		10	8	3	-1	← Insert (9)
		~~9~~	8	3	-1	← Insert (2)
		8	3	2	-1	

$K\log_2 K$ → Insertions of first K elements

$(N-K)$ → remaining elements

$1+2\log_2 K$ → Insert next upcoming element

K → extra space for max heap

way4 using the concept of quick sort, select a pivot & aggregate the $E_i < P_k < E_j$

if $(k < k')$ → search E_j, $k-k'$

else → search E_i, k

The performance vary depending on the pivot selection.

10 3 (12) -1 8 15 9 13 2

(P = 12)

10 (3) -1 8 9 2 (12) 13 15

(P = 3)

-1 2 (3) 8 9 10 → search $(k-k')$

(k' found)

Best case : $T(N) = 2T(N/2) + N \Rightarrow N\log_2 N$

Worst case : $T(N) = T(N-1) + N \Rightarrow N^2$

Average : $T(N) = T\left(\frac{9N}{10}\right) + T\left(\frac{N}{10}\right) + N \Rightarrow N\log_2 N$

↓ 90% of elements on left side

↳ 10% of elements on right

Quick sort

This is the approximation considered for avg case

Best Case : N

Worst case : $N(N-K) \Rightarrow T(N) = T(N-1) + N$

Average : $N \Rightarrow T(N) = T\left(\frac{9N}{10}\right) + N \approx N$

The above complexities are for Quick Select approach

$N(N-K)$ — K — select N-K pivots

Quick Select

Problem Given an ~~array, find its~~ K - sorted array, find the elements

	0	1	2	3	4	5	6	7	8
arN:	5	8	2	10	18	12	15	20	11
arN:	2	5	8	10	11	15 12	15	18	20

K=4 → The difference b/w the original index & new sorted index should be at max 'K'

E(8) = 11 Before

E(4) = 11 After

8-4 = 4 (K=4) condition satisfied

way1 Sort the array.

$O(N\log_2 N)$, $O(1)$

way2 Use a min_heap to store the first K+1 elements

Complexity : $O(N\log_2 k)$, $O(k+1)$

Problem Given a 2D array, print the elements in sorted order. Row-wise sorted matrix N*M

Mat_{N*M}			
3	10	18	20
-1	5	7	15
2	12	23	25
-4	3	7	10
5	10	16	24

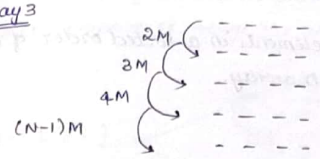
way1 : $O(NM + NM\log_2 NM)$, $O(NM)$

Convert to 1D array & sort the elements

way2 Insert all the elements into a MIN-HEAP

$O(NM\log_2 NM + NM)$, $O(NM)$

way3



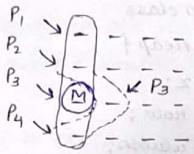
Merge 2 sorted rows each time

$= M(2+3+4+\ldots N+1-1)$

$= \frac{N(N+M)}{2}$, $M-M$

Complexity : $O(N^2 * M)$, $O(MN)$

way4 Maitain 'N' pointers, find the min element. Once min element found, move the pointer to next element & find the min.



pointer[N] = {P1, P2, P3, P4}

Complexity : $O(N * NM)$, $O(N)$

↳ Finding min element

↳ Repeat for NM elements

↳ pointer array

way5 Push the first column elements to a MIN-HEAP, delete the min, push the next element that is deleted.

How do you maintain the index of the deleted item ? As, we need to push the next element beside the deleted ones.

Date 21.09.2019

CLASS - 20

Problem Contd...

Find / Print the elements in a sorted order of a row-wise sorted 2D array.

Solution Contd... way5

We need to store the row & column of each element that is pushed to the MIN-HEAP either by the following ways:

- Use a Pair

  Pair<int, Pair<int, int>>

- Use a custom class

```
Class MinHeap {
    int x;
    int row;
    int column;
    @Override comparator()
}
```

NOTE we need to push the element beside the deleted min element from the heap.

Complexity : $O(NM * \log_2 N)$, $O(N)$

↓ ↳ Heap

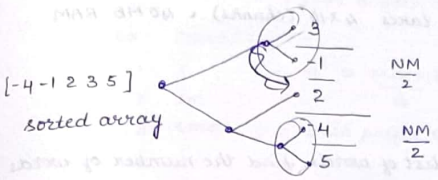
Insert N elements & delete min & push next

way6

NOTE Sorted HashMap uses iterative approach (inOrder traversal) to find the MIN/MAX element.

Internal implementation uses BBST

consider the first column elements & split them based on rows then compare & merge the elements.



$T(N) = 2T\left(\frac{N}{2}\right) + NM$

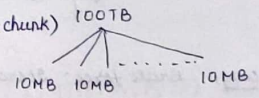
Complexity : , $O(NM)$

* In this method, we are just changing the approach to merge two sorted rows (similar to approach/way3)

Problem Given a 100TB of integer data, sort them using 100MB RAM space.

10^7 B → 100MB (10MB chunk)

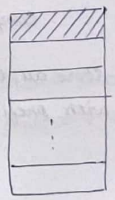
10^{14} B → 100TB



we get $\frac{10^4}{10^7}$ B = 10^7 chunks

Refer to solution (way 5) of previous problem & apply the same here

- Divide the data into 10^7 chunks of 10MB each.



- The space required to store each chunk takes
  int - 4bytes
  $4 \times 10MB = 40MB$
- Push each/first element of each chunck into MIN-HEAP
  it takes 4×10^7 (chunks) = 40MB RAM

Problem Given a list of words, find the number of words starting with query Q.

N words: intern, smart, stem, stick, slim, slack, interview, algorithm, get, penpencil

$1 \le len(w) \le M$

Q: sml = 0 (words starting with 'sml')
st = 2
sl = 2

way1 Brute force: Iterate over each word in the list and count the no of words starting with 'Q'

Complexity: $O(Q * NM), O(1)$

way2 Store all the prefixes of all the words into a hashMap with prefix as string & count as int

As string is not a primitive datatype of so, comparing strings in a hashMap would take $O(M)$.

Complexity: $O(N * \frac{(M+1)M}{2} + QM), O(NM^2)$

$\approx O(NM^2 + QM), O(NM^2)$

Ex: smart

1 s
2 sm
3 sma
⋮ ⋮

$M^2 \approx \frac{M(M+1)}{2} \approx 1+2+3+\cdots M$

Each prefix takes

way3 Similar to the finding frequency problem, apply BS here.

- Sort the list of words
- Maintain two pointers P_1 & P_2 to point to start & end words starting with query Qi
- Use binary search to find start index P_1 & end index P_2 in the list of sorted words.
- The frequency / count is calculated by $(P_2 - P_1 + 1)$

```
P1 = 0, P2 = N-1
for (int i=0 to N-1)
    P1 = BS1 (L, q[i], P1, P2, i)
    P2 = BS2 (L, q[i], P1, P2, i)
return P2 - P1 + 1
```

we are performing M Binary searches to find the count of each query. And sorting takes N words * M size space.

Complexity: $O(MN\log_2 N + Q*M\log_2 N)$, $O(N*M)$

↓ Sorting the list of N words of size M

↓ HashMap or ... to store sorted ...

Q queries take $M * \log_2 N$ times to compute the count

M - length of each word

$\log_2 N$ - find pointers p1 & p2

way4 Using the hash code, compute hash code for each prefix in a word using rolling hash technique.

Ex: intern

i → $i * prime^j$ → h1

in → $h1 + n * prime^j$ → h2

int → $h2 + t * prime^j$ → h3

i.e Rolling hash technique.

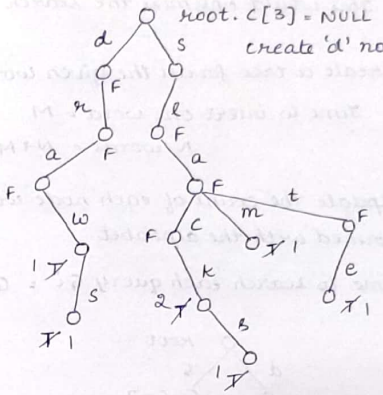
We should take of collisions / minimize collisions by taking a good hash function.

This approach involves,

- Rolling hash
- Good hash function
- Precomputation
- Handle collisions

Complexity: $O(NM + QM)$, $O(NM)$

way5 Using trees, create (26) child nodes, assuming that the list of words only contain lower case alphabets.



Words:
draw
slack
slacks
slam
slate
slack

HOW How do you fix the WA ?

- Count the number of leaf nodes
- Mark if the word is given in the list or not

Ex: slack , slacks

boolean isWord = 'T'

- Marking T/F doesn't help if there are duplicate

Ex: slack, slack

int count = 2 at that node 'K'

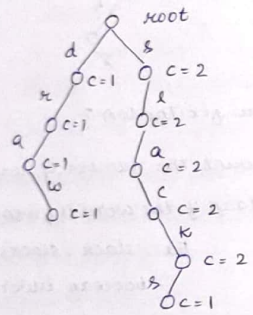
HOW How do you fix the TLE ?

- count the number of leaf nodes.

This would take same time as bruteforce

- Update the count of each node such that it indicates the no of words starting with Qi this would optimize the search

Solution
- Create a tree for all the given words
  Time to insert one word = M
  N words = N*M
- Update the count of each node with no of words formed with the alphabet.
- Time to search each query Qi = Q*M



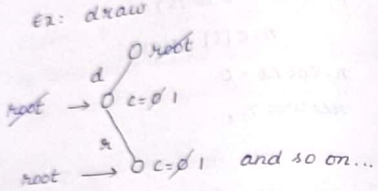

```
class Node {
    Node c[26];
    int count;
}
Node CN() {
    Node n = new Node();
    for(int i=0 to 25)
        n.c[i] = NULL;
    n.count = 0
    return n;
}
MAIN:
    Node root = CN();
    loop(N)
        Read(w)
        insert(root, w);
    loop(Q)
        Read(w)
        print Query(root, w);
```

IMP ★ This is called TRIE data structure, which has the below complexity to find the no of words starting with qi from a list of N words of length M.

★ Complexity: O(N*M + Q*M), O(N*M)

```
void insert(Node root, String w) {
    for(int i=0 to w.length) {
        int idx = w[i] - 97;
        if(root.c[idx] == null)
            root.c[idx] = CN();
        root = root.c[idx];
        root.count++;
    }
}
```

Ex: draw



and so on...

```
int printQuery (Node root, String w) {
    for (int i=0 to w.length) {
        int idx = w[i] - 97;
        if (root.c[idx] == null)
            return 0;
        root = root.c[idx];
    }
    return root.count;
}
```

Problem Given a 2D array with 1's and 0's, count the number of distinct rows in the matrix.

ar_{NM}

0 0 1 1 0 1	①
0 1 0 1 0 1	②
0 0 1 1 0 1	①
1 0 1 1 1 0	
0 1 0 1 0 1	②

ans = 3 distinct rows

way1 Brute force: check each row with all other rows

Complexity : $O(N * MN)$, $O(1)$

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

way2 Store each row in a Hashset

complexity: $O(NM)$, $O(NM)$

way3 If the columns size is less than 64, then we can convert the row elements to a decimal notation.
Store these decimals values to a Hashset which gives the count of distinct elements as its size();

Complexity : $O(N*M)$, $O(N*M)$
$+N$

N*M - convert to decimal
N - Hashset

way4 Use TRIE datastructure to count the unique paths
Maintain a global variable to update ans.
Increment ans only if atleast one new node is created while inserting the row data into the tree.

Complexity : $O(NM)$, $O(NM)$

Problem Given an array of size N, find the max XOR value of any two elements

ar_N : 12 5 18 29 7 4 16

$$\max_{i,j} (ar(i) \wedge ar(j))$$

way1 Brute force: Compute XOR for every pair & find ans.

Complexity: $O(N^2)$, $O(1)$

way2 Consider each element in the array,

```
          MSB       LSB
a =       0 0 1 0 1   (5)
18
29  <->   1 1 1 0 1   (29)
16
          ---------
          1 1 0 0 0   (24)
```

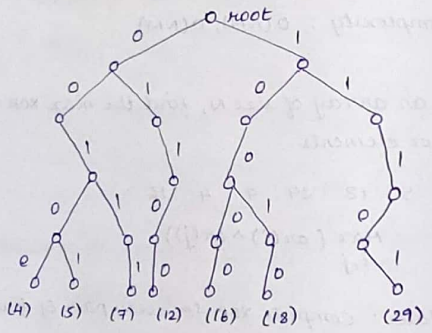
If we have 0 at MSB, shortlist all the elements that have 1 as its MSB to get max output.

Finding the set of possible answers as 'b' itself take time to check each bit

Complexity: $O(30 * N^2)$, $O(N)$

way3 Insert the array elements into a binary TRIE and fix the element (a) & find element (b)

arr : 12 5 18 29 7 4 16




```
          0 1 2 3 4
a =       0 0 1 0 1   (5)
b =       1 1 1 0 1   (29)
```

Start with 0^{th} bit, if its 0, traverse the tree with path 1
if there's no path with 1 goto path 0

NOTE All the elements should have same number of bits to construct TRIE

Constructing the TRIE will take $30 * N$ time
Fixing each bit of a and traversing the TRIE takes $30 * N$
The space of the TRIE would be $30 * N$

Complexity : $O(30N + 30N)$, $O(30N)$
$\approx O(N)$, $O(N)$

```
void insert (Node root, int x, int m) {
    for (int i = 0 m → 0) {
        b = checkBit (x, i) ? 1 : 0;
        if (root.c[b] == null)
            root.c[b] = createNode();
}
```

TODO `int query (Node root,`

Problem Find the maximum subarray XOR from a given array of size N.

ans:

Way1 Brute force: generate all combinations ~~in (2^N) times~~ & calculate the XOR, maintaing the ans-max

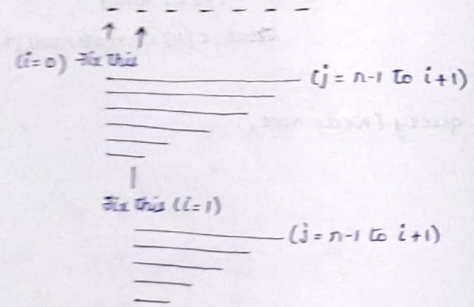
Complexity: $O(N^3+N)$, $O(1)$

Way2 Update the second loop to avoid 3rd loop in the above approach.

Complexity: $O(N^2)$, $O(1)$

Way3 Consider a subarray (i,j) then its xor can be calculated by $XOR(i,j) = XOR(0,j) \wedge XOR(0,i-1)$

fix the index 0 as its ending index of the subarray and compute the XOR



idx = 3

[0:3] = (0,3) ^ 0
[1:3] = (0,3) ^ (0,0)
[2:3] = (0,3) ^ (0,1)
[3:3] = (0,3) ^ (0,2)

consider this as 'a'

idx = 4

[0:4] = (0,4) ^ 0
[1:4] = (0,4) ^ (0,0)
[2:4] = (0,4) ^ (0,1)
[3:4] = (0,4) ^ (0,2)
[4:4] = (0,4) ^ (0,3)

Insert these elements to TRIE and find 'b' to get max XOR

```
int a = 0
for (int i=0 to N) {
    insert (root, a);
    a = a^ar(i);
    ans = max (ans, find (root, a));
}
```

* Complexity: $O(N), O(N)$

Problem Given a string and a list of substrings, partition the string such that all the partitions cover the list of substrings given.

L: { "abc", "z", "cdx", "yz", "xyz", "cdz", "ab" }

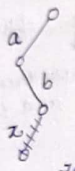
str: abxcdxyzabc

Recursion + Backtracking + TRIE → T/F, c = 2

abxcdxyzabc

is 'abx' a valid string ?

Extract the substring & check with list of words given.



TRIE contains the list of words/ substrings given.

abx doesn't exist

For each partition, check if there's a pair

TODO

Problem: given an array of size N, find the median of the elements [0:i] subarray

	0	1	2	3	4	5	6	7	8
arr :	5	3	10	15	18	2	4	25	7
ans :	5	3	5	5	10	5	5	5	7

$\forall_{i=0}^{N-1}$ arr[0:i] : MEDIAN

way1 Brute force : Perform insertion sort for every subarray to another array

There are only 'N' subarrays as first index is fixed. Then find the mid element from the sorted array

Complexity : $O(N^2)$, $O(\frac{1}{N})$

inplace sorting algorithm

way2 sort each subarray elements & find median

complexity : $O(N \log_2 N * N)$, $O(N)$

way3 Use Min Heap and Max Heap to find the median, such that

(i) $x \in L, R$

(ii) $|L| - |R| = [0, 1]$

If the difference is more than 1, delete the max element of L and insert it into R.

If the difference is less than 0, delete the min element of R and insert it into L.

Trace out the elements & find median,

L	R	M	
5		5	diff = 1
~~5~~, 3	5	3	diff = 2
3, 5	~~5~~, 10	5	diff = -1
3, 5	10, 15	5	diff = 0
3, 5, 10	~~10~~, 15, 18	10	diff = -1
2, 3, 5, ~~10~~	10, 15, 18	5	diff = 2
2, 3, 4, 5	10, 15, 18	5	diff = 1
2, 3, 4, 5	10, 15, 18, 25	5	diff = 0
2, 3, 4, 5, 7	~~7~~, 10, 15, 18, 25	7	diff = -1
[MAX_HEAP]	[MIN_HEAP]		

Complexity :

Date
29-09-2019

CLASS - 21

Problem Given a string, and a list of sub strings, check if each substring exist is contained in the given string

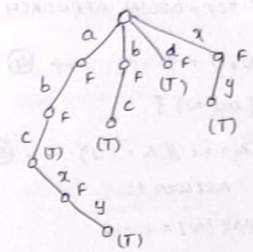
Contd... [refer previous class]

Find the number of ways to partition the string such that each partition is a valid word given in the list.

way1:

```
int func(String str, int N, int idx, Node root){
    if(idx==N)  // last
        return 1;
    int ans=0, Node troot = root; // global trav
    while(idx<N){
        int c = str(idx) - 97;
        if(root.child[c]==NULL)
            return ans;
        root = root.child[c];
        if(root.flag=='T')
            ans = ans + func(str, N, idx+1, troot);
        idx++;
    }
    return ans;
}
```

// Construct a TRIE with the list of given words



Problem Implement fibonacci series.

ar_N: 1 1 2 3 5 8 13 .... $N \geq 1$

```
int fun(int N){
    if(N==1 || N==2)
        return 1;
    return fun(N-1) + fun(N-2);
}
```

Complexity : $O(2^N)$
$O(1)$

```
int func(int N){  * // BOTTOM-UP APPROACH
    int ar[N+1];  // ar[3];
    ar[1] = ar[2] = 1;
    for(int i=3; i<=N; i++)
        ar[i] = ar[i-1] + ar[i-2];
    return ar[N]
}
```

Complexity : $O(N)$,
$O(N)$

Tabulation

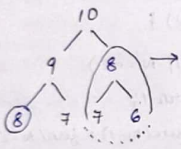
The recursion approach takes more time because we compute each 'N' value twice. To optimize this, we must compute & store the result in an array. Then use this value next time.

Optimized Approach: * TOP-DOWN APPROACH

```
int ar[N+1] = {-1};  ⟶ (4)
int fibR(int N) {
    if (N==1 || N==2)  ⟶ (3)
        return 1;
    if (ar[N]==-1)
        ar[N] = fibR(N-1) + fibR(N-2);
    return ar[N];
}
```

Recursion with Memoization

let say, N = 10



⟶ we avoid calculating this N' by storing its value

ar(i) → ith fabonacii number → (1)

→ ar(i-1) + ar(i-2) → (2)

(1) → Dynamic Program State

(2) → Dynamic Program Expression

(3) → Base Condition

(4) → Dynamic Program Table

Calculate time → # States * Time (state)

(No of states) (Time for single state)

Complexity: O(N), O(N)

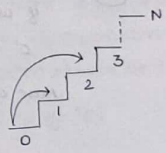
* Dynamic Programming

we apply this concept where there are overlapping sub-problems.

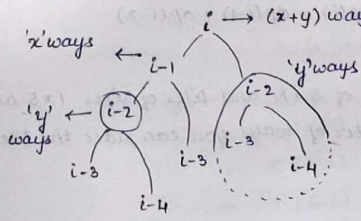
Once the sub-problem is identified, find the following

- DP state
  dp(i) = ith fabonacci no
- DP Expression
  dp(i) = dp(i-1) + dp(i-2)
- Base condition
- Implement the program either by using
  - Tabulation (BOTTOM UP) : iteratively
  - Recursion using Memoization (TOP DOWN)

Problem Given N steps, find the number of ~~steps~~ ways to N^{th} step if you are allowed to take either 1 step or 2 steps at a time.



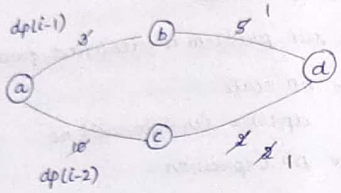
way1 To reach the i^{th} step, we can either come from $(i-1)^{Th}$ step or $(i-2)^{Th}$ step

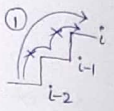


Overlapping sub-problem

dp(i) → # ways to reach i^{th} step

→ dp(i-1) + dp(i-2)





Ignoring the other way as it is already covered in dp(i-1)

No of ways to reach d → {a, b, d}

→ 3 * 5 = 15 ways

→ dp(i-1) * 1 ways

No of ways to reach d → {a, c, d}

→ 10 * 2 = 20 ways

→ dp(i-2) * 1 ways

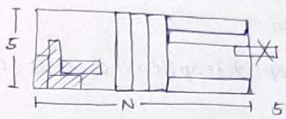
why

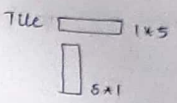
dp(0) = 1 but not dp(0) = 0, because when N = 0 if we can reach the 0^{th} step in one way. If we say '0' we indicate that there is no way to reach 0^{th} step

Base condition → dp(0) = 1 and dp(1) = 1

$$\forall_{i=2}^{N} \; dp(i) = dp(i-1) + dp(i-2)$$

Problem Given a floor of 5 * N and tiles of sizes 1 * 5 and 5 * 1 find the number of ways you can place the tiles to cover the floor.





As the vertical units are constant (5), consider the horizontal units which decides the no of ways to fill the floor

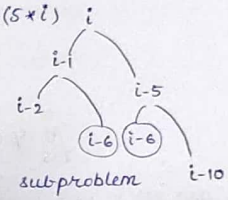
We can either place a vertical tile (i-1)

(OR)

can place a horizontal tile (i-5)

dp(i) → # ways to fill a floor of size (5 * i)

dp(i-1) * 1 + dp(i-5) * 1



Base condition

dp(0) = dp(1) = dp(2) = dp(3)

= dp(4) ⇒ 1

why

why dp(0) = 1 ? when dp(5) is calculated, it has 2 ways to fill the floor. But when dp(0) = 0, the dp expression returns 1 which is not correct, so dp(0) should be '1'.

dp[N] → can be reduced to dp[6]

dp[6] = { 1, 1, 1, 1, 1, - }

dp		idx
5	→	5 (4,0)
6	→	0 (5,1)
7	→	1 (6,2) → (6%6, 2%6) → (0,2)

dp expression :

$$\forall_{i=5}^{N} dp(i\%) = dp((i-1)\%6) + dp((i-5)\%6)$$

Problem Given the previous problem, consider that tiles are coloured, black on the front side & white on the back side. Find the no of ways to fill the floor.

Solution

$dp(1) =$  = 2 ways

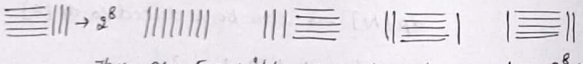
$dp(5) =$  B 2, W 2, B 2, B 2, W 2 = 2^5 ways

$\forall_{i=5}^{N} dp(i)$ → # ways to fill the tiles

→ $dp(i-1) * 2 + dp(i-5) * 32$

Considering the previous solution, can we say that the result would be $ar[N] * 2^N$? YES

$dp(8) = 5 = ar[N]$



There are 5 possible ways & each way has 2^8 combinations of placing the tiles

So, $dp(i) = ans(i) * 2^i$

Problem Given a number N, find the number of decimals which doesn't have 2 consequetive 1's

N = 4

0 0
0 1
1 0
1 1 (X)

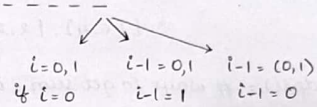
Total ways = 3

Find out the ans for N = 1, 2, 3, 4, 5 and observe the pattern

ans = {1, 1, 2, 3, 5, 8}

way1 It follows fabonacii series.

way2 construct a binary string of length N



The hit & trail may not work always, so derive the exp.

dp0(i) = # binary state of length i+1 such that it doesn't have consequetive 1's and ending with 0

dp1(i) = dp0(i-1) + dp1(i-1)

↳ # binary state of length i+1 such that it doesn't have consequetive 1's and ending with 1

dp0(i-1)

Base condition : dp0(0) = 1

dp1(0) = 1

int dp0[N]

int dp1[N]

ans: dp0[N-1] + dp1[N-1]

we can optimize the space by storing the previous state in 2 variables, such that,

current = c0 = p0 + p1

c1 = p0

Problem Given a 6-sided dice, find the number ways we can roll a dice to get a sum of N

N = 10 ans =

= {[6,4], [2,2,2,4], [3,3,4] ... }

dp(i) = # ways to get sum = i

= dp(i-1) + dp(i-2) + ---- + dp(i-6)

$= \sum_{j=1}^{6} dp(i-j)$

Base condition dp(1) = 1

dp(2) = 2 dp(5) = 16

dp(3) = 4 dp(6) = 32

dp(4) = 8

dp(3) = dp(2) + dp(1) + dp(0)

= 2 + 1 + 1 = 4

→ $\sum_{j=1}^{Min(i,6)} dp(i-j)$ (OR) $\sum_{j=1}^{6} dp(i-j)$ $(i-j>0)$

optimize the space, by taking ar[7] and use %. to get value / computed value for i^{th} index

Problem consider above problem, but each number on the dice costs.

Let's say, c(1) = 5 c(4) = 8

c(2) = 3 c(5) = 20

c(3) = 1 c(6) = 7

// cost array

find the minimum cost to roll the dice to get sum of N

NOTE Dynamic Programming

→ Overlapping subproblems

→ Optimal Substructures

dp(i) → Minimum sum cost to get a sum of i

= $\min_{j=1}^{6} (dp(i-j) + c(j))$ → $i-j \geq 0$

Base Condition = dp(0) = 0

If there no dice rolled, the cost is 0.

→ Prove the optimal substructure problem

→ define a dp state

→ write dp expression

→ base condition

→ define dp table size

→ compute time & space complexities

→ locate your answer

→ space optimization

Problem Given N houses and costs of each house if painted in Red, Green and Blue and no house can be painted in same colour, then find the total cost that is minimum

N :	1	2	3	4
R :	10	3 ✓	12	5 ✓
G :	6	9	✓2	3
B :	1 ✓	1	10	13

ans = 1 + 3 + 2 + 5
= 11

dpR(i) = Minimum cost of painting i houses with ith housed painted in Red (R)

= Min(dpB(i-1), dpG(i-1)) + R(i)

dpB(i) = Minimum cost of painting i houses with ith house painted in Blue (B)

= Min(dpR(i-1), dpG(i-1)) + B(i)

dpG(i) = Minimum cost of painting i houses with ith house painted in Green (G)

= min(dpB(i-1), dpR(i-1)) + G(i)

Base Condition : dpR(0) = 0
dpB(0) = 0
dpG(0) = 0

Compute dpR(i), dpG(i) and dpB(i) in one loop

$\forall_{i=1}^{N}$ [compute]

Ans : min(dpR(N), dpB(N), dpG(N))

complexity : O(N), O(1)

Problem given N jobs and costs of each machine to carry out the job, find the min cost to complete the costs, such that there is an additional cost if we want to switch the m/c.

N :	1	2	3	4	5
A :	3	2 → 6		12	14
B :	1	16	32	5 → 2	

(c on the 2 → 6 step, c on the 5 → 2 step)

N = 5
C = 100

Problem Find the maximum sub array sum.

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

N = 12

way1 Brute force: Generate all possible combinations

$O(N^2 * N)$, $O(1)$

way2 Carry forward the sum

$O(N^2)$, $O(1)$

way3 Maintain a prefix sum, start afresh whenever we encounter a negative value

arr: 3 5 -10 2 4 -1 6 3 -4

PS: 3 8 -2 2 6 +5 11 14 10

Iterate again to find the max sum.

$O(N)$, $O(1)$

way4 Use dp and define the state

$dp(i) \rightarrow$ max subarray sum till i including ith element

$\rightarrow dp(i-1) + ar(i)$

$\rightarrow \max(dp(i-1), 0) + ar(i)$

Base condition: $dp(0) = ar(0)$

ans: $\max_{i=0}^{N-1} dp(i)$

Space optimization: Use two variables, to store max_ans & the current sum

ans = ar(0), x = ar(0)

x = max(x, 0) + ar(i)

ans = max(ans, x)

Kadane's Algorithm

Answer: ans.

Problem Given an array size N, find the maximum subsequence such that chosen elements cannot be adjacent.

arr: 3 5 -10 2 4 -1 6 3 -4

ans = {5, 4, 6}

Solution Maintain a flag to track if we can include the current element or not.

```
MAIN:
    fun(ar, N, 0, 0, T);
way1
int ans = ar(0) or INT_MIN;
void maxSum(int[] ar, int N, int sum, int idx, bool flag)
    if (idx == N) {
        ans = max(ans, sum);
        return;
    }
    if (flag == 'F') {
        maxSum(ar, N, sum, idx+1, T);
    }
    if (flag == 'T') {
        maxSum(ar, N, sum + ar(idx), idx+1, F);
        maxSum(ar, N, sum, idx+1, T);
    }
}
```

Date
28.07.2019

CLASS-22

Problem Contd...

Find max subsequence array sum.

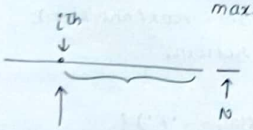
way 2 For every index, we can maintain 2 states,

dp(i) → dp(0) should n't consider i th Ele
dp(1) should / can consider i-th Ele

In the above approach, we pass 'sum' to every function call without maintaining it.

Instead of passing the variable 'sum', we can store its state at every function call & reuse it.

```
if (flag == 'F')
    dp0(idx) = maxSum(ar, N, idx+1, T);
if (flag == 'T')
    dp1(idx) = MAX( maxsum(ar, N, idx+1, F),
                    maxsum(ar, N, idx+1, T));
```



when you call the func dp(i) = dp(N) = 0
from idx = i ; it returns
the max subsequence sum from i to N-1

We are applying Recursion with Backtracking + Memoization

```
int maxSum(int[] ar, int N, int idx, boolean flag){
    if (idx == N)
        return 0;
    if (flag == 'F'){
        if (dp0(idx) == -1){
            dp0(idx) = maxSum(ar, N, idx+1, T)
        }
        return dp0(idx);
    }
    if (flag == 'T'){
        if(dp1(idx) == -1){
            dp1(idx) = Math.max((maxSum(ar, N, idx+1
                , F) + ar(idx),
                maxSum(ar, N, idx+1, T));
        }
        return dp1(idx);
    }
}
```

* dp0(i) → # max non-adjacent sub seq sum
sum from 0 to i
dp1(i-1)
(OR) for(i to N-1)
dp1(i+1)

* dp1(i) → # max non-adjacent sub seq sum

$$\text{MAX}\begin{bmatrix} dp0(i-1) + ar(i) \\ dp1(i-1) \end{bmatrix}$$

way3 Maintain dp(i) which stores the max subseq sum

ar_N : 3 5 -10 2 4

3 5 -10+3 2+5

-7

3 5 -7 2+5 ← max(3,5)+2

3 5 -7 7 4+5 ← max(3,5,-7)+4

dp(i) : 3 5 -7 7 9

ans : 9 (max subseq sum)

dp(i) = dp(N-1) = # max non-adjacent subsequence array sum

- for(int j=0 to i) including ith element

$$\max_{j=0}^{i-2} (dp(j)) + ar(i)$$

complexity : $O(N^2), O(N)$

↓

Iterate over array el

find max sum/element till that element

way4 We can still avoid finding MAX element everytime to compute dp(i)

m = ar(0)

$$\forall_{i=2}^{N-1} \; dp(i) = m + ar(i); \quad m = MAX(m, dp(i-1));$$

i.e for (int i-2 to N-1) {

dp(i) = m + ar(i);

m = Math.max(m, dp(i-1));

}

* complexity : $O(N), O(N)$

Problem Find the longest increasing subsequence in a given array of size N.

ar_N : 3 10 15 7 12 4 5 20 9 13

ans : 5 {3, 4, 5, 9, 13}

way1 generate all subsequences & check for increasing order

complexity : $2^N * N$, 1

we can compute the max-length on the fly,

complexity : 2^N, 1

way2 ar_N : 3 10 15 7 12 ← ith element

possible sets = [{3, 7, 12}, {10, 12}, {3, 12}, {7, 12}]

ar_N :	3	10	15	7	12	4	5	20	9	13
length :	1	2	3	2	3	2	3	4	4	5

We use Optimal Substructure here.

$\forall_{i=0}^{N-1}$ $dp(i)$ → length (LIS) till i, including i

$$= \max_{j=0}^{i-1} (dp(j)) + 1$$

$ar(j) < ar(i)$

Complexity : $O(N^2)$, $O(N)$

way 3 Take an array, store the smallest element ending in the sequence of length i.

	0	1	2	3	4	5	6	7	8	9
A_N :	3	10	15	7	12	4	5	20	9	13

B_N : 3 ~~10~~ ~~15~~ ~~20~~ 13

~~7~~ ~~12~~ 9

4 5

How How do you find, which element to be replaced ?

Find the ceil of the current element & replace it

Ex : when the state of B_N is as below,

and the next element is 7

B_N : 0 1 2 3 4 5

3 ~~10~~ 15

ceil (7) = 10, so replace it

$ar(2) = 7$.

NOTE The B_N may not contain the list of elements that are increasing order. So, you cannot consider it as the final answer set.

Ans The index of the last element filled in B_N is the LIS

How How do you find the ceil (x).

The numbers in B_N are always sorted in this approach so, apply Binary search.

Complexity : $O(N \log N)$, $O(N)$

way 4 Using the DP approach, we can implement it using

$dp(i)$ and $p(i)$ → maintains index of last ele or parent index

Backtrack the dp computation

	0	1	2	3	4	5	6	7	8	9
ar_N :	3	10	15	7	12	4	5	20	9	13
$dp(i)$:	1	2	3	2	3	2	3	4	4	5
$p(i)$:	-1	0	1	0	3	0	5	4	6	8

start from last element, 13

= , 5 , 9 , 13

↓

This can be [20, 9]

But consider the parent (i) to get the element.

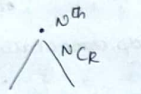
```
for (int i = 0 to N-1) {
    m = 0, p = -1;
    for (int j = 0 to i-1) {
        if (ar(j) < ar(i) && dp(j) > m)
            m = dp(j);
            p = j;
    }
    dp(i) = m + 1;
    p(i) = p;
}
```

$$\star \quad {}^{N}C_{R} = \frac{N!}{R!(N-R)!} = {}^{N-1}C_{R-1} + {}^{N-1}C_{R}$$

Consider RHS,

$${}^{N-1}C_{R-1} + {}^{N-1}C_{R}$$

we can say that, the number of ways of picking N^{th} El



(without) ${}^{N-1}C_{R-1}$ (+) ${}^{N-1}C_{R}$ (with)

Prove, $\frac{N!}{R!(N-R)!} = {}^{N-1}C_{R-1} + {}^{N-1}C_{R}$

$$= \frac{(N-1)!}{(R-2)!(N-R-2)!} + \frac{(N-1)!}{(R-1)!(N-R-1)!}$$

$$= \frac{(N-1)!}{(R-1)!(N-R-1)!}\left[\frac{1}{N-R} + \frac{1}{R}\right]$$

$$= \frac{N(N-1)!}{R(R-1)!(N-R)(N-R-1)!}$$

$$= \frac{N!}{R!(N-R)!}$$

Problem Implement the logic for finding ${}^{N}C_{R}$ using DP.

$dp(i,j) \rightarrow$ # ways to choose j items from i items

$= dp(i-1, j-1) + dp(i-1, j)$

When you store R, N in a 2D array, we see that current state always depends ~~fo~~ on its previous states

Lets say $(i,j) = (3,2) \rightarrow C$

state depends on $(i-1, j-1) \rightarrow (2,1) \rightarrow P_1$

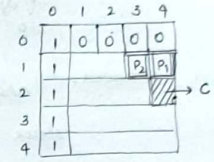
$(i-1, j) \rightarrow (2,2) \rightarrow P_2$

The Base Condition would be for (i=0 ; j= 0 to N-1)

and for (j=0 ; i=0 to N-1)

$\rightarrow \forall_{i=0}^{N} dp(i,0) = 1$

$\rightarrow \forall_{j=1}^{R} dp(0,j) = 0$



Answer : $dp[N,R]$

int $dp[N+1][R+1]$ → space

Complexity : $O(NR), O(NR)$

Solution Recursion with Memoization

```
int dp[N+1][R+1] = {-1};
fun (int N, int R, ~~int i, int j~~) {
    if ( i == N)
        return 0;
    if (j == R)
        return 1;
```

Space Optimization : We just need 2 rows and R columns.

$\forall_{i=0}^{N} \forall_{j=0}^{R}$ $dp(i\%2, j)$ → # ways to choose j items from i items

$= dp((i-1)\%2, j-1) + dp((i-1)\%2, j)$

Base condition would be,

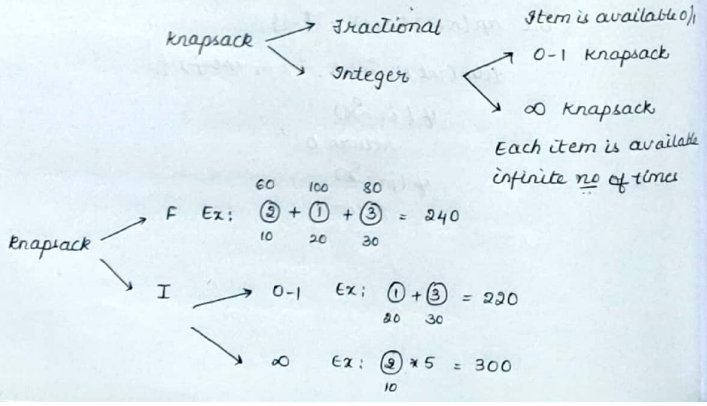
$\forall_{i=0}^{1} dp(i,0) = 1$ $\quad$ $\forall_{j=1}^{R} dp(0,j) = 0$

Answer, $ans = dp[N\%2, R]$

space = int dp[2][R+1];

complexity : $\boxed{O(NR),\ O(R)}$

Problem Given a list of quantities and their costs of each item. find the max profit by selling them for a given quantity 'k'



N = 1 2 3

w(i) : 20 10 30

v(i) : 100 60 120 $\quad$ K = 50

For fractional knapsack, we use greedy approach based on the ratio.

☆ complexity : $\boxed{N \log N}$

```
ans = 0
int i = 0 to N
    ans += min(K, w(i)) * R(i)
    K = K - w(i)
```

Find the ratio $R(i) = v(i)/w(i)$ and sort them. The first ratio at 0^{th} index gives us max profit.

Then check with 'K' value if it can completed with each quantity fraction.

☆ For Integer + 0-1 Knapsack,

Brute force : Generate all combinations with sum = k

Complexity : 2^N

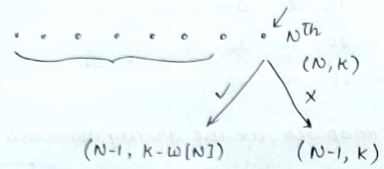
→ Can use overlapping subproblems approach

→ Also, we can use optimal substructures approach

N : 1 2 3 4 5

w(i) : 3 6 5 2 4

v(i) : 12 20 15 6 10



We get Max profit if either by considering N^{th} item or by NOT considering N^{th} item

Ans: Math.Max ((N-1, K-w[N]) + v[N]) , (N-1, K))

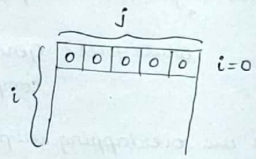
$\forall_{i=1}^{N} \forall_{j=0}^{K}$ dp(i,j) → # maximum profit to fill a knapsack of size k = 50 using first i items

→ max (dp(i-1, j-w(i)) + v(i) , dp(i-1, j))

j - w(i) ≥ 0 → j ≥ w(i)

Base condition:

$\forall_{j=0}^{K}$ dp(0,j) = 0



Ans: dp[N][K]

Complexity: O(NK), O(NK)

we can still optimize space, as the current state depends on its immediate previous state, the space can be

int dp[2][K]

★ Apply mod 1\2 to above equations.

Complexity: O(NK), O(K)

★ for 0-1 knapsack,

for the given weights and costs, fill the matrix,

MAX (dp(i-1, j-w(i)) + v(i) , dp(i-1, j))

SIZE → (i,j)		0	1	2	3	4	5	6	7	K = 7
	0	0	0	0	0	0	0	0	0	
✓	1	0	0	0	12	12	12	12	12	
X	2	0	0	0	12	12	12	20	20	
X	3	0	0	0	12	12	15	20	20	
X	4	0	0	6	12	12	18	20	21	
✓	5	0	0	6	12	12	18	20	22	[1,5]

j = k

i : N → 1

if (dp(i,j) == dp(i-1, j))

ith element is not considered (X)

else

ith element is considered (✓)

★ for infinite knapsack,

we can consider each quantity infinite times,

j ≥ w(i) → j ≥ x(w(i))

i.e j - x w(i) ≥ 0

Also v(i) → x w(i)

Ex: $dp(2,50) = dp(2,40) + 100,\ dp(1,50)$

$dp(2,40) = dp(2,30) + 100,\ dp(1,40)$

For infinite knapsack, the DP expression is as follows

$$\max(dp(i, j-w(i)) + v(i),\ dp(i-1, j))$$

(OR)

$$\max(dp(i-1, j-xw(i)) + xv(i),\ dp(i-1, j))$$

Three loops (i, j, x)

Problem: Given a set of denominations and their quantities. In this problem, each coin is available only once.

arr: 5 8 15 1 25 13

K = 37 [8, 15, 1, 13]

This is a 0-1 knapsack problem.

Way1: Brute force - Find all the subsequence sum = k

08-2019

CLASS - 23

Problem Contd... coin change problem

Solution: 0-1 Knapsack problem

dp(i,j) → If it possible to get a sum = j
using first i elements of the array

→ dp(i-1, j-ar(i)) || dp(i-1, j)

j ≥ w(i) || Referring to previous problem 0-1 Knapsack

The dp(i,j) returns a boolean value.

i.e whether the given 'w' = 37, 39 can be formed or not.

Problem Find the minimum number of elements required to get sum 'k' from the given array of size N.

dp(i,j) → Minimum no of elements to get sum = k/j
using first i elements of the array

→ (dp(i-1, j-ar(i)) + 1 , dp(i-1, j)) MIN

↓

Add one indicating that current element is considered in the list to get a sum j

Problem Find the MIN absolute difference between 'k' and the sum of i elements in a given array of size N

	0	1	2	3	4	5
Ex:	5	8	15	1	35	13

K = 35

so, possible answer could be,

↳ 35

↳ 5 + 8 + 15 + 1 + 13 = 42.

Ans = |42 - 35| = 7

The given input is not in sorted order

$|(sum(p1) - sum(p2))|$

Ex:2 possible sum formed from the given array of elem

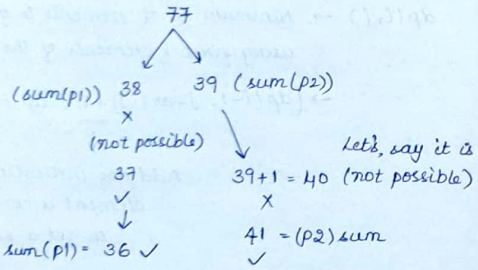
↳ 35 + 5 = 40 (sum(p1))

↳ 8 + 15 + 1 + 13 = 37 (sum(p2))

Ans = |40 - 37| = 3

Approach Consider total sum,

sum = 5 + 8 + 1 + 15 + 35 + 13 = 77



- Divide the total sum by 2 ⇒ sum/2
- iterate until we find sum(p1) & sum(p2) is possible from the given array elements.
- If not, decrement the sum by one & check.

(j, s-j)

Problem Other version of previous problem, find the number of ways to get a sum of 'K' from the given array

Ar_N: 5 8 15 1 35 13

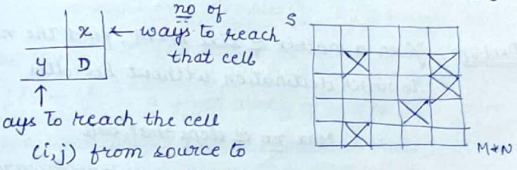
K = 21 → 5 + 15 + 1

→ 8 + 13

TODO

Problem Given a maze of size N * M, find the number of ways to reach destination from source.

Considering that there might be hurdles in the maze



we can move either right or down (2 directions allowed)

if (i,j → is Blocked)

then no of ways to reach that block cell

= 0

dp(i,j) → number of ways to reach the destination cell from source cell

→ dp(i-1,j) + dp(i,j-1)

→ 0, if (i,j) is blocked

Base Condition:

(i) If destination itself is blocked

(ii) The 0^{th} row is filled with 1
if there is a block, fill the remaining as 0

(iii) similarly, like the (ii) 0^{th} column is filled with 1
if there is a block, fill the remaining as 0

Space optimization:

Each cell is dependent only on its previous cells so it can be optimized.

Complexity: $O(N*M), O(N)$

Problem Given a matrix of size $N*M$, find the number of ways to reach destination without hurdles

Max no of steps that can be taken by moving forward or downward

$\Rightarrow N+M-2$

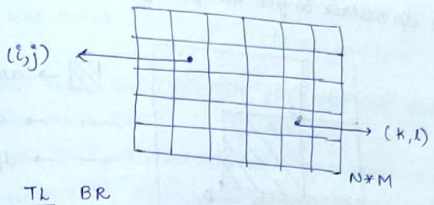
Ex: Matrix = 4X6.

steps = 8 → M-1 (5) or N-1 (3)

$^{N+M-2}C_{M-1} \Rightarrow {}^{N+M-2}C_{N-1}$

TODO

Problem Given a matrix of $N*M$ and indicies of source (i,j) and destination (K,L), find the sum of all the elements in that sub matrix formed by source & destination



TL BR

i j K L

Q ↓ $0 \le i \le K < N$

$0 \le j \le L < M$

Brute force: Iterate over all the elements $(i,j) \rightarrow (K,L)$ and find the sum.

Complexity: $O(Q*NM), O(1)$

way2 Use the HINT → solution for find the sum of array element from $(i) \rightarrow (j)$ in a given array of size N.

We used prefix sum to implement that.

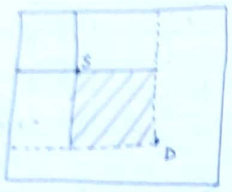
Similarly, use prefix sum for each row & find total sum by iterating over (j to L)

```
for (int x = i to K) {
    sum = sum + PS(x,L) - PS(x,j-1)
        (OR)
    sum = sum + P(x,L) - PS(x,j) + mat(x,j)
}
```

Complexity: $O(N*M + Q*N), O(N+M)$

$$dp(i,j) = \sum_{x=0}^{i}\sum_{y=0}^{j} mat(x,y)$$

Idea: use dp matrix to get ans for each query in O(1)



▨ → ans

..... → dp(k,l) D

—— → dp(i,j) S

$$dp(i,j) = \sum_{x=0}^{i}\sum_{y=0}^{j} mat(x,y)$$

Ans: $dp(k,l) - dp(i-1,l) - dp(k,j-1) + dp(i-1,j-1)$

$\sum_{x=0}^{i}\sum_{y=0}^{j} mat(x,y) \Rightarrow dp(i-1,j) + dp(i,j-1) - dp(i-1,j-1) + mat(i,j)$

The above is to compute the dp matrix

Ex: dp(2,5) → denotes the sum of all elements from (0,0) → (2,5)

** Complexity: O(N*M + Q*1), O(N*M)

Base condition: compute PS (prefix sum) for 0th row and 0th column

(OR) we can consider 2D → 1D array and compute PS prefix sum

compute row-wise prefix sum first, then use the same array and compute column-wise prefix sum.

Problem: Find the maximum sub-matrix sum, given an array of size N*M.

Approach: For every sub-matrix, find the sum and maintain the max-ans

Assume that we already have dp matrix with prefix sum such that dp(i,j) → denotes the sum of all elements from (0,0) → (i,j)

Find all combinations of cells; i.e sub matrices

```
for (int i = 0 to N) {
    for (int j = 0 to M) {
        for (int k = i to N) {
            for (int l = j to M) {
                // All possible combinations of
                //   (i,j) → (k,l)
                //   sub-matrices.
                // compute sum & maintain max-ans
            }
        }
    }
}
```

sum ⇒ dp(k,l) - dp(i-1,l) - dp(k,j-1) + dp(i-1,j-1) (handle this)

max-ans = MAX(max-ans, sum);

Complexity: O(N²*M²*1), O(N*M)

Way 2 Instead of maintaining a dp matrix, try computing the sum for every $(i,j) \rightarrow (k,l)$

Like in previous approach, generate all combinations of $(i,j) \rightarrow (k,l)$

Apply Kadane's algorithm to compute the sum of that sub-matrix.

Ex:

	5	3	-10	8	-10	-4	2
$(i,j) \rightarrow$	7	3	-1	4	7	-3	10
	10	-2	-45	3	-1	6	-2
	-1	8	5	10	8	-5	7 ← (k,l)

- compute sum of each column,

SUM → 16 9 -41 17 14 -2 15

Apply KADANE's algorithm and find the max sub array sum.

SUM → ~~16 + 9 - 41~~ (not considered)

→ 17 + 14 - 2 + 15 = 44

So, the sum of elements (MAX) in the submatrix of $(i,j) \rightarrow (k,l)$ is 44

Similary, find for all other combinations

```
int ans =
for (int l = 0 to N) {
    for (int k = i to N) {
        sum = 0;
        for (int j = 0 to M) {
            sum = sum + PS(k,j) - PS(i-1,j)
            // Add conditions
        }
    }
}
```

Problem Given an array of size N and two players A and B. A player can pick the first or last element and should make the max_sum. A player who ends up picking elements with max-sum wins the game. Assume that they are playing optimally.

ar: 10 3 8 4 12 3 15

Player 1 : picks 15 [10, 15]
2 : picks 10 [10, 3]
1 : picks 3 [3, 3]
2 : picks 8 [8, 3]
1 : picks 4 [4, 3]
2 : picks 12 [12, 3]
1 : picks 3 [3]

$sum_A = 15+3+4+3$

$sum_B = 10+8+12$

Let's compute the max_sum formed by picking the first (OR) last element using dp.

0 : N-1

↳ Calculate sum S

↳ S-M

Let's say you picked element M

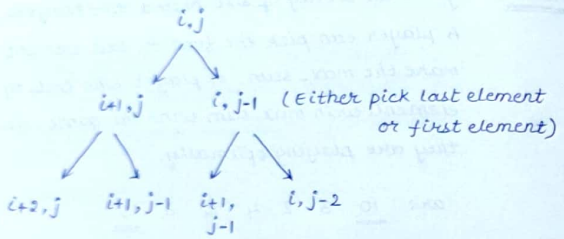
if (S' < S-M)

A wins

else

B wins

Overlapping subproblems



dp(i,j) → Maximum possible sum/score when playing in the subarray [i,j]

→ MAX { ar(i) + MIN { dp(i+2,j), dp(i+1,j-1) }, ar(j) + MIN { dp(i+1,j-1), dp(i,j-2) } }

Playing OPTIMALLY → we compute every possible way and select the best

GREEDY → Selecting the best at that point of time

Base condition :

i == j N = 1

i j N = 2

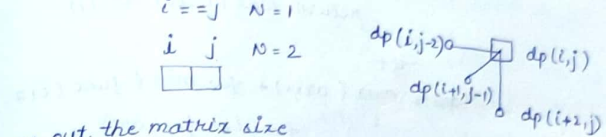
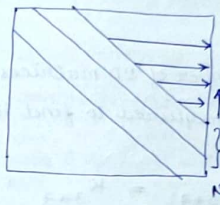


figure out the matrix size

find the order of filling the matrix

→ using Iteration

consider a N*N matrix, where base condition involves filling the diagonal elements as shown,



* → using recursion with Memoization

NOTE when you figure a problem is to be solved using dp, come up with dp Expression. Then when your are struck with base condition, i.e populating dp matrix (or) it becomes difficult to figure out the approch, Move to Recursion with memoization

```
int func (int i, int j, int ar[])
    if (i==j)
        return ar(i);
}
```

```
if (j == i+1)
    return max (ar(i), ar(j))

ans = max ( ar(i) + min ( func (i+2, j, ar),
                          func (i+1, j-1, ar)) ,
          ar(j) + min ( func (i+1, j-1, ar),
                        func (i, j-2, ar)))
// max_ans = Math.max (max_ans, ans);
return ans;
}
```

Problem Given 'N' number of 2D matrices, find the no of multiplications required to find its / their product.

$N_{3\times5} \quad M_{5\times7} = K_{3\times7}$

total multiplications required

$= 3 \times 5 \times 7 = 105$

	0	1	2	3	4	5	6	7	
ar_N :	3	5	7	2	9	10	4	12	$= R_{3\times12}$

N = 8

Let's consider the combinations of matrices that can be chosen to find the no of multiplications (MINIMUM)

Ex: $a_{3\times8}$ $b_{5\times12} \longrightarrow c_{3\times12}$

c_a (cost of multiplying) = 0 $\quad c_b = x$

(or)

no of multiplications) $\quad$ total = $0 + x + 3\times5\times12$

$[0,7] \longrightarrow$ $[0,0]$ $[1,7]$ $+ 3 * 5 * 12$

$[0,1]$ $[2,7]$

$[0,2]$ $[3,7]$ $[3,3]$ $[4,7]$

$[0,3]$ $[4,7]$ $[3,4]$ $[5,7]$

$[3,5]$ $[6,7]$

$[0,6]$ $[7,7]$

Overlapping subproblems

$cost(0,7) \rightarrow cost(i,j)$

$\rightarrow cost(0,0) + cost(1,7) + 3\times5\times12$

$\rightarrow cost(i,k) + cost(k+1,j) + ar(i) \times ar(k+1) \times ar(j+1)$

$dp(i,j) \rightarrow$ Minimum no of multiplications that can be performed to find the product of N arrays

$\rightarrow \min_{k=i}^{j-1} (dp(i,k) + dp(k+1,j) + (ar(i) * ar(k+1) * ar(j+1))$

Base condition : The $dp(i,i)$ should $0 \rightarrow cost(i,i) = 0$

All the diagonal elements would be 0

(or) if (i == j)
    return 0;

Try the above problem both iteratively & recursively.

* Ans : $\boxed{dp[0, N-2]}$

Problem Name : MATRIX CHAIN MULTIPLICATION

Problem Find the largest rectangular sub-matrix containing only 1's in a given matrix containing 0's and 1's

at N×M

0	0	1	0	1	1
1	1	1	0	1	0
1	1	1	1	1	0
1	1	1	0	0	0
0	1	0	1	1	1

Max sub-matrix with max no of 1's = 3 × 3

way1 generate all the sub-matrices and check

```
if ( n*m = sum[i,j → k,l])
    ans = (i,j) // store area_max
          (k,l)
```

Complexity: $O(N^3 \times M^2 + N \times M)$, $O(N \times M)$

generate all combinations of sub-matrices ↓ generate dp matrix (?) to store prefix sum

way2 similar to previously discussed question, use Kadane's algorithm.

Ex:

1	1	1	0	1 0 ← i
1	1	1	1	1 0
1	1	1	0	0 0 ← k

Complexity: $O(N^2 \times M)$,

way3 HINT Rectangular Area

Ex:

0	0	1	0	1	1	1	1
0	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1
0	1	0	1	1	0	1	1

Base row →

The above question can be interpreted as below,

(MAX) Rectangular Area in a histogram

0	0		0	1	1		
0				1	1		
				0	0		
2	3	4	3	1	1	4	4
0	1	0	1	1	0	1	1

Base row

0 4 0 4 2 0 5 5 → histogram values considering last row as base

This approach is an Out Of Box solution

- For each row as base row, compute the histogram values
- Calculate the max area using the optimized solution for this question (this → histogram area)

Complexity: $O(N \times M + N \times M)$, $O(N \times M + 3M)$

NOTE Refer to your TODO LIST for some more D.P imp questions

Date
04.09.2019

CLASS - 24

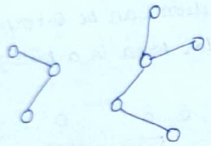
- Number of palindromic substrings
- Number of palindromic subsequences

TODO

Implement the above two problems using DP.

☆ Graphs

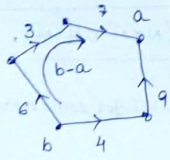
* Graph theory:



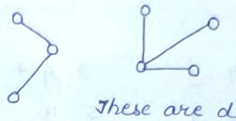
Types of graphs:

→ Undirected Vs Directed
→ Disconnected Vs Connected
→ Unweighted Vs Weighted
→ Acyclic Vs Cyclic
→ Simple Vs Complex

→ Connected Graphs should have a path from every pair of nodes. They are further classified as,
  → weakly connected {b → a}
  → strongly connected {a → b, b → a}



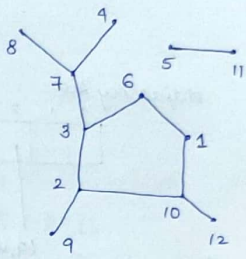
There is no path from a → b
weakly connected graph



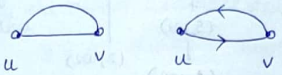
These are disconnected graphs

NOTE: A strongly connected graph is always a cyclic graph

Undirected, disconnected, simple weighted, acyclic graph



(u,v) or (u, v, w) → Notation



* Graph Representation

→ Adjacency Matrix
→ Adjacency List

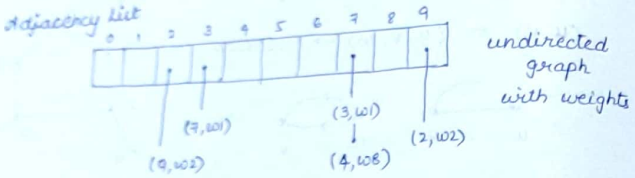
Problem Read the input and create an adjacency matrix.

```
Scan N and M for each UV (as M') {M' = N+1}
int m[][] = new int[N'][M']; {0}
// Given an undirected unweighted graph
for (int i=0 to M') {
    scan N → u
    scan M → v
    m[u][v] = 1  // = w if it is a weighted graph }
}   m[v][u] = 1
```

Given input.

N	M
12	11

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

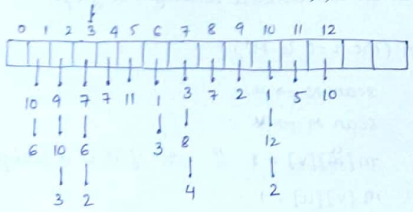


ArrayList < ArrayList < Objects >> → node, weight

(or)

ArrayList < Pair < Integer, Integer >>

```
for (int i=0 to M') {
    Read u, v, w;
    g.get(u).add(new Pair<int, int>(){
        .add(v,w);
    });
```



Problem: Given a source and destination, simply check if there a path exists or not.

consider the graph input in previous problem

Ex:

S	D	
3	12	→ T
2	11	→ F

* Graph Traversals
  - → Breadth First Search
  - → Depth First search

consider the above example and trace if there is a path.

S = 3, D = 12 bool visited[] = {F};

Queue: ~~3~~ ~~7~~ ~~6~~ ~~2~~ (3) ~~8~~ ~~4~~ ~~1~~ (3) ~~9~~ ~~10~~ (3)

(3) visited, (3) visited, (3) visited

(7) (7) (10) (6) (2) (1) 12 (2)

(7) visited, (7) visited, (10) visited, (6) visited, (2) visited, (1) v, 12 Found destination, (2) v

Return 'T' as there is a path from 3 to 12.

```
int s = 3, d = 12;
boolean visited[] = new boolean[N+1]{F};
boolean BFS(int[][] m, int S, int D) {
    Queue<int> q = new Queue<int>();
    q.push(s);
    visited[s] = 'T';
```

```
while (!q.isEmpty()) {
    // int size = q.size();
    // while (size-- > 0) {
        int d' = q.dequeue();
        if (d' == D) {
            return True;
        }
        for (int j = 1 to N) {
            if ( m[i][j] == 1)
                q.enqueue( m[i][j] );
            }       visited[j] = 'True' ;
    }
```

(OR)

use an adjacency list,

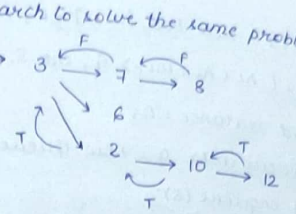
```
while ( !q.isEmpty()) {
    int u = q.dequeue();
    for ( int v : G(u)) {
        if ( visited [v] == F) {
            q.enqueue(v);
            visited [v] = T;
        }
    }
}
```

complexity: $O(|V|+|E|)$, $O(N+1)$

$O(N+1) \hookrightarrow O(|V|)$

$|V|$ → set of vertices + $|E|$ → set of edges

use depth first search to solve the same problem,



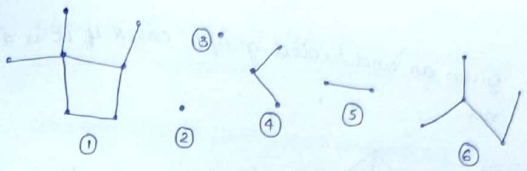

```
boolean visited [N+1] = {F};

boolean DFS ( ArrayList < ArrayList >> G, int S, int D, bool[] visited) {
    if ( S == D)
        return T;
    if ( visited [S] == T)
        return F;
    visited [S] = T;
    for ( int v : G(S)) {
        if ( DFS ( G, V, D, visited))
            return T;
    }
    return F;
}
```

Problem Given a graph and source & destination nodes, find the length of the shortest path in an undirected & unweighted graph.

Approach we should BFS here, as it is a level order traversal

```
bool vis[] = new bool [N+1] {F};
bool BFS ( AL<AL<int>> G, int S, int D) {
    int distance = 0;
    Queue<int> q = new Queue<int>();
    q.enqueue(S);
    vis[S] = T;
    while ( !q.isEmpty()) {
        int size = q.size();
        while (size-- >0) {
            int v = q.dequeue();
            if ( v == D)
                return distance;
            for ( int v' : G(v)) {
                if ( vis[v'] == F) {
                    q.enqueue(v');
                    vis[v'] = T;
                }
            }
        }
        distance++;
    }
}
```

Problem Given a disconnected graph, count the number of connected components and return the value.



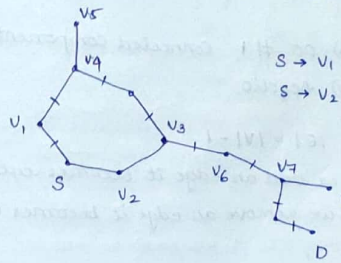
There are (6) connected components

```
bool vis = new bool [N+1] {F};
int count = 0;
for (int i = 1; i ≤ N; i++) {
    if (vis[i] == F) {
        BFS (G, visitedArray, i);
        count++;
    }
}
```

we can either use BFS or DFS, all we need to do is visit all the nodes & update the vis[] array.

Complexity : $O(|V|+|E|)$, $O(|V|)$

why? Why shouldn't we use DFS for shortest path algorithm



$S \to V_1$ distance = 8 {$V_3 \to T$}

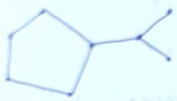
$S \to V_2$ {V_3 is not visited again & the shortest path can't be found}

Problem Given an undirected graph, check if it is a tree or not.

Ex: (i)

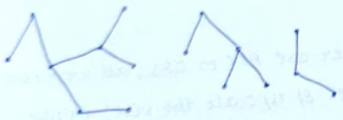
YES it is a tree

(ii)



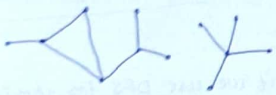
cyclic graph
NOT a tree

(iii)



FOREST
But not a tree

(iv)



Cyclic & disconnected graph
NOT a tree

a) CC #1 Connected component count = 1

b) Acyclic

Condition : $|E| = |V| - 1$

If we add an edge it becomes cyclic

If we remove an edge it becomes disconnected

Problem Given a directed graph, find the shortest path from source to destination.

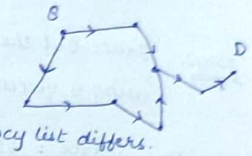
NOTE: Unweighted [Directed or Undirected] Graph

↓

SHORTEST PATH

↓

BFS

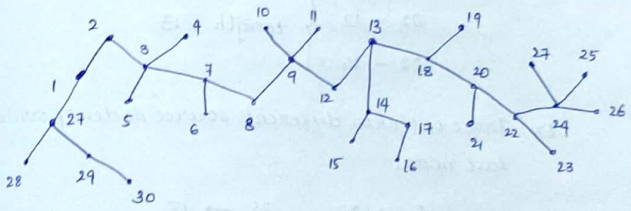


Inserting the graph data to adjacency list differs.

```
dist[N+1] = {};
dist[S] = 0
```

For a given tree, if there are Q queries asking for shortest path from S to D, then compute to all nodes from S & store them.

Problem Given an undirected graph, find the length of the longest path in it.



Way1 Brute force : Calculate longest path for every node to all other nodes.

```
for (int i = N) {
    BFS(G, i, visited);
}
```

Complexity: $O(|V|(|V|+|E|))$, $O(|V|)$ ↳ visited array

Perform BFS for every node as source — BFS to find longest path

Way 2 Trace out the longest path for any node as source using a queue & observe the last node that is removed.

Ex: Source = 25

Queue: ~~25~~ ~~24~~ ~~26~~ ~~23~~ ~~1~~ ~~22~~ ~~2~~ ~~3~~ (24) ~~14~~ (1) ~~21~~ ~~27~~ 5
~~6~~ ~~20~~ ~~8~~ ~~7~~ ~~9~~ ~~18~~ ~~10~~ ~~11~~ ~~19~~ ~~17~~ ~~16~~ ~~18~~ 12 13 15

- Run a BFS to get one of the ending nodes (s')
- Run BFS again with s' as source node to find the longest path.

22 - 13
22 - 12
22 - 15
} length = 13

Ex: Trace out with different source nodes & write the last node,

5 → 22 25 → 15
15 → 22 21 → 12
1 → 12 22 → 15

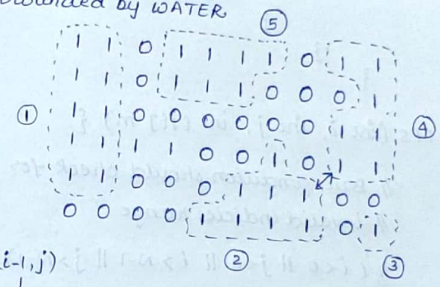
Now, pick any one node as s' from {22, 12, 15}
Run BFS (G, s', visited) to return longest path

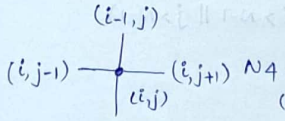
Complexity: $O(2(|V|+|E|))$, $O(|V|)$ ↳ visited array

Run BFS twice

NOTE In BFS, from whichever we pick a node as source we always end up traversing one of the END nodes in the last iteration. so definitely the last node that is removed from the queue is going to be an END node which is going to be a pat node in the longest path

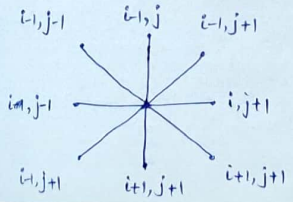
Problem Given a 2D matrix with 0's and 1's where 1 → LAND and 0 → WATER, find the number of islands, i.e LAND surrounded by WATER





N4 - Connectivity (or) N4 - Neighbourhood

Islands = 5



N8 - Connectivity {Merge ② & ④}
(or) N8 - Neighbourhood

Islands = 4

Approach Consider the given 2D matrix with 0's & 1's as a graph input (i.e adjacency matrix)
Now, Count the number of connected components which will give us the number of islands.

```
MAIN:
    int count = 0  // island count
    for (int i = 0 to N) {
        for (int j = 0 to M) {
            if (m[i][j] == 1) {
                count++;
                DFS (i, j, m);
            }
        }
    }

void DFS (int i, int j, int [][] m) {
    // Base condition should check for
    // invalid indicies range
    if (i<0 || j<0 || i>N-1 || j>N-1)
        return;
    m[i][j] = 0;
    // Making it as 0, acts as visited array
    DFS (i+1, j, m);
    DFS (i-1, j, m);
    DFS (i, j+1, m);
    DFS (i, j-1, m);
}
```

Instead of making 4 four recursive calls for each (i,j)

```
Let di[4] = {0, -1, 0, 1}
    dj[4] = {-1, 0, 1, 0}

for (int i = 0 to 3) {
    DFS (i + di[k], j + dj[k], m);
}
```

Similarly, store the values for N8 - Connectivity graph.

NOTE

Adjacency Matrix	Adjacency List
→ $O(N^2)$ space complexity	→ $O(N+M) = N^2$ (worst case)
	 Complete graph N^2 edges
→ Space consuming when the limits of graph are more.	→ This is preferred DS Ex: useful when there are 10^6 edges.

Date
10.05.2019

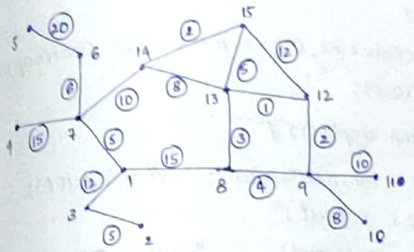
CLASS - 25

Problem Given a 8x8 chess board and a knight, check if all the cells in the chess board can be visited or not.

dx	-2	2	1	-1	-2	-2	-1	1	2
dy	1	1	2	2	1	-1	-2	-2	-1

```
void DFS ( int i, int j, int [][] visited) {
    if ( i<0 || j<0 || i > N-1 || j > N-1)
        return;
    visited [i][j] = 1;
    for(int l=0 to 7) {
        DFS (dx[l], dy[l], visited);
    }
}

MAIN: {
    DFS (startx, starty, visited);
    for ( int i=0 to 8)
        for (int j=0 to 8)
            if ( visited [i][j] == 0) {
                S.O.P ("NO")
                break;
            }
}
```

Problem Given an undirected weighted graph, find the shortest path.



Start = Node 7
End = Node 9

Maintain an array of edges and min. distance to reach them nodes

Initially, the distances are update to ∞ (INT_MAX)

Maintain one more visited array to track the visited nodes in the path from source to destination.

X		X	X		X	X	
1	2	3	4	5	6	7	
∞	∞	∞	∞	∞	∞	∞	(MIN(0,15))
5	22	17	15	26	6	0	(MIN(0,10))
X				X	X	X	X
8	9	10	11	12	13	14	15
∞	∞	∞	∞	∞	∞	∞	∞
20	20			24	18	10	12

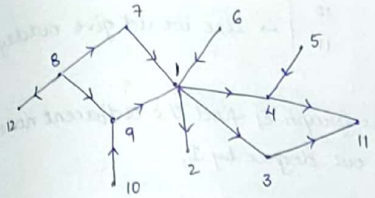
MIN(5,24)
MIN(20,20) MIN(20,24) 18 17 (MIN(18,17))
MIN(17,19) MIN(10,25)
MIN(12,22)
MIN(12,30)

Dijkstra's Algorithm

```
int Dijkstra( AL<AL<Pair<int,int>>> G, int S, int D){
    int d[G.size()] = {∞}
    d[s] = 0;
    MinHeap<Pair<int,int>> mh = new MinHeap<>();
    mh.push(0,S);
    while(!mh.empty()){
        p = mh.deleteMin();   // mh.poll();
        dist = p.first;
        u = p.second;          → if(u==D) return dist;
if(dist==d[u]){ ←
 → ②   for(Pair q : G[u]){
            v = q.first
            w = q.second
            newDistance = dist + w;
                          ↓ d[u]
            if(newDistance < d[v]){
                d[v] = newDistance;
                mh.push(newDistance, v);
            }
        } } → if-close ②
    }
    return -1;
}
```

Complexity : O(1

Problem Given a directed graph, denoting libraries & their dependencies, print an order such that all of them can be executed



denotes Library 1 is dependent on Library 2

12	13				
8	12	4	11	3	11
10	9	7	1	5	4
8	9	1	4	9	1
8	7	6	1		
1	2	1	3		

Based on the adjacency list, one of the order to print the libraries is,

Ans: 12, 2, 11, 3, 4, 5, 1, 6, 7, 9, 10, 8,

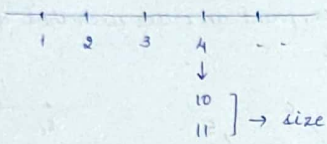
Approach Calculate the degree (out-degree) of each node

Consider a node with 0 (out-degree) as the starting library to be executed

Then decrement the out-degrees of its adjacent nodes

How? How do find the adjacent nodes of that current node?

out [N+1] =



→ size would give outdegree.

Contd..

Iterate the entire graph & find the adjacent nodes & decrement their out-degree by 1.

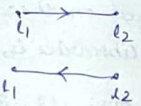
Complexity : $O(|V|(|V|+|E|))$

- $|V|$: Iterate all the nodes to print them.
- $|V|+|E|$: To iterate & find adjacent edges

way2 We can optimize this, by storing the adjacent nodes. While reading the graph, reverse the direction & store, so that the adjacent list would contain

Instead of `G[u].push(v);`

Use `G[v].push(u);`



`out[u]++;`

Once the directions are reversed & graph computed, use in-degree array of each node.

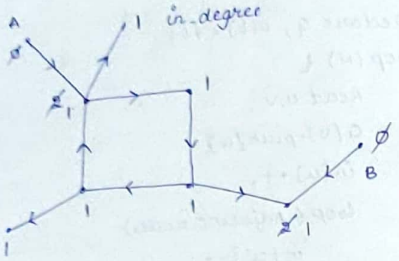
```
G[v].push(u);
in[u]++;
```

```
Read N, M
Declare G, in[] = {0}
loop (M) {
    Read u,v;
    G[v].push[u]
    in[u]++;
}   loop (Adjacent nodes)
        in[u]--;
Iterate in_degree list
for (int i=1 to N)
    if (in(i) == 0)
        L.add(i);
while( !L.isEmpty()) {
    u = L.pop();
    for (int v : G[u]) {  int in[v]--;
        if (in[v] == 0) {
            L.add(v);
        }
    }
}
```

Problem Detect a cycle in a given graph (directed graph)

NOTE: The above/previous question is nothing but

★ TOPOLOGICAL SORT



If there are no nodes left with in-degree = 0 then it indicate a cycle in the graph.

After visiting the nodes A & B, there are no nodes with in-degree = 0.

Complexity : $O(|V|+|E|)$, $O(|V|)$

★ Topological Sort :

→ ① Kahn's Algorithm
[which uses in/out-degree]

→ ② DFS
[with slight modification]

Problem Given an undirected graph (weighted), convert it into a MST (Minimum spanning Tree)

→ Prim's
→ Kruskal's

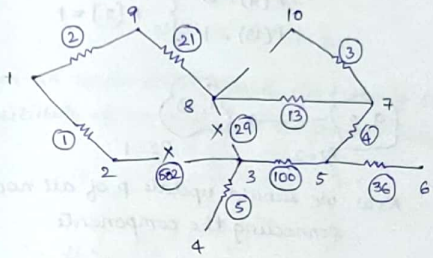
★ Kruskal's algorithm

→ sort the edges based on weight

$O(|E|\log_2|E| + |E|)$

→ check for cycles in an undirected graph
To be specific check for each component

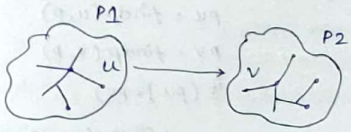
// $O(|E|\log_2|E| + |E|(|V|+|E|))$
$|V|$



→ Donot connect the node which can form a cycle in that component.

Drawback : we need to traverse the component to check for a cycle.

Better way :

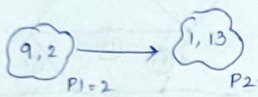


If (parent (v) != parent (u))
then join (u, v)

Node :	1	2	3	4	5	6	7	8	9	10	11
Parent :	1	~~2~~ 1	~~3~~ 1	~~4~~ ~~5~~ 1	~~5~~ 1	~~6~~ ~~5~~ 1	~~7~~ ~~5~~ 1	~~8~~ ~~5~~ 1	~~9~~ ~~2~~ 1	~~10~~ ~~8~~ 1	~~11~~ ~~10~~ ~~8~~ 1

Node :	12	13
Parent :	~~12~~ ~~10~~ ~~8~~ 1	~~13~~ 1

P(9) = 2
P(13) = 1
} P[2] = 1



Now, we should update P of all nodes after connecting the components

9, 2, 1, 13 → P = 1.

```
int findParent ( int u) { }  while ( P[u] != u) {
                                    u = P[u] }
    sortedEdges                 return u; }
    for ( u,v in E) {
        pu = findp(u,P)
        pv = findp(v,P)
        if (pu != pv)
            P[max(pu,pv)] = min(pu,pv);
    }
}
```

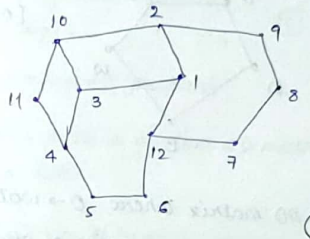
To avoid hops,

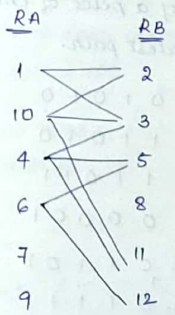
```
int findParent (int u, int [] p) {
        if (P[u] == u)
            return u;
        return ( P[u] = find(P[u], P));
}
```

This is called,

→ DISJOINT DS
→ UNION-FIND DS

Problem Given an undirected graph, find the if there is a possibility of no adjacent nodes fall into same religion





NOTE

These are called
* bipartite graphs
→ which can be divided into two partitions
→ graph colouring

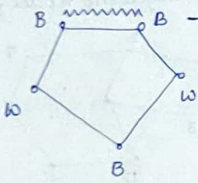
Problem Given a graph, check if it has odd length cycle.

For a cycle, $|E| = |V|$

NOTE If a graph is 2-colourable, then it has even length cycles, else it has odd length cycles

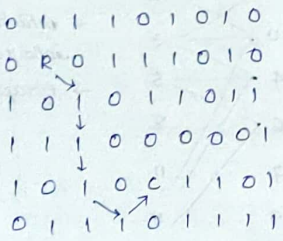
GRAPH COLOURING
- 2 colourable
- 3 colourable
- ↓

Implementation → BFS / DFS



→ Rule breaks [ODD LENGTH]

Problem Given a 2D matrix where 0 → water & 1 → land. Placed a lazy rat & a peice of cheese, find the lexicographically shortest path.



steps = 5

given two indecies

$(i_1, j_1) \rightarrow (i_2, j_2)$

if ($i_1 = i_2$ and $j_1 < j_2$)

Pick (i_1, j_1)

i.e → $(i,j) < (k,l)$ iff $(i<k)$ or $(i == k$ && $j<l)$

i.e → Find shortest path in an undirected & unweighted graph using BFS (level-order traversal)

Construct N8-connectivity matrix

Order of traversal is as shown in the Fig.1

for lexicographical traversal

dx[], dy[]

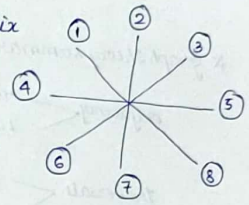
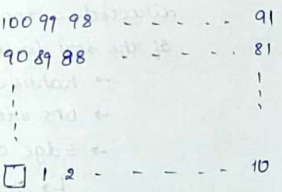


Fig1.

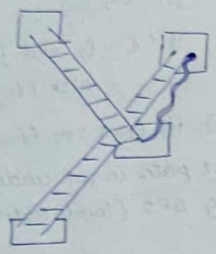
★ Internal Contest-3 problem

- store the path in another 2D matrix & print it in reverse order.

Problem Given a snakes & ladders game board & start & ending pts of each ladder & snake. Find the min no of steps to reach 100.



Point — Greedy will not work in this problem



It is a directed graph
unweighted graph
→ use BFS

* Graph Theory summary:-

Adjacency < Matrix, List

Traversals < BFS, DFS

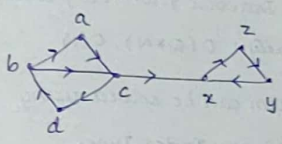
(SSSP) shortest Path from single source < from single source node (BFS) unweighted graphs; Dijkstra's algorithm weighted graphs

cycle detection < undirected → check connected components = 1 & |E| = |V|-1 or simply use Disjoint sets
directed → topological sort
If the sort fails, the graph has cycles
→ Kahn's algorithm
→ DFS with modification
→ Edge classification
→ Forward Edge
→ Tree Edge
→ Cross Edge
→ ✓ Back Edge

Minimum Spanning Tree < Prim's, Kruskals

Graph colouring < Bi-partite graphs
NO odd length cycles exist

→ Strongly Connected Graphs



DFS < Start, Reversing, End

→ Max-Flow Algorithm [Most Important]

→ Vertex Cut

→ Floyd-Warshall [is important]

Date
11.08.2019

CLASS - 26

Problem Given an array of N elements and below conditions,

$ar_{N=12}$ 5 10 3 2 7 4 18 9 8 1

Q | i, j $\quad 0 \le i \le j < N$ $\quad \max_{k=i}^{j} (ar(k))$

Way1 Brute force : Traverse from i to j and compute ~~sum~~ max

Complexity : $O(Q*N), O(1)$

The above problem can be solved using

→ Binary Index Trees

→ Segment Trees

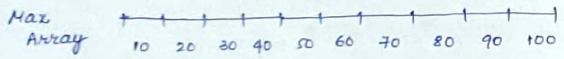
→ Sqrt Decomposition

Way2

★ Sqrt Decomposition

Let's say $N = 100$

$\sqrt{N} = 10$, divide the elements into 10 parts



$i = 57$ $j = 85$

Consider the ~~sum~~ max[6] and ~~sum~~ max[7]

{60-69} {70-79}

Iterate from $i = 57$ to 59 and $i = 80$ to 85

Total iterations = 3 + 1 + 1 + 6 = 11

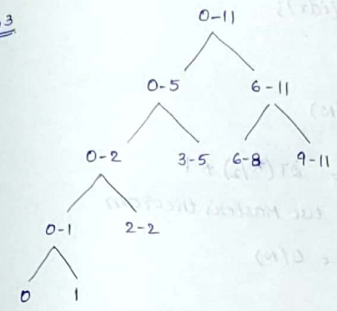
Complexity : $\boxed{O(N + Q\sqrt{N}), O(\sqrt{N})}$

★ Segment Trees

Consider the array, $N = 12$

Construct a tree, such that each nodes holds the max sum of a particular set of (i,j) like,

Way3




```
MAIN:   loop(T)
            Read(N)
            ar[N]
            Read ar
Nlog2N ←    st[?]
↓           build(ar, st, 1, 0, N-1);   (low, high)
10N                  ↑
20N          Index of root
```

```
int build (int[] ar, int[] st, int idx, int low, int high){
    if (low == high){
        st[idx] = ar[low];
        return st[idx];
    }
```

```
int mid = (low + high)/2
st[idx] = max(build(ar, st, 2*idx, low, mid),
              build(ar, st, 2*idx+1, mid+1, high));
return st[idx];
}
```

Complexity : O(N)

Time complexity = $2T(N/2) + 1$
for building segment tree — Use Master's theorem
$= O(N)$

Now, compute max for a given range of (i,j)

Base conditions:

→ if it is completely outside
→ if it is completely inside

```
int query(int i, int j, int st[], int idx, int low,
          int high) {
    if (i > high || j < low)
        return INT_MIN;
    if (~~i < low i >= low && j <= high~~)
        ~~return st[idx];~~
    if (i <= low && j >= high)
        return st[idx];
    mid = (low + high)/2;
    return (MAX(query(i, j, st, 2*idx, low, mid),
                query(i, j, st, 2*idx+1, mid+1, high)));
}
```

Complexity to find max for a given query
$= 2T(N/2) + 1$
$= O(N)$
where N = Height of the tree
$= O(H)$
$= O(\log_2 N)$

* Total Complexity : $O(N + Q\log_2 N), O(N\log_2 N)$
  - $O(N)$: Construct segment tree.
  - $Q\log_2 N$: Q * height of the tree
  - $N\log_2 N$: space for segment tree

Let's say there are two types queries given

Q1 : asks for MIN in the range (i,j)
Q2 : update value at a index k in ar[N]

Once we update the value of an element, the whole segment tree has to be updated to reflect Q2.

Q1 : i, j

Q2 : i x : ar(i) = x.

```
int
void int query2(int i, int[] st, int x, int ar, int low,
                                                int high){
    // ar[i] = x;
    if (low == high)
        return st[idx];
    if (i == low) {
        st[idx] = x;
        return st[idx];
    }
    mid = (low + high)/2;

}
```

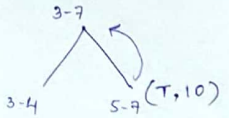
★ Lazy propagation

Let's say there is a third query

Q3: i, j $\forall_{k=i}^{j} ar(k) = x$

We can reuse the logic query 1

Ex: 5-7 ⇒ update value to 10



value to be propagated

& Bool value flag

We update the value only when query 1 visits those nodes. query 3 = $\log_2 N$

Sum instead of $N \log_2 N$