1) * Intersection of 2 LinkedList:

HashSet hs

hs.add(head1) head1.next. until head1 null

traverse 2nd LinkedList

if (!hs.add(head2))

return head2.

head2 = head2.next.

return null.

2) * Detect cycle in LL:

fast = head.next slow = head.

while (fast != null && fast.next != null)

if (fast == slow) true;

slow = slow.next.

fast = fast.next.next

return false.

3) * Reverse Node in k-group

initialize cur, prev, next with dummy

Add dummy node at start

count total node excluding dummy

while (count >= k)

cur = prev.next

nex = cur.next

for (i = 1; to k)

{ cur.next = ~~prev~~.next

nex.next = prev.next;

prev.nex = next;

nex = curr.next

}

prev = curr;

count -= k.

return dummy.next;

4) Check if singly linked list is palindrome.

enter all elements into stack
Iterate over LL and pop an element
If same continue
else return false

TC: O(n)
SC: O(n)

2nd: find mid of LL
reverse the second half
and the compare the first half & 2nd half

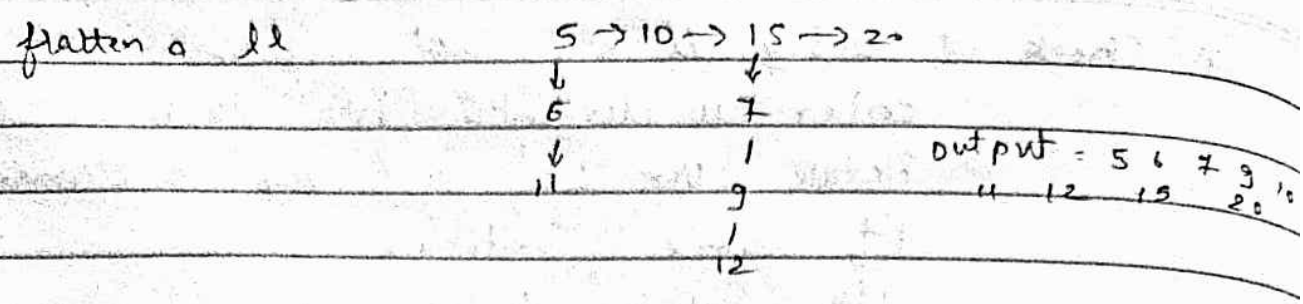
TC: O(n) SC: O(1)

5) Detect cycle in LL and return the start node of cycle.

```
fast = head, slow = head
while (fast != null && fast.next != null)
{   slow = slow.next;
    fast = fast.next.next;
    if (slow == fast)
    {   fast = head;
        while (fast != slow) {
            fast = fast.next;
            slow = slow.next;
        }
        return slow;
    }
}
return null
```

TC: O(n) SC = O(1)

* flatten a ll




```
flatten (Node root)
    if (root = null or root.next = null) return root;
    root.next = flatten(root.next)
    root = merge (root, root.next)
    return root.

merge (Node a, Node b)
    if (a == null) return b;
    if (b == null) return a;
    Node result;
    if (a.data < b.data) {
        result = a;
        result.bottom = merge (a.bottom, b)
    }
    else {
        result = b;
        result.bottom = merge (a, b.bottom);
    }
    return result;
```

TC: O(n) SC: O(1)

* Rotate a ll by k times to right:

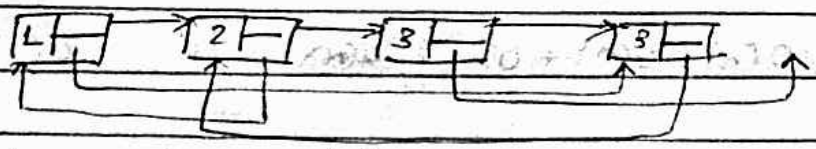
```
rotate (ListNode head)
{   ptr = null;
    temp = head;
    while (temp.next != null) {
        ptr = temp; temp = temp.next;
    }
```

// capture 2nd last Node

```
ptr.next = null;
temp.next = head;
head = temp;
return temp;
}
```

for handling large k values
count node in LL
and perform k = k % count;

8 * Copy list with ~~deep~~ random pointer:

① 

add nodes between.


```
iter = head, front = head;
while (itr != null)
{ front = iter.next;
  Node c = new Node(itr.val);
  iter.next = c;
  c.next = front;
itr = ~~front~~ front;
}
```

② point the random values.

```
iter = head
while (iter != null)
if (itr.random != null)
        itr.next.random = iter.random.next;
iter = itr.next.next;
}
```

(3) make original list intact and changes the new list;

```
Node d = new Node(0)
itr = head, start = d;
while (itr != null) {
  front = itr.next.next
  d.next = itr.next;
  itr.next = front;
  d = d.next;
  itr = front;
return start.next;
```

TC: O(n) + O(n) + O(n) = O(n) SC = O(1)

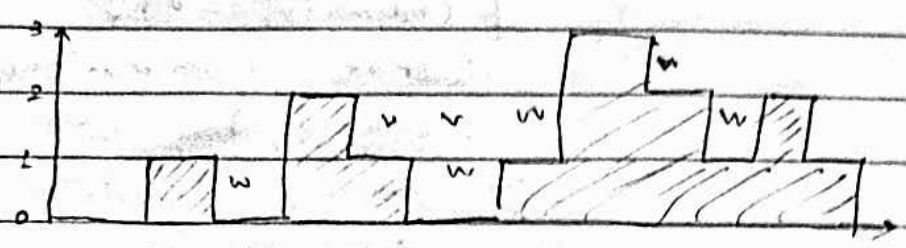
9) 3Sum: return triplets such that they are of different ind

```
int sum == 0,
  Arrays.sort(nums)
  ArrayList<ArrayList<Integer>> res = new ArrayList<>()
for (int i=0; i<n-2) {
  if (i==0 || (i>0 && num[i] != num[i-1])) {
int lo = i+1, hi = num.length()-1, sum = 0 - nums[i]
while (lo < hi) {
if (num[lo]+num[hi] == sum) {
ArrayList<Integer> l = new ArrayList();
  l.add(num[i]), l.add(num[lo]) l.add(num[hi])
  res.add(l);
while (lo<hi && num[lo] == num[lo+1]) lo++;
while (lo<hi && num[hi] == num[hi-1]) hi--;
} else if (num[lo]+num[hi] < sum) lo++;
else hi--;
return res
```

TC: O(N²) O(N)

10) Trap Rain water Compute how much water it can trap:

Input ip: [0 1 0 2 1 0 1 5 2 1 2 1]




```
int res = 0
int n = arr.length
int left = 0, right = n-1
int lmax = 0, rmax = 0.
while (left <= right) {
    if (arr[left] <= arr[right]) {
        if (leftmax < arr[left]) leftmax = arr[left]
        else res += lmax - arr[left] }
    left++; }
    else {
        if(rmax < arr[right]) rmax = arr[right]
        else res += rmax - arr[right]
    right--;}
}
return res;          TC : O(N)    SC : O(1)
```

11) Remove duplicate from sorted array:

2 pointer:

```
i = 0
for (j = 1 ; j < arr.length ; j++)
{   if (arr[i] != arr[j])
    { i++;   arr[i] = arr[j];
    }
}
return i+1;          TC : O(N)    SC : O(1)
```

12) Max consecutive ones input binary val.

```
int ones = 0, max = 0;
for (int i=0; i< nums.length; i++)
 { if (nums[i] == 0) {
        max = Math.max(ones, max);
        ones = 0; }
    else ones++;
 } return Math.max (max, ones);
```

TC : O(N) SC : O(1)

Greedy Algo:

13) N meeting in one room:

```
class Pair {
    int s, e;
    Pair (int s, int e) {
    this.s = s; this.e = e;
    }
}

maxMeeting (int start[], int end[], int n) {
int res = 0;
Pair arr[] = new Pair [n];

Arrays.sort (arr, new Comparator<Pair> () {
    public int compare (Pair p1, Pair p2) {
        return p1.e - p2.e;
    }
});
int last = 0;
for(int i = 0; i<n; i++){
```

```
if (arr[i].s > last) {
    res++;  last = arr[i].e;
  }
} return res;
}
```

TC: O(n) for inserting in ds.
O(n log n) for sorting
O(n) for checking
∴ O(n log n) SC: O(n)

11) Min platforms: given arrival & depart time of trains find min platforms required.

```
Arrays.sort(arr); Arrays.sort(dep);
platform=1, result=1
i=1 j=0
while (i<n && j<n) {
  if (arr[i]<=dep[j]) {
    platform++; i++}
  else if (arr[i] > dep[j]){
    platform--; j++}
  result = Math.max(result, platform);
}
return result
```

TC: O(n log n) SC: O(1)

15 Job scheduling problem:
given the jobid, deadline, profit we need to maximize the profit.

Sort jobs in decreasing order of profit
Arrays.sort(arr, (a, b) → b.profit - a.profit);

find max deadline

result [] = new int [max_deadline];

initialize the array element with -1

countjob = 0, jobprofit = 0

```
for (int i=0; i<n; i++) {
    for (j = arr[i].deadline; j>0; j--) {
        if (result[j] == -1) {
            result[j] = i;
            profit += arr[i].profit;
            countj++;
            break;
        }
    }
}
```

TC : O(n log n) sorting + O(N*M) for calculation

SC : O(M) for array result;

16) Fractional Knapsack:

class Item contains value and weight.

```
Arrays.sort(arr, new Comparator<Item>() {
    public int compare(Item a, Item b) {
        double r1 = (double) a.value / (double) a.weight;
        double r2 = " b " " " b.";
        if (r1 < r2) return 1;
        else if (r2 < r1) return -1;
        else return 0;
    }
});
int finalw = 0;
double profit = 0;
```

```
for (int i=0 ; i<n ; i++) {
    if ( finalW + arr[i].weight <= W) {
        finalW += arr[i].weight;
        profit += arr[i].value;
    }
    else {
        int remain = W - finalW;
        profit += ((double) arr[i].value / (double)
                   arr[i].weight) * remain;
        break;
    }
return profit
```

TC : $O(n \log n) + O(n)$ ∴ $O(n \log n + 1)$

SC : $O(1)$

17) find minimum number of coins :

greedy approach.

```
for(int i=n-1; i>=0 ; i--)
{
    while ( coins[i] <= V)
        { V -= coins[i];
          min++;
        }
}
return min;
```

TC : $O(V)$ SC : $O(1)$

DP approach :

```
int table[] = new int [v+1];
    table[0] = 0
for (int i = 1; i <= v; i++)
        table[i] = Integer.MAX_VALUE;
    }
for (int i = 1; i <= v; i++)
{
    for (int j = 0; j < m; j++)
        {
            if (coins[j] <= i)
            {
                sub = table[i - coins[j]];
                if (sub != Integer.MAX_VALUE
                    && sub+1 < table[i]) {
                    table[i] = sub+1; }
            }
        }
}
if (table[v] == Integer.MAX_VALUE) return -1;
return table[v];
```

TC: O(vm) SC: O(v)

- Recursion

18) Subset sum

```
eg  I/p = {2,3}
    out = {0, 2, 3, 5}

sol:
    fun(int ind, Array arr, Array ans, sum) {
        if (ind == arr.size()) {
            ans.add(sum) return;
        }
```

```
fun(ind+1, arr, ans, sum+arr.get(i+1))
fun(ind+1, arr, ans, sum);

}
```

TC: $O(2^n)$

SC: $O(2^n)$

19. Subset 2: should not include duplicate in ans.

```
void function(int ind, int nums[], ArrayList<Integer> sub,
        ArrayList<List<Integer>> ans){
    ans.add(new ArrayList<>(sub));
    for(int i=ind; i<nums.length; i++){
        if(i!=ind && nums[i-1]==nums[i])
            continue;
        sub.add(nums[i])
        function(ind+1, nums, sub, ans);
        sub.remove(sub.size()-1);
    }
}
```

// avoid duplicate

TC: $O(2^n)$ for generating subsets $O(k)$ for inserting $\therefore O(2^n * k)$

SC: $O(2^n * k)$

* Bubble sort:

```
for(i=0; i<n; i++)          → pass
{   for(j=0; j<n-i-1; j++)
    {   if(arr[j] > arr[j+1])
            swap
    }
}
```

TC: $O(n^2)$ SC: $O(1)$

* Improved B sort

```
for (int i = 0; i < n; i++)
{ for (j = 0; j < n - i - 1; j++)
  { if (arr[j] > arr[j+1])
    { swap
      flag = true;
    }
  } if (flag != true) break;
}
```

* Insertion sort :

```
for (i = 1; i < n; i++)
{ temp = a[i];
  j = i - 1;
  while (j >= 0 && a[j] > temp)
  { a[j+1] = a[j]
    j--;
  }
  a[j+1] = temp;
}
```

TC : WC : $O(n^2)$ BC : $O(n)$

SC : $O(1)$

* Selection sort :

```
for (i = 0; i < n-1; i++)
{ min = i;
  for (j = i+1; j < n; j++)
  { if (a[j] < a[min])
    { min = j;
    }
  }
```

if (min != i) { swap(a[i], a[min]); }

}

TC : $O(n^2)$ SC : $O(1)$

* Quick sort : decide pivot and elements on left are smaller or. elements or right are greater.

WC : $O(n^2)$

BC : $O(n \log n)$

```
int partition (lb, ub, arr[]) {
    pivot = arr[lb];
    start = lb, end = up
    while (start < end) {
     while (arr[start] <= pivot) start++;
     while (arr[end] * pivot) end--;
     if (start < end) swap (arr, start, end);
    }
    swap (arr, lb, end);    return end;
}
quicksort (lb, ub, arr[]) {
   if (lb < ub)
    { loc = partition (lb, ub, arr);
      quicksort (lb, loc-1, arr);
      quicksort ( ~~ub~~ loc+1, ub, arr);
    }
}
```

*

* Merge Sort :

7 2 1 5 6 4

left = 7
l = 2 ~~2~~ 5
r = 4

~~4~~ 2 1 5 6 7
↑ ↑
1 2 [4] 5 6 7

p = 4
l - ~~4~~ 3
r - ~~4~~ 2

3rd smallest
ans = 4

[1 2
[2, 1, 3]
[3, 2, 1]

Page No.:

~~4~~ 2 1 5

5 2 1 7 (0, 2) (4, 3)

1 2 5 7

```
merge (arr, l, m, r)
n1 = m - l + 1
n2 = r - m
arr1[n1] arr2[n2]
i=0; i<n1.
    arr1[i] = arr[l+i];
j=0 j<n2
    arr2[j] = arr[m+1+j]
i=1  j=0      k = ~~2~~ l
while (i<n1 && j<n2)
    if (arr1[i] <= arr2[j])
        arr[k] = arr1[i]  i++
    else  arr[k] = arr2[j]  j++
    k++
}
while (i<n1)  arr[k] = arr1[i]  k++  i++
while (j<n2)  arr[k] = arr2[j]  k++  j++
~~sort~~ ( ) {
if (l<r) {
    mid = (l+r)/2
    sort (arr, l, m)
    sort (arr, m+1, r)
    merge (arr, l, m, r);
```

20) Combination Sum 1 : input = [2, 3, 6, 7] tar = 7
out = [[2 2 3] [7]]

```
helper(ind, arr[], target, sub, ans)
{  if(ind == arr.length)
     { if(target == 0) {
         ans.add(new ArrayList<>(sub));
       } return;
     }
  if(arr[ind] <= target)
  {  sub.add(arr[ind]);
     helper(ind, arr, target - arr[ind], sub, ans)
     sub.remove(sub.size()-1)
  }
  helper(ind+1, arr, target, sub, ans);
}
```

TC : $O(2^{t*k})$ t = target k = avg length of ar
SC : $O(k*x)$ x = no. of combination

21 Combination sum 2 : Unique combinations :
in = [2, 5, 2, 1, 2] tar = 5
out = [[5], [1, 2, 2]]

```
Arrays.sort(arr);
Helper(0, tar, arr, sublist, ans);
return ans;

helper(int ind, tar, arr[], sub, ans){
  if(tar == 0) {
    ans.add(sub); return;
  }
```

```
for(int i=ind; i<arr.length; i++)
  { if (i>ind && arr[i]==arr[i-1]) continue;
    if (arr[i]>tar) break;
    sub.add(arr[i]);
    helper(i+1, tar-arr[i], arr, sub, ans);
    sub.remove(sub.size()-1);
  }
```

TC: $O(2^n * k)$ SC: $O(k*x)$

22) Palindrom Partition : in) = aab
out = [[a, a, b] [aa, b]]

```
helper(int ind, String s, sub, ans)
  { if (ind == s.length())
      { ans.add(new ArrayList<>(sub));
        return;
      }
    for(i=ind; i<s.length(); i++)
    {
      if(isPalindrome(s, ind, i))
      { sub.add(s.substring(ind, i+1));
        helper(ind+1, s, sub, ans);
        sub.remove(sub.size()-1);
      }
```

```
isPalindrome(String s, int start, end) {
  while (start<=end)
  { if (charAt(start++) != s.charAt(end--)) return false
  }
  return true; }
}
```

TC: $O(2^n * k * (n/2))$ 2^n for substring, k = avg length, n/2 for palindrome check

SC: $O(k * x)$

2) Permutation sequence:

In n=3 k=3

out = 3rd permutation in [123, ... 321]

= 213

// calculate total permutation for n-1

```
fact = 1
for (int i = 1; i < n; i++)
{ fact = fact * i;
  num.add(i);
{ num.add(i);
// due to 0 indexed.
k = k-1
String ans = "";
while (true) {
  ans += "" + num.get(k/fact);
  num.remove(k/fact);
  if (num.size() == 0) { break; }
  k = k % fact;
  fact = fact / num.size();
{ return ans.
```

TC: $O(N^2)$. SC: $O(N)$;

2a) Print all permutation of String/array:

```
Helper(nums, ans, 0)
Helper(nums[], ans, ind)
  if (ind == nums.length) {
    ArrayList a = new ArrayList
```

```
for (int i=0; nums.length; i++) {
    a.add(nums[i]);
} ans.add(a);
}

for (i=ind; i<nums.length; i++)
    { swap(nums, i, ind);
      helper(nums, ans, ind+1)
      swap(nums, i, ind);
    }
}
```

TC: O(N! * N) SC: O(1);

25) N-Queens:

```
// check if Queen in left row, left upper dig &
    // left lower dig
boolean validate(board, row, col)
{ dr = row  dc = col
  while(row>=0 && col>=0)
    { if(board[row][col] == 'Q') return false;
      row--  col--
    }
  col = dc; row = dr;
  while(col>=0) {
    if(board[row][col] == 'Q') return false;
    col--
  }
row=dr  col=dc
while(row < board.length && col>=0) {
    if(board[row][col] == 'Q') return false;
    row++  col--;
return true; }
```

```
// for converting board[i] to string
List<String> construct (board){
  for (i=0 ; i< board.length; i++){
    String s = new String(board[i])
    ans.add (s);
  }
return ans;

// board a char 2 d array initialized by '.'
  dfs (0, board, ans)
dfs (int col, board, ans) {
  if (col == board.length) {
    ans.add (construct(board));
    return; }
  for (int i=0; i< board.length; i++) {
    if (validate (board, i, col)
    board [i] [col] = 'q';
    dfs (col+1, board, ans)
    board [i] [col] = '.';
  }
```

TC : $O(N! * N)$

SC : $O(N^2)$

26) Sudoku solver 9*9 :

```
isValid (board, row, col, char c) {
  // check rows, cols and 3*3 box for same number
  for (i=0; i<9; i++) {
    if (board [i] [col] == c) return false
    if (board [row] [i] == c) return false
    if (board [3*(row/3) + i/3] [3*(col/3)+i%3] == c)
                return false
  }
return true.
```

```
solve (board) {
for (i=0 ; i<9 ; i++)
  { for (j=0 ; j<9 ; j++) {
      if (board[i][j] == '.') {

        for (char c='1' ; c<='9' ; c++) {
          if (isValid(board, i, j, c))
            {
              board[i][j] = c
              if (solve (board)) return true;
              else {
              board[i][j] == '.';
              }
            {
          } return false
      {
    }
  }
return true;
```

TC : $O(9^{n^2})$

SC : $O(1)$

27) M-coloring : given graph[][] , number of color, write to find if graph can be colored using the colors provided.

```
int color[] = new int[v];
solve (graph, color, 0, m, n)
```

```
solve(graph[][], color[], node, m, n)
  if (node == n) return true
  for (i=1; i<=m; i++)
    { if (isSafe(graph, color, i, node)
      {
        color[node] = i;
        if (solve(graph, color, node+1, m, n)) retur
                                                  true
        color[node] = 0;
      }
  return false;

isSafe(graph, color, col, node)
  { for (i=0; i<graph.length; i++) {
      if (graph[node][i] && color[i]==col)
          return false;
    return true.
  }
}
```

TC: $O(N^m)$ SC: $O(N)$

28) Rat in Maze: find all possible solution to (0-1)(n-1)

ans: RDDVR RDRVRD

// dir = DLRU depending on this make di and dj

di[] = {1, 0, 0, -1} (D, U)

dj[] = {0, -1, 1, 0} (L, R)

visit[][] = boolean[n][n]

if(m[0][0]==1) helper(0, 0, m, n, ans, "", visited, di, dj)

```
helper(int i, int j, m[][], n, ans, move, visited[][], di[],
                dj[]) {
    if (i == n-1 && j == n-1) ans.add(move) return.
    String dir = "DLRU";
    for (int ind=0; ind<4; ind++)
    { nexti = i+di[ind]   nextj = j+dj[ind]
    if (nexti>=0 && nextj >=0 && nexti<n && nextj<n
        && visited[nexti][nextj] == false && m[nexti][nextj]==1)
    { visited[i][j] = 1;
    helper(nexti, nextj, m, n, ans, move + dir.charAt(ind),
                visited, di, dj)
        visited[i][j] = false
    }
}
```

TC: $O(4^{(m*n)})$

SC: $O(m*n)$

29) Word break: input: god is now no where here
godisnowherenowhere.

o/p: god is no where no where
god is no where now here.
god is now here no where
god is now here now here.

```
// create a set and add dic element to it for
// fast retrival
    helper(0, s, hs, s.length());

helper(int ind, s, hs, size) {
    if (ind == size) { Arraylist temp = new Arraylist
    temp.add(""); return temp; }
```

```
ArrayList<String> sub,
ArrayList<String> complete
word = "";
for (i=ind; i<size; i++) {
  word += s.charAt(i);
  if (! hs.contains(word)) continue;

  sub = helper(i+1, s, hs, size)
  for (j=0; j<sub.size(); j++) {
      if (sub.get(j).length != 0) {
          string temp = word;
          temp += " ";
          temp += sub.get(j);
          sub.set(j, temp)
      }
      else { sub.set(j, word); }
  }
  for (j=0; j<sub.size(); j++) {
      complete.add(sub.get(j)); }
  return complete;
}
```

TC: $O(N^2 * (2^{N-1}))$

SC: $O(N^2 * (2^{N-1}))$

20) Newton Raphson method: nth root of m

n = 3 m = 27 ans 5.000000

```
double error = 1e-7, diff = 1e18, xv = 2;
while (diff > error) {
  xv1 = (Math.pow(xv, n) * (n-1) + m) / (Math.pow(xv, n-1) * n);
  diff = Math.abs(xv - xv1);
```

```
        xv = xv1;
    }
    return xv;
```

TC : O(log(m) * log(N));
SC : O(1).

31) Median Sorted ~~Array~~ Matrix : 1 8 5 ans 5
2 6 9
3 6 9

```
findd (ArrayList A) {
row = a.size(); col = a.get(0).size();
min = Integer.MAX_VALUE, max = Integer.MINVALUE
//find min and max in arraylist
for (i=0; i<row; i++) {
    min = Math.min(min, a.get(i).get(0));
    max = Math.max(max, a.get(i).get(col-1));
    }
req = (row*col)/2, ans = 0
while (min <= max)
{ mid = (min+max)/2;
  count = 0;
  for (i=0; i<row; i++ {
      count += binarysearch(a.get(i), mid);
      }
  if (count < req+1) { min = mid+1;
  //else { ans = mid;
      max = mid-1;
  }
}
return ans;
```

```
binary search (ArrayList<Integer> a, int val)
{
    if (a.get(0) > val) return 0;
    low = 0, high = a.size()-1, mid = 0;
    while (low <= high) {
    mid = (low + high) / 2;
    if (a.get(mid) > val)  h = mid-1
    else      low = mid+1
    }
    return low;
}
```

TC:

SC:

32) Single element in sorted array: O(log n) every element = for except 1 element is = for

O(n) solution is

Xor all elements ans is odd element

BS:

```
low = 0    high = n-2
while (low <= high)
mid = (low + high)/2
if (mid%2 == 0) {
    if (nums[mid] != nums[mid+1]) {
        high = mid-1; }
    else { low = mid+1; }
}
else {
    if (nums[mid] == nums[mid+1]) high = mid-1
    else  low = mid+1;
}
} return nums[low]
```

TC: O(log n) SC = O(1)

23) Search in Rotated Sorted array:

```
start = 0, end = nums.length-1;
while (start <= end) {
    mid = start + (end-start)/2;
    if (nums[mid] == target) return mid;
else if (nums[mid] > nums[start]) {
    if (target < nums[mid] && target >= nums[start])
    { end = mid-1; }
    else start = mid+1;
else if (nums[mid] < nums[start]) {
    if (target < nums[start] && target > nums[mid]) start = mid.
    else end = mid-1;
    }
    else { start++; }
}
return -1;
```

TC: O(log N) SC: O(1).

3.11) Median of 2 sorted arrays : (No extra space)

```
median(a[], b[])
{ // make the smallest array as a.
    if (a.length > b.length) median(b, a);
  n = a.length , m = b.length.
  low = 0   high = n
  while (low <= high)
  mid = (low + high)/2
  part = (m+n+1)/2 - mid
  if (part > m) {
    low = mid+1.
    continue;
  }   leftmax = MIN_VALUE   rightmin = MAX_VALUE
  if (mid > 0) leftmax = Math.max(leftmax, a[mid-1]);
  if (part > 0) rightmin = Math.max(leftmax, b[part-1]);
  if (mid < n) rightmin = Math.min(rightmin, a[mid])

  if [part < m] rightmin = Math.min(rightmin, b[part]);

  if (leftmax <= rightmin) {
((min)&1)==1)   if ((m+n)%2 == 1) return leftmax;
    else return (leftmax + rightmin)/2.0;
  }
  if (a[mid] < leftmax) low = mid+1;
  else high = mid-1;
}
return -1;
}
```

TC: O(log(M,n)) SC: O(1)

35) Kth element in sorted Array:

```
Kth (arr1, arr2, len1, len2, k)
{ if ((len1 + len2) < k || k < 1) return -1
  if (len1 > len2) Kth(arr2, arr1, len2, len1, k);
  if (k == 1) min(arr1[0], arr2[0]);
  if (len1 == 0) return arr2[k-1];

  int i = min(len1, k/2)
      j = min(len2, k/2)
  if (arr1[i-1] > arr2[j-1])
  { temp[] = Arrays.copyOfRange(arr2, j, len2)
    return Kth(arr1, temp, len1, len2-j; k-j)
  }
  temp[] = Arrays.copyOfRange(arr1, i, len1)
    return Kth(temp, arr2, len1-i, len2, k-i);
}
```

TC : log(k) SC : log(k)

36) Allocate Minimum number of Pages:

In = A[12, 84, 67, 90] B = 2

Out = 113 (maximum of min pages allotted)

```
books (A, B)
{ if (A.size < B) return -1;
  low = A.get(0), high = 0;
  for (int a : A) {
    high += a;
    low = min(low, a); }
  res = -1
  while (low <= high) {
  mid = (low + high)/2;
  if (isPossible(A, mid, B)) { res = mid; high = mid - 1; }
```

```
        else { low = mid+1;
        }
    return ans;
    }
    boolean isPossible(A, pages, students) {
    int cnt=0, sum=0;
    for (i=0; i<A.size; i++) {
        if (sum + A.get(i) > pages) {
            cnt++;
            sum = A.get(i);
            if (sum > pages) return false;
        } else
            { sum += A.get(i);
        }
    }
    if (cnt < students) return true;
    else return false;
```

27 Aggressive cows:

[1, 2, 8, 4, 9] Cows = 3.

assign cows such that min distance between them is maximum. Arrays

```
low=1  high=
    Arrays.sort(a);
    low = 1   high = a.get(a.size-1) - a.get(0).
    while (low <= high) {
    mid = (low + high)/2;
    if (isPossible(a, n, mid, cows))   ans = Math.max(ans, mid)
        low = mid+1
    else   high = mid-1
```

```
isPossible (a, n, minDiff, cows)
  int cnt = 1
  lastplaced = a.get(0).
  for (i=1; i<n; i++) {
    if (lastplaced - a.get(i) - lastplaced >= mindiff
      cnt++;
      lastplaced = a.get(i);
      if (cnt == cows) return true
  }
  return false.
```

TC: O(N log M)

3 8) Quick select Algorithm:

Kth smallest or kth largest element.

K largest →

```
l = 0   r = length - 1,   kth.
while (true) {
  idx = partion (arr, l, r);
  if (idx == k-1)
    Kth = arr[idx]
    break;
  if (idx < k-1)
    l = idx + 1
  else
    r = idx - 1
return idx.

partition (arr, left, right)
  pivot = arr[l]
```

```
l = left+1      r = right
while (arr[l] <= pivot && l <= r)
{ if (arr[l] < pivot && arr[r] > pivot) {
    temp = arr[l]
    arr[l] = arr[r]
    arr[r] = temp
    l++   r--
if (arr[l] >= pivot)  l++
if (arr[r] <= pivot)  r--
}
temp = arr[left]
arr[left] = arr[r]
arr[r] = temp
return r;
```

TC : O(n) SC : O(1)

39) Max/Min Heap implementation - GFG.

40 Maximum sum combination.

A = [2 3] B = [4 5] C = 2 ans = [8 7]

```
PriorityQueue<Integer> pq = new PriorityQueue();
Collections.sort(A, Collections.reverseOrder())
//sort both array in reverse order.
for (i=0 ; i<a.size; i++)
{ for (j=0; j<b.size ; j++)
    { sum = a.get(i) + b.get(j)
      if (pq.size < c)  pq.add(sum)
      else if (sum > pq.peek()) { pq.poll();
                                  pq.add(sum);
      } else { break; }
}
```

```
ArrayList<Integer> ans = new ArrayList(pq)
Collections.sort(ans, Collections.reverseOrder())
return ans;
```

41 find median from datastream:

```
addnum(num){
if(maxheap.isEmpty() || maxheap.peek >= num){
    maxheap.add(num)
else minheap.add(num);

if (maxheap.size() >= minheap.size()+1)
    minheap.add(maxheap.poll);
else if (maxheap.size()< minheap.size())
    maxheap.add(minheap.poll())
}

findMedian()
} if (maxheap.size == minheap.size) return
    (maxheap.peek() + minheap.peek)/2.0;
else return maxheap.peek();
```

42 Merge k sorted arrays

// create a min heap and add all elements to it

// remove one element and add to the ans until minheap is not empty.

TC: $O(N*K + K^2)$

// create a class with 3 elements

// first, second third.

class pqComp implements Comparator<Pair>{

```
int compare (Pair p1, Pair p2) {
    if (p1.first < p2.first) return -1
    else if (p1.first == p2.first) return 0
    else return 1
  }
}

PriorityQueue minH = new PriorityQueue (new PQcom());

for (i=0; i<k; i++) {
  minH.add ( ~~karrays.get(i).getLe~~ new Pair (Aarray.get(i).
                                                get(0), i, 0));
}
While (!minH.isEmpty()) {
  Pair curr = minH.poll(); or minH.remove()
  int i = curr.second
      j = curr.third;
  result.add(curr.first)
  if (j+1 < karrays.get(i).~~get~~ size()) {
    minH.add(new Pair(Aarrays.get(i).get(j+1), i, j+1));
} }
return result
```

TC : $O(k * N * \log(k))$

45) Top K frequent elements.

in: [1, 1, 1, 2, 2, 3] k=2

ans = [1, 2]

```
HashMap<Integer, Integer> map = new HashMap();
  for (i=0; i<a.length; i++) {
    map.put(a[i], map.getOrDefault(a[i], 0)+1);
  }
```

```
PriorityQueue pq = new PriorityQueue((a,b) -> map.get(a)
                                        map.get(b));
~~for(i=0; i<a.length; i++){~~
  for (int i : map.keyList()){
      pq.add(i);
      if (pq.size > k) pq.poll();
    }
int ans[] = new int[n]
    for(i=k-1; i>=0; i--)
        { ans[i] = pq.poll();
        }
    return ans;
}
```

Stack:

44) Nearest smaller element:

In - A = [4, 5, 2, 10, 8]

Ans = [-1, 4, -1, 2, 2]

```
ArrayList<Integer> result = new ArrayList();
Stack<Integer> st = new Stack();
for(i=0; i< A.size(); i++)
{ while (!st.isEmpty() && st.peek >= A.get(i)) st.p
  if (st.isEmpty()) ~~add~~ result.add(-1)
  else result.add(st.peek());
  st.push(A.get(i))
}
return result;
```

45) LRU implementation: (Doubly LL & HashMap)

```
class Node {
    Node next, prev;
    int key, val;
```

```
Node (int k, int v) {key = k  val = v; }
{
private Node head = new Node(0,0), tail = new Node(0,0)
    HashMap<Integer, Node> map;
    capacity
insert(Node n)
{ map.put(n.key, n);
    no.next = head.next
    n.next.prev = n
    head.next = n
    n.prev = head.
}
remove (Node n) {
    map.remove(n.key)
    n.prev.next = n.next;
    n.next.prev = n.prev }
get (key)
{ if (map.containsKey(key)) {
    Node n = map.get(key);
    remove(n)
    insert(n)
    return n.value;
    }
    else return -1;
}
put(key, val)
    if (map.containsKey(key)) remove(map.get(key));
    if (map.size == cap)  remove(tail.prev)
    insert(new Node(key, val));
}
```

46) Implement stack using array.

47) Implement ~~stack~~ Queue using array.

48) Implement stack using queue (2).
while adding element to queue
run a for loop from 0 to que.size-1
and add the elements by removing the
first element.

TC O(n) SC O(N)

49) Implement Queue using 2 stacks:

```
push (O(1))
stack in, out;
    push (x) {
      in.push(x);
      }
    pop() {
    while (!in.isEmpty)
        out.push(in.pop());
    return out.pop();
    }
```

TC : O(1) SC: O(2N)

50) Valid paranthesis: In {"()[]{}" or "[)" or "["

```
for (char c : s.toCharArray())
{ if (c == '(' || c == '{' || c == '[') st.push(c)
else {
if (st.isEmpty()) return false;
```

```
char ch = st.pop();
if ((ch == '(' && c == ')') || (ch == '{' && c == '}') ||
    (ch == '[' && c == ']')) continue;
else return false;
}
return st.isEmpty();
}
```

TC: O(n) SC: O(n)

5) Next greater element : (to the right)

in = [4, 6, 5] out = [6, 8, -1]

```
ans[n-1] = -1
st.push(arr[n-1])
for(i=n-2; i>=0; i--)
  while(!st.isEmpty() && st.peek() <= arr[i]) st.pop();
  if(st.isEmpty()) ans[i] = -1
  else ans[i] = st.peek()
  st.push(arr[i])
}
```

Same Queue Array is Circular:

```
A = [5, 10, 6, 2, 1, 2, 8, 7, 2, 9]
ans = [10, -1, 8, 8, 2, 8, 7, 9, 9, 0]
for(i = 2*N-1; i>=0; i--)
while(!q.isEmpty() && st.peek() <= arr[i%N]) st.pop();
if(i<n) { ans[i] = if(st.isEmpty()) ans[i] = st.peek();
          else ans[i] = -1;
        }
st.push(arr[i%N])
} return ans
```

TC: O(N) SC: O(N)

52) Sorting a Stack (recursion)

```
sort(st){
if(st.isEmpty) return;
top = st.pop();
sort(st); //removing the element and emptying the stack.
stackInsert(st, top)
}
stackInsert(st, curr){
if (st.isEmpty() || st.peek < curr)
    st.push(curr) return;
top = st.pop()
stackInsert(st, curr); //Sorting and putting min.
st.push(top);
}
```

TC: $O(N^2)$ SC: $O(N)$

53. LFU: DLL HashMap:

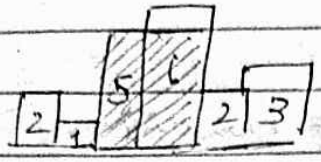
```
class DLLNode{
int key, value, frequency;
DLLNode prev, next;
DLLNode(K,V){
key=K;
value=V;
frequency = 1
prev = null;
next = null;
}
}
```

```
class DoublyLinkedList {
    int listSize;
    DLLNode head, tail;
    DoublyLinkedList() {
        head = new DLLNode(0,0);
        tail = new DLLNode(0,0);
        head.next = tail; tail.prev = head;
        listSize = 0;
    }
addNode(DLLNode n) {
    n.next = head.next;
    n.prev = head
    head.next = n.
    n.next.prev = n; listSize++;
}
removeNode(DLLNode d)
{   d.next.prev = d.prev
    d.prev.next = d.next
    listSize--;
}
// two class one is Node and other is DoublyLL.

// LFU class has 3 int variable capacity = cap, curSize = 0, minFr = 1
Map<Integer, DLLNode> cache = new HashMap();
Map<Integer, DoublyLinkedList> frequencyList = new HashMap();

updateList(DLLNode node) {
    DoublyLinkedList list = frequencyList.get(node.frequency);
    list.removeNode(node);
    if (node.frequency == minFrequency && list.listSize == 0)
        minFrequency++;
```

```
        node.frequency ++;
    DoublyLinkedList newL = frequencyList.getOrDefault(node.frequency,
                                new DoublyLinkedList());
    newL.addNode(node);
    frequencyList.put(node.frequency, newL);
}

get(int key) {
    if (!cache.containsKey(key)) return -1;
    DLLNode n = cache.get(key);
    updateList(n)
    return n.value;
}

put(key, val) {
    if (capacity == 0) return;
    if (cache.containsKey(key)) {
        DLLNode n = cache.get(key);
        n.value = value;
        updateList(n);
    }
    else {
        cursize++;
        if (cursize > capacity) {
            DoublyLinkedList list = frequencyList.get(minFrequency)
            cache.remove(list.tail.prev.key)
            list.removeNode(list.tail.prev);
            cursize--;
        }
        minFrequency = 1;
        DoublyLinkedList list = frequencyList.getOrDefault(minFrequency, new Doubly
        DLLNode n = new DLLNode(key, val)
        list.addNode(n)
        cache.put(key, n)
        frequencyList.put(1, list);
    }
```

54. Largest Rectangle in histogram:

= 10




```
Stack<Integer> st = new Stack<>()
int leftSmall[] = new int[heights.length]
int rightSmall[] = ——u——
for(i=0; i< heights.length; i++) {
    while(!st.isEmpty() && his[st.peek] >= heights[i]) st.pop();
    if (st.isEmpty()) leftSmall[i] = 0
    else leftSmall[i] = st.peek() + 1;
    st.push(i);
} // then Empty the stack:
for(i= heights.length-1; i>=0; i--) {
    while (!st.isEmpty() && his[st.peek] >= heights[i]) st.pop();
    if (st.isEmpty()) rightSmall[i] = n-1
    else rightSmall[i] = st.peek()-1;
    st.push(i);
}
// (rightSmall[i] - leftSmall[i] + 1) * heights[i]
for(i=0; i<n; i++) {
    max = Math.max(max, rightSmall[i] - leftSmall[i]+1) * heights[i];
}
return max;
```

TC = O(N) + O(N) + O(N) + O(N) + O(N)

SC = O(3N)

55) Sliding Window Maxi:

IO = [1 3 -1 -3 5 3 6 7] k = 3

OUT = [3, 3, 5, 5, 6, 7]

```
Deque<Integer> d = new ArrayDeque();
int ans[] = new int[nums.length - k + 1];
r=0;
for (i=0; i<n; i++)
  { while(!d.isEmpty() && d.peek() == i-k) d.poll

    while(!d.isEmpty() && nums[d.peekLast()] < nums[i
        d.pollLast();
    d.offer(i);
    if (i>=k-1) { ans[r++] = d.peek(); }
  }
return ans;
}
```

TC : O(N) SC : O(K)

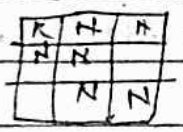
56) Min Stack:

TC: O(1) Stack<long> st, min;

```
push(val)
  { if (st.isEmpty) st.push(val) min = val;
    else {
      if (val < min) { st.push(2*val - min); min = val;
      else { st.push(val); }
    }
  }

pop(){
  if(st.isEmpty()) return;
  long val = st.pop();
```

```
if (val < min) min = 2*min - val;
{
top() {
long val = st.peek();
if (val < min) { return min.intValue(); }
else return st.peek().intValue();
}
getMin() {
return min.intValue();
}
TC : O(N)
```

57 Min time to rotten all oranges



ans 4

```
for(i=0; i<row) {
  for (j=0; i<col) {
    if (grid[i][j] != 0) total++;
    if (grid[i][j] == 2) q.offer(int[]{i,j});
  }
}
dy[] = {0,0,1,-1}  dx[] = {1,-1,0,0}
while (!q.isEmpty()) {
size = q.size();
cnt += size
while (size--, >0) {
int arr[] = q.poll();
for (i=0; i<4) {
x = dx[i] + arr[0]   y = dy[i] + arr[1]
if (x<0 || y<0 || x>=row || y>=col || grid[x][y] != 1)
    continue;
grid[x][y] = 2   q.offer(new int[]{x,y});
```

```
        }
    }
    if(q.size() != 0) min++;
}
return total == cnt ? min : -1;
```

TC O(n*n)*4 SC: O(n*N)

58) Stock Span :
↳ last days where stock was low than current day

In = [100, 80, 60, 70, 60, 75, 85] ans = [1, 1, 1, 2, 1, 4, 6]

```
stack<int[]> st;
day = 0
next(price)
if (day == 0) { st.push(new int[]{price, 0}); day++ return }
while (!st.isEmpty() && st.peek()[0] <= price) { st.p
if (st.isEmpty()) { ans = day+1; }
else { ans = day - st.peek()[1] }
st.push(new int[]{price, day});
day++;
return ans;
}
```

TC: O(N)

59) Maximum of Min for wiry window size :

in = [8 3 4 2 4] out = [4 3 3 2 2]

```
// create an ans array of length n assign Integer.MIN
// count the prev small for each element if st.empty
//      the assign -1
// count next small for each element if st.empty assign
```

```
for (i=0; i<n) {
  len = next[i] - prev[i] - 1
  ans[len-1] = Math.max(ans[len-1], arr[i]);
}
//some locations are not filled
for (i=n-2; i>=0)
  { ans[i] = Math.max(ans[i], ans[i+1]);
  }
TC: O(N)        SC: O(N)
```

80) Celebrity problem: find celebrity (a person known to all and who knows no one)

n = 3 own = 1

know(0,1) = T know(1,2) = F know(2,1) = T

know(0,2) = T know(1,0) = F know(2,0) = T.

```
//push all ids on to stack.
while (st.size > 1) {
  a = st.pop(), b = st.pop();
  if (knows(a,b)) st.push(b);
  else {st.push(a)
  }
celeb = st.pop();
boolean knows = false knownBy = true;
for(i=0; i<n; i++) {
  if( i!= celeb && knows(celeb, i)) knows = true break
  {
for(i=0; i<n) { if (i!=celeb && !knows(i,celeb)) knownBy = false
                                                  break; }
if (!knows && knownBy) return celeb;
return -1;
TC: O(N)      SC = O(1)
```

two pointers can be used

start id = 0 and end id = n-1

// while (start < end)

// if (knows (start, end) start++

else if end--;

rest is same. as previous

ulch = start;

TC = O(N) SC = O(1)

String:

61) Reverse words in string (remove unexpected spaces)

```
i = s.length()-1
while (i>=0){
    while (i>0 && s.charAt(i)==' ') i--;
    int j=i;
    if (j>=0){
        while (j>=0 && s.charAt(j)!=' ') j--;
        ans += s.substring(j+1, i+1);
        ans += " ";
    }
}
return ans.substring(0, ans.length()-1);
```

TC: O(N) SC = O(1)

62) Longest Palindromic Substring (if two with max return the one at start)

in = ababc out = aba.

```
expandDirection (str, left, right)
{ start = left  end = right
  while (start>=0 and end<str.length() && str.charAt(start--) == str.charAt(end++))
    start--, end++;
```

```
return (end - start + 1);
}

start = 0 , end = 0
for (i=0 to str.length(); )
{ len1 = expandDistance (str, i, i); // even length
  len2 = expandDistance (str, i, i+1); // odd length
  len = max (len1, len2);
  if (len > end - start + 1) {
    start = i - (len-1)/2;
    end = i + len/2;
  }
} return str.substring (start, end+1);
}
```

TC = $O(N^2)$ SC: $O(1)$

63 a) Roman to Intger : IN - VIX OUT = 9

```
for (i = s.length()-1 ; i >= 0 ) {
  ch = s.charAt(i);
  if (ch == 'I') ans += 1;
  else if (ch == 'V') ans += 5
  else if (ch == 'X') ans += 10
  else if (ch == 'L') ans += 50
  else if (ch == 'C') ans += 100
  else if (ch == 'D') ans += 500
  else if (ch == 'M') ans += 1000
```

~~else~~

```
if (i != s.length()-1) {
if (s.charAt(i+1) == 'V' || s.charAt(i+1) == 'X') ans -= 2;
    ch == 'I') ans -= 2;
else if ((s.charAt(i+1) == 'L' || s.charAt(i+1) == 'C') && ch == 'X') ans -= 20
```

```
else if ((s.charAt(i+1) == 'D' || s.charAt(i+1) == 'M') && s.charAt(i) ==
        ans -= 200;
    }
    return ans;
}
```

TC : O(N) SC : O(1)

b) Integer to roman :

S II

```
divi[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1}
String rom = {"M", "CM", "D", "CD", "C", "XC", "L", "XL",
              "X", "IX", "V", "IV", "I"}
for(i=0; i < divi.length; i++) {
    if (num/divi[i] != 0) {
        count = num/divi[i];
        while (count-- > 0) {
        ans += rom[i];
        }
        num = num % divi[i]; if (num == 0) break;
    }
return ans;
```

TC : O(N) SC : O(L)

64) a) AtoI

" -42 " ans = -42 int.

```
s.strip()  // lead and trailing white space remove
for(i=0; i < s.length();) {
ch = s.charAt(i);
if (!((ch >= '0' || ch <= '9') || ((ch == '+' || ch == '-') && i == 0))
    { s = s.substring(0, i)
    break;
}
}
```

```
if (s.length == 0 || (s.length == 1 && (s.charAt(0) == '+' ||
                                        s.charAt(0) == '-')))
    return 0;
double d = Double.parseDouble(s);
return (int) Math.floor(d);
}
```

TC : O(N) SC : O(1)

b) strStr:

```
in = h = hello  n = ll  find 1st occurrence of n in h

return haystack.indexOf(needle);

first = needle.charAt(0)
if (haystack.length() < needle.length) break;
ind = -1, count = 0  flag = false;
for (i=0 ; i<=haystack.length - needle.length ; )
  { if (first == haystack.charAt(i)){
      for(j=0 ; j < needle.length) {
        if (needle.charAt(j) == haystack.charAt(i+j)
            count++
        else break;
        if (count == needle.length) {ind=i; flag=true;
    }
    count=0;
    if(flag) break;
  }
{ return ind
```

TC : O(N) SC : O(1)

65) LCP (longest common prefix)

ip = ["flower", "flow", "fl"] ans = fl

```
if (strs == null || strs.length == 0) return "";
int low = 0, high = MAX_VALUE
for(string s : strs) { high = Math.min(high, st.length());

while (low <= high) {
mid = (low+high)/2;
if (common(mid, strs)) low = mid+1;
else high = mid-1;
}
return strs[0].substring(0, (low+high)/2);

common (string strs[], mid)
  string match = strs[0].substring(0, mid);
for(i=1; i< strs.length; i++) {
    if(! strs[i].startsWith(match)) return false
  }
return true;
}
```

TC : O(s log m) SC : O(1)

66) Repeated string Match:

a = abcd, b = cdabcdab:

how many time a must be repeated to get b as a substring of a.

```
String tem = a
```

```
rpt = b.length/a.length;
count = 1
for (i=0; i< rpt+2; i++) {
    if (a.contains(b)) return count;
    else {
        a += str;
        count++
    }
}
return -1;
}
```

* Rabin Karp (for finding patter in string).

```
m = str.length   n = pat.length
prime = 31       mod = 998244353;
ArrayList<Long> primePower, h, ~~reverse~~
// add 0 in primePower for m times
primePower.set(0, (long) 1)
for (i=1; i<m) {
  primePower.set(i, (primePower.get(i-1) * prime) % mod);
}
// add m+1 0 in h.
for (i=0; i<m) {
  h.set(i+1, (h.get(i) + (str.charAt(i) - 'A' + 1) * primeNo.get(i)
                                                  % mod).
}
long hashP = 0;
for (i=0; i<n) {
  hashPm = (hashP + (pat.charAt(i) - 'A' + 1) * primeNo.get(i)) % mod)
```

```
ArrayList<Integer> occurrences;
for(i=0; i+m-1<n;) {
  long curr = (h.get(m+i) + mod - h.get(i))% mod;
  if ( curr == hashr * primeRaised.get(i) % mod) {
        occurrences.add(i);
    }
{
return occurrences;
```

TC: O(m+n) SC: O(M)

Date: / / Page No.:

6.8) z-algorithm: to find all occurrence of patter in a string.

```
concat = pat + "$" + stri;
arr[] = new int[concat.length];
left=0   right=0
for(i=1; i<n; i++){
  if (i>right){
    left = right = i;
    while(right<n && concat.charAt(right-left)==
                     concat.charAt(right))
      {right++;}
    arr[i] = right - left
    right--;
  }
  else{
    k = i - left;
    if (arr[k] < right - i + 1){
      arr[i] = arr[k]; }
    else{
    left = i;
    while(right<n && concat.charAt(right-left) == concat.
                     charAt(right)){ right++;}
    arr[i] = right - left
    right--;
    }
  }
}
// get index using   if(arr[i] == pat.length()){
    for(i=1; i<n;){ ind = i - pat.length()-1;} }
```

TC: O(N+m) SC = O(N+M)

6) KMP : (pattern in string)

```
find lps (pat, lps[])
    i=1  len=0  lps[0]=0;
    while (i<strlen) {
        if (pat.charAt(i) == pat.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        }
        else {
            if (len != 0) { len = lps[len-1]; }
            else { lps[i] = len; i++; }
        }
    }
    i=0, j=0
    while (i < str.length()) {
        if (str.charAt(i) == pat.charAt(j)) { i++; j++; }
        if (j == pat.length()) ind = i-1;  j = lps[j-1];
        else if (i < str.length() && str.charAt(i) != pat.charAt(j)) {
            if (j!=0) j = lps[j-1];
            else i++;
        }
    }
```

TC: O(N+M) SC: O(M)

7) Minimum characters for Palindrome.

Count of min char that need to be added at front.

Date: __/__/__ Page No. : _____

```
lps (str, lps[]) {
i = 1, len = 0, lps[0] = 0
while (i < str.length()) {
    if (str.charAt(i) == str.charAt(len)) {
        len++;
        lps[i] = len;
        i++;
    }
    else {
        if (len != 0) { len = lps[len-1]; }
        else { lps[i] = 0;
               i++;
        }
    }
}

minChar (str) {
StringBuilder sb = new StringBuilder(str);
sb = sb.reverse();          lps[] = new int[c.length()]
String c = str + "$" + sb.toString();
lps(c, lps);
return str.length() - lps[c.length()-1];
}
```

TC : O(N) SC : O(N)

*1) check for Anagram : different words having same number of letters and same letters.

```
if (isAnagram) ans = true
```

```
if (str1.length != str2.length) return false
int No_of_char = 256
int hash[] = new int[256];
for (i=0; i<256; i++) {
        hash[i]=0;
}
int i;
for (i=0; i<str1.length; i++){
   hash[str1.charAt(i)]++;
   hash[str2.charAt(i)]--;
}
for (i=0; i<256; i++) {
    if (hash[i] !=0) { return false; }
{
return true;
```

TC: O(N) SC: O(N)

72) Count & Say : eg n=4 = (1) = 11
(2) = 21
(3) = 1211

```
string ans = "1";
iter = n-1;
while ( iter-- > 0) {
   curInd = 0,
   temp = "";
   ArrayList<Integer> count = new ArrayList<>();
while (curInd < ans.length()) {
   curChar = ans.charAt(curInd);
   charCount = 0
```

Date: __/__/__ Page No. : _____

```
while ( currInd < s.length && s.charAt(currInd) == currChar ){
        char count++;
        currInd++;
    }
    temp += currChar;
    count.add (char count);
  }
  s = "";
  for (i=0; i< count.size(); i++) {
     s += ('0' + count.get(i))
     s += temp.charAt(i);
    }
}
return s;
}
```

TC: $O(N^2)$ SC: $O(M)$ → length of final string

73) Compare version number:

a) type 001 == 01 == 1.

```
ArrayList<String> v1 = new ArrayList(Arrays.asList(v1.split("."))
same for v2 arraylist.
//add extra 0 to small array
while (v1.size() < v2.size()) v1.add("0");
while (v2.size() < v1.size()) v2.add("0");

for (i=0; i< v1.size(); i++) {
  int rv1 = Integer.parseInt(v1.get(i));
  rv2 = --- .(v2.get(i));
  if (v1 < v2) return -1;
```

else if (v2 < v1) return 1;

{

return 0;

TC (ON) SC : O(2N)

b) Type 2: without 1001 (illegal)

removeTrailZero(String a) {

p = a.length-1;

for (i = a.length-1; i >= 1; i -= 2) {

if (a.charAt(i) == '0' and a.charAt(i-1) == '.')

p -= 2; {

else break;

}

return a.subString(0, p+1);

}

a = removeTrailZero(a)

b = removeTrailZero(b)

start1 = 0, start2 = 0, end1 = 0, end2 = 0

while (true) {

while (end1 < a.length && a.charAt(end1) != '.') end1++

while (end2 < b.length() && b.charAt(end2) != '.') end2++;

if (end1 > end2) return 1;

else if (end1 < end2) return -1;

for (i = start1; i < end1; i++) {

if (a.charAt(i) > b.charAt(i)) return 1;

else if (b.charAt(i) > a.charAt(i)) return -1;

}

if (end1 == a.length && end2 == b.length) return 0;

Date: __/__/__ Page No. : _____

```
if (end1 == a.length) return -1;
else if (end2 == b.length) return 1;
start1 = end1   start2 = end2
end1++ , end2++;
```

TC: O(N+M) O(1) = SC.

Tree:

7a) Inorder:

a) Recursive:

```
if (root != null) {
inorder(root.left)
arr.add(root.val)
inorder(root.right)
} return arr;
```

b) Iterative:

```
Stack<TreeNode> st = new Stack();   node = root
while (true) {
    if (node != null) {
        st.push(node)
        node = node.left;
    }
    else {
        if (st.isEmpty()) break;
        node = st.pop();
        arr.add(node.val);
        node = node.right
    }
}
return arr;
```

TC: O(N) SC: O(N)

75. Preorder:

a)
```
if (root != null) {
    arr.add(root.val);
    preorder(root.left)
    preorder(root.right)
}
return arr.
```

b)
```
Stack<TreeNode> st = new Stack();
if (root == null) return arr;
st.push(root)
while (!st.isEmpty) {
    temp = st.pop();
    arr.add(temp.val)
    if (temp.right != null) st.push(temp.right);
    if (temp.left != null) st.push(temp.left);
}
return arr;
```

TC : O(N) SC: O(N)

76) Postorder

a)
```
if (root != null) {
    postorder(root.left)
    postorder(root.right)
    arr.add(root.val)
}
return arr
```

Date: __/__/__ Page No. : _____

b) stack<TreeNode> st = new stack();
TreeNode cur = root;
while (cur != null || !st.isEmpty()) {
if (cur != null) { st.push(cur) cur = cur.left; }
else {
temp = st.peek().right
if (temp == null) {
temp = st.pop();
arr.add(temp.val);
while (!st.isEmpty() && temp == st.peek().right) {
temp = st.pop();
arr.add(temp.val);
}
}
else {
cur = temp;
}
}
}
return arr;
TC: O(N) SC: O(N)

#7) Morris Inorder (Threaded BT)
cur = root.
while (cur != null) {
if (cur.left == null) {
ans.add(cur.val);
cur = cur.right;
}
else {

```
TreeNode prev = cur.left;
while (prev.right != null && prev.right != cur) {
    prev = prev.right
}
if (prev.right == null) {
    prev.right = cur;
    cur = cur.left;
}
else {
    prev.right = null
    ans.add(cur.val)
    cur = cur.right;
}
}
return ans;
```

TC: O(N) SC: O(1)

preorder if(---) {
ans.add(cur.val)
prev.right = cur
cur = cur.left
}
else {
prev.right = null
cur = cur.right
}

* Find all Numbers less than n whose digit have absolute diff 1

eg n = 20 ans = 10 12

```
Queue<long> q = new LinkedList<Long>();
q.add((long),)
while(!q.isEmpty) {
    step = q.poll();
    if (step<n && step>=10) ans.add(step);
    if (step<=0 || step>n) continue;
    lastD = step%10
    prev = step*10 + (lastD-1)
    post = step*10 + (lastD+1)
```

```
if (lastID == 0)  q.offer(pos)
else if (lastID == 9)  q.offer(pos)
else { q.offer(pos)  q.offer(pos) }
}
Collections.sort(ans);
return ans;
}
```

78) Left view / Right view of a tree:

```
leftView(root, ans, d) {
if (root == null) return
if (ans.size == depth) ans.add(root.val);
leftView(root.left, ans, d+1);
leftView(root.right, ans, d+1);
}
//for right
rightView(root.right, ans, d+1)
rightView(root.left, ans, d+1)
```

TC: O(N) SC: O(N)

79) Bottom View:

```
Queue<Node> q = new LinkedList();
Map<Integer, Integer> map = new TreeMap();
root.hd = 0
q.offer(root)
while (!q.isEmpty()) {
Node t = q.poll();
int hd = t.hd;
map.put(hd, t.val);
```

```
if (t.left != null) { t.left.hd = hd-1; map.put
                      q.add(t.left);
                    }
if (t.right != null) { t.right.hd = hd+1;
                       q.add(t.right);
                     }
}
for (Map.Entry<Integer, Integer> m : map.entrySet()) {
        ans.add(m.getValue());
}
return ans;
```

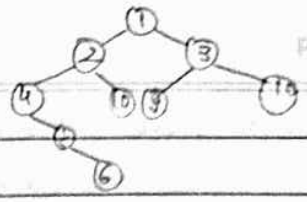
TC: O(N) SC: O(N)

Ex: Top view:

```
Queue<Pair> q = new LinkedList<>();
Map<Integer, Integer> map = new TreeMap<>();
q.add(new Pair(root, 0));
while (!q.isEmpty()) {
 Pair p = q.poll();
 Node t = p.node
 int l = p.l;
 if (!map.containsKey(l)) map.put(l, t.val);
 if (t.left != null) {
   q.offer(new Pair(t.left, l-1));
 }
 if (t.right != null) {
   q.offer(new Pair(t.right, l+1));
 }
}
return ans;
```

TC: O(N) SC: O(N)

Date: ___/___/___ Page No.: _____



8) Vertical order traversal

ans: 4 2 5 1 9 10 6 3 10

```
class Tuple { Node n; int row; int col; }

TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>>
    map;
Queue<Tuple> q = new LinkedList<>();
q.offer(new Tuple(root, 0, 0));
while (!q.isEmpty()) {
Tuple t = q.poll();
Node n = t.n;
int x = t.row, y = t.col;
if (!map.containsKey(x)) { map.put(x, new TreeMap<>()
                                    {
if (!map.get(x).containsKey(y)) map.get(x).put(y, new
                                    PriorityQueue<>());
map.get(x).get(y).offer(n.data);}
if (n.left != null) {
    q.offer(new Tuple(n.left, x-1, y+1));
if (n.right != null) q.offer(new Tuple(n.right, x+1, y+1));
}
for (TreeMap<Integer, PriorityQueue<Integer>> ys : map.values()) {
    list.add(new ArrayList<>());
    for (PriorityQueue<Integer> pq : ys.values()) {
        while (!pq.isEmpty()) {
            list.get(list.size()-1).add(pq.poll());
        }
    }
}
return list;
}
```

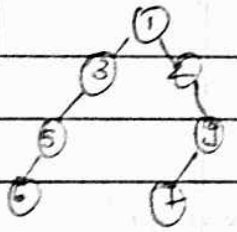
TC: O(N log N + log N + log N) SC: O(N)

Q2) Path to given Node:

```
if (a == null) return;
helper(a, b, ans)
Collections.reverse(ans);
return ans;
helper(Node a, int b, ans) {
  if (a == null) return false
  if (a.val == b) ans.add(a.val); return true;
  if (helper(a.left, b, ans) || helper(a.right, b, ans))
      ans.add(a.val);
      return true
return false;
}
```

T.C: O(N) S.C: O(1)

Q3) Max width of binary tree : (including null nodes between leftmost not-null & rightmost not-null node).



ans = 7

```
Queue<Pair> q = new LinkedList<>();
q.offer(new Pair(root, 0));
int max = 0;
while (!q.isEmpty()) {
  size = q.size()
  min = q.peek().num;   // for handling overflow
  int left = right = 0
  for(i=0; i<size; i++) {
  Pair p = q.poll();
  Node n = p.node;
  int num = p.num - min;
```

```
        if (i==0) first = num;
        if (i==size-1) last = num;
        if (n.left != null) q.offer(new Pair(n.left, 2*num+1));
        if (n.right != null) q.offer(new Pair(n.right, 2*num+2));
      }
      max = Math.max(max, right - left + 1);
    }
    return max;
```

T.C: O(N) S.C: O(N)

84) Level order traversal :

```
if(root == null) return ans;
Queue<Node> q = new LinkedList<>();
q.offer(root);
while (!q.isEmpty()) {
    size = q.size();
    sub = new ArrayList<>();
    for (i=0; i<size; i++) {
        Node n = q.poll();
        sub.add(n.val);
        if (n.left != null) q.offer(n.left);
        if (n.right != null) q.offer(n.right);
    }
    ans.add(new ArrayList(sub));
}
return ans;
}
```

T.C: O(N) S.C: O(N)

85) Height of Binary tree:

Recursive:

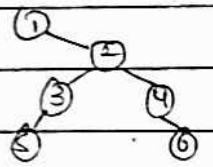
```
{ if (root == null) return 0;
return Math.max(1+maxDepth(root.left), 1+maxDepth(root.right));
}
```

SC: O(1) + O(H) H = height of tree

TC: O(N).

Iterative: level order traversal:

86) Diameter of tree: longest path between any two Node passing root or not.



ans = 4.

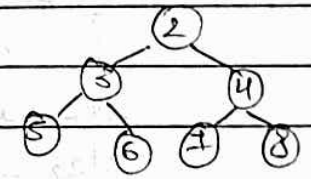
```
d[] = int[1];
height(root, d)
return d[0];
height (root, d[]) {
if (root == null) return 0;
l = height (root.left, d);
r = height (root.right, d);
d[0] = max(d[0], l+r)
return 1 + Math.max (l, r);
```

87) Check if balanced binary tree:

```
{ if (root == null) return true;
  return height(root) != -1;
}
height(root)
  if (root == null) return 0;
  l = height(root.left)
  if (l == -1) return -1;
  r = height(root.right)
  if (r == -1) return -1
  if (Math.abs(l-r) > 1) return -1
  return 1 + Math.max(l, r);
}
```

TC: O(N) SC: O(1)

88) Lowest Common Ancestor :



p = 8 q = 5
ans = 2.

```
lca(root, p, q)
{ if (root == null || p == root || q == root) return root.
  l = lca(root.left, p, q)
  r = lca(root.right, p, q)
  if (l == null) return right
  else if (right == null) return left
  else return root;
}
```

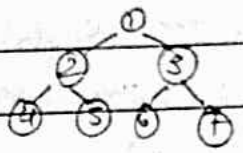
TC: O(N) SC: O(N) for space.

Eg) Check if two trees are identical or not :

```
{ if (p == null && q == null) return true;
  if (p == null || q == null) return false;
  return p.val == q.val && fun(p.left, q.left) && fun(p.right,
                                                      q.right)
}
```

TC: O(Min(M,N)) SC: O(Min(H_1, H_2))

90) ZigZag Traversal :



ans [[1], [3,2], [4,5,6,7]]

```
Queue<TreeNode> q;
q.offer(root);   flag = true;
while(q.isEmpty()) {
  size = q.size();
  for(i=0; i<size; i++) {
    TreeNode n = q.poll();
    if(flag) sub.add(n.val);
    else sub.add(0, n.val);
    if (n.left != null) q.offer(n.left);
    if (n.right != null) q.offer(n.right);
  }
  ans.add(sub);
  flag = !flag
}
return ans;
}
```

TC: O(N) SC: O(N)

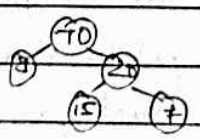
91) Boundary Traversal :

```
left(root, ans) {
if (root == null || (root.left == null && root.right == null))
    return;
ans.add(root.val)
if (root.left != null) left(root.left, ans)
else left(root.right, ans)
{
```

```
right (root, ans) {
if (root == null || (root.left == null and root.right == null)
        return
if (root.right != null) right (root.right, ans)
else right (root.left, ans)        ans.add (root.val)
}
leaf (root, ans) {
if (root == null) return
if (root.left == null && root.right == null) ans.add
                                              (root.val)
return
leaf (root.left, ans)
leaf (root.right, ans)
}
max () {
if (root == null) return
ans.add (root.val)
left (root.left, ans)
leaf (root.left, ans)
leaf (root.right, ans)
righ (root.right, ans)
}
```

O(N) O(N)

92) Path sum :



ans = 42

```
int dp = {Integer.MIN_VALUE}
max (root, d)
return d[0];
```

```
max (root, d){
if (root == null) return 0;
l = Math.max (max (root.left, d), 0)
r = Math.max (max (root.right, d), 0);

d[0] = Math.max (d[0], r+l+root.val);
return Math.max (r, l) + root.val;
}
```

TC: O(N) SC: O(N)

23) Construct tree from Inorder & preorder traversal:-

```
build (ino, preo)
{ Map<int, int> m;
for (i=1 to n)
map.put (in[i], i);

return construct (pre, 0, pre.length-1, in, 0, in.length-1,
map);
}
construct (pre[], startpre, endpre, in[], startin, endin,
map) {
if (startpre > endpre || startin > endin) return null;
root = new TreeNode (pre[startpre])
ind = map.get (root.val);
numleft = ind - startin;
root.left = construct (pre, startpre+1, startpre+numleft,
in, startin, ind-1, map);
root.right = construct (pre, startpre+numleft+1, endpre, in,
ind+1, endin, map);
return root;
}
```

TC: O(N) SC: O(N)

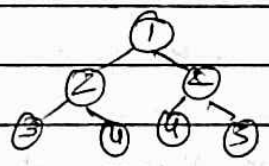
Date: __/__/__ Page No.: _____

94) Construct Tree using inorder & postorder traversal

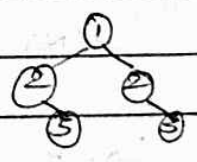
```
build(inorder, post) {
return helper(postorder, 0, post.length-1,
              in, 0, in.length-1)
}
helper(post, startp, endp, in, starti, endi) {
  if(startp > endp || starti > endi) return null;

  TreeNode root = new TreeNode(post[endp]);
  for(i = starti; i <= endi; i++) {
    if(in[i] == root.val) {
      k = i; break;
    }
  }
  root.left = helper(post, startp, startp+k-starti-1, in,
                     starti, k-1);
  root.right = helper(post, startp+k-starti, endp-1,
                      in, k+1, endi);
  return root;
}
```

TC: O(N) SC: O(N)

95) Symmetric tree :



ans = true



ans false

```
isSym(root) {
  if(root == null) return true;
  return helper(root.left, root.right);
}
```

```
helper (r1, r2) {
if (r1 == null || r2 == null) return r1 == r2;
boolean l, r = false
if (r1.val == r2.val) {
    l = helper (r1.left, r2.right);
    r = helper (r1.right, r1.left);
}
return l && r;
}
```

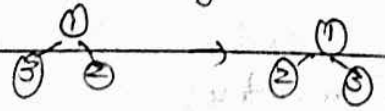
TC: O(N) SC: O(N).

96) flatten binary tree to ll : preorder traversal right left, root

```
prev = null
flatten(root) {
  if (root == null) return;
  flatten (root.right);
  flatten (root.left);
  root.right = prev
  root.left = null
  prev = root;
}
```

O(N) O(N)

97) Check binary tree is Mirror of itself :




```
if (root == null) return;
root = helper(root)
```

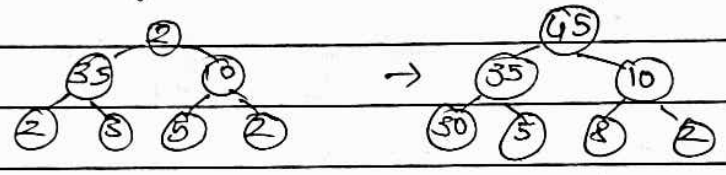
Date: __/__/__ Page No.: _____

```
Node helper(root) {
  if (root == null) return null;
  Node temp = helper(root.left);
  root.left = helper(root.right);
  root.right = temp;
  return root;
}
```

TC: O(N) SC: O(1) (with recursive stack)

9.) Children sum property:

val of a node = value of its children nodes



val can only be incremented

```
fun() {
if (root == null || (root.left == null && root.right == null)
    return;
diff = root.val - ((root.left != null) ? root.left.val : 0) + ((root.right
    != null) ? root.right.val : 0);

if (diff > 0) {
  if (root.left != null) root.left.val += diff
else if (root.right != null) root.right.val += diff
}
fun(root.left)
fun(root.right)
root.val = (root.left != null ? root.left.val : 0) +
           (root.right != null ? root.right.val : 0);
}
```

TC: O(N) SC: O(N)

Array :

99) Set Matrix Zero :

if an ele is 0 set entire row and col as 0.

```
is Col -
is Col = false
for (i=0; i<row; i++) {
  if (mat[i][0] == 0) isCol = true;
  for (j=1; j<col; j++) {
    if (mat[i][j] == 0) {
      mat[i][0] = 0;
      mat[0][j] = 0;
    }
  }
}

for (i = row-1; i>=0; i--) {
  for (j = col-1; j>=1; j--) {
    if (mat[i][0] == 0 || mat[0][j]==0) mat[i][j]=0;
  }
  if (isCol == true) {
    mat[i][0] = 0
  }
}
```

TC: O(2*(N*M)) SC: O(1)

100) Pascal triangle :

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

given a number return first numRow of Pascal triangle

```
List<List<Int>> gen(numRows) {
  List<List<Int>> list = new ArrayList<List<Int>>();
  for (i=0; i<numRows; i++) {
    ArrayList<Int> l = new ArrayList<>();
    for (j=0; j<=i; j++) {
      if (j==i || j==0) l.add(1);
      else {
        l.add(list.get(i-1).get(j-1) + list.get(i-1).
                                        get(j));
      }
    }
    list.add(l);
  }
  return list;
}
```

TC: O(numRows²) SC: O(numRows²)

101) Next permutation : input = [1 2 3] out = [1 3 2]
next should be lexograp. in = [3 2 1] out = [1 2 3]
ically greater :

```
i = arr.length-2;
while (i>=0 && arr[i] >= arr[i+1]) i--;
if (i>=0) {
  j = arr.length-1
  while (arr[j] <= arr[i]) j--;
  swap(arr, i, j)
}
reverse(arr, i+1, arr.length-1);
```

// find start of increasing seq

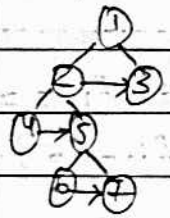
102) Maximum Subarray (Kadanes Algorithm)

nums = [5 4 -1 7 8] arr = 23

```
{ n = arr.length
max = Integer.MIN_VALUE, sum = 0
for (i=0; i<n; i++) {
  sum += arr[i]
  max = Math.max(max, sum);
  if (sum < 0) {
    sum = 0;
  }
}
return max;
}
```

O(N) O(1)

103) Popwate the next right pointer in each node




```
{ start = root
while (start != null) {
ptr = start
prv = null
start = null
while (ptr != null) {
if (ptr.left != null) {
  if (prev != null) {
    prev.next = ptr.left;
  }
if (start == null) {
  start = ptr.left;
}
prev = ptr.left;
```

```
}
if (ptr.right != null) {
    if (prev != null) {
        prev.next = ptr.right
    }
    if (start == null) {
        start = ptr.right
    }
    prev = ptr.right;
}
ptr = ptr.next;
}
}
```

O(N) O(1)

104) Search in BST :

```
while (root != null) {
if (root.val == x) return root;
else if (root.val > x) root = root.left
else root = root.right;
}
return null;
```

O(H) O(1)

105) Construct BST : given array in increasing order.

```
start = 0    end = arr.length - 1
bst (arr, start, end) {
if (start > end) return null;
Node root = new Node (arr[(start + end)/2]);
```

```
root.left = bst(arr, start, m-1);
root.right = bst(arr, start, m+1);
return root;
}
```

O(N) O(log(N));

105) construct BST from preorder traversal:

```
root = new Node(pre[0]);
for(i=1; i< pre.length; i++){
    construct(root, pre[i]);
}
return root;
}
construct(root, data){
if(root == null){
    node = new Node(data);
    return node;
}
if(root.val > data){
root.left = construct(root.left, data);
}
else root.right = construct(root.right, data);
return root;
}
```

O(N²) O(N²)

106) Check BT is a BST ? partial BST left node val <= root
right >= root.

Date: __/__/__ Page No.: _____

```
fun (root) {
 if (root == null) true;
 return helper(root, Long.MIN_VALUE, Long.MAX_VALUE
}
helper(root, long min, long max) {
 if (root == null) return true;
 if (root.val <= min || root.val >= max) return fa
 return helper(root.left, min, root.val) && helper(
                root.right, root.val, max);
}
```

O(N) O(N).

107) Lowest common ancestor of BST:

```
d1 = p.val   d2 = q.val
cur = root
while (cur != null) {
 if (d1 > cur.val && d2 > cur.val) cur = cur.right
 else if (d1 < cur.val && d2 < cur.val) cur = cur.left
 else break;
}
return cur;
```

O(N) O(1)

108) Inorder pre and successor of BST:

key = 60 and root given.

predecessor = ~~smallest~~ biggest on LHS

successor = smallest on RHS

```
pre = -1   suc = -1
while (pr root.data != key) {
if (key > root.data) {
    pr = root.data;
    root = root.right;
}
else {
    suc = root.data;
    root = root.left;
}
}
leftsub = root.left.
while (leftsub != null) {
pre    pre = root.leftsub.data;
    leftsub = leftsub.right;
}

rightsub = root.right;
while (rightsub != null) {
suc    rightsub pre = rightsub.data;
    rightsub = rightsub.left;
}
```

O(N) O(1)

109) Floor in BST : a key given, find greatest value less than equal to BST.key.

```
pr = null;
while (root != null) {
if (root.data == val) return root.data;
else if (root.data > x) root = root.left;
```

```
        else {
    prev = root ;
        root = root.right;
        }
    }
    return prev.data;
}
```

O(h) O(1);

110) Ceil in BST : given a key find smallest value greater than equal to key.

```
prev = null
while (root != null) {
    if (root.data == x) return x;
    else if (root.data > x) { prev = root;
                              root = root.left;
                            }
    else { root = root.right; }
}
return prev == null ? -1 : prev.data;
```

O(h) O(1)

111) Kth - smallest in BST :

count = 0, ans = 0

```
int Kthsm(root, K) {
    search(root, K);
    return ans;
}
```

```
search (root, k) {
if root == null return
search (root. left, k);
count++
if (count == k) ans = root. val
search (root.right}, k);
}
```

O(min(k,N)) O(min(k,N))

112 Kth largest element in BST :

```
int ans = 0          fun (root, k) {
                     arr [] = {k};
                     search (root, arr);
                     return ans;
                     }
                search (root, k) {
                if (root == null) return.
                search (root. right, arr)
                 arr[0]--
                if (arr[0] == 0) ans = root.val;
                search (root. left, arr);
                }
```

O(min (K,N)) O(min(K,N));

113) Two sum 4 : a target k return true if there exisicts two element such that there sum == k.

```
fun(root, k) {
HashSet<Integer> st = new HashSet<>();
return inorder(root, k, st);
}
inOrder(root, k, st) {
  if (root == null) return false;
  if (st.contains(k-root.val)) return true;
  st.add(root.val);
  return inorder(root.left, k, st) ||
         inorder(root.right, k, st);
}
```

O(n) O(n)

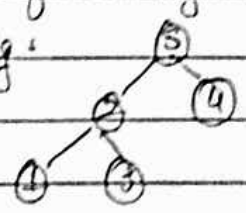
114) BST iterator :

1) Simple inorder traversal add elements in arrayList in ascending order, maintain pointer.

2) Maintain stack and fill all the elements from root to left when next() called pop the top element and check if top.right present if yes then fill stack with the element and element on left leftMostInorder(). O(1) Avg O(h)

115) Size of largest BST in Binary tree:

eg:



ans = 3

tree at root 2.

```
class NodeVal {
    int max, min, size;
    NodeVal (-max, min, -size) {
        max = -max, min = -min, size = -size
    }
}

return max(root).size;

NodeVal max (root) {
    if (root == null) {
    return new NodeVal (Integer.MIN_VAL, Integer.MAX_VAL,
                        0);
    }
    NodeVal left = max(root.left)
    NodeVal right = max(root.right);

    if (left.max < root.val && root.val < right.min) {
    return new NodeVal (Math.max(right.max, root.val), Math.min(
                        left.min, root.val), left.size + right.size
                        +1);
    }

    return new NodeVal (Integer.MAX, Integer.MIN, Math.max(
                        left.size, right.size));
}
```

O(N) O(N)

Date: __/__/__ Page No.: _____

116) Serialize and deserialize a binary tree.

```
serialize(root) {
if (root == null) return "";
Queue<TreeNode> q;
StringBuilder res;
q.offer(root);
while (!q.isEmpty()) {
TreeNode n = q.poll();
if (n == null) {
  res.append("n ");
  continue;
}
res.append(node.val + " ");
q.offer(n.left)
q.offer(n.right);
}
return res.toString();
}
deserialize(data) {
if (data == null || data.length() == 0) return null;
Queue<TreeNode> q;
String[] arr = data.split(" ");
TreeNode n = new TreeNode(Integer.parseInt(arr[0]));
q.offer(n);
for(i=1; i<arr.length; i++) {
TreeNode p = q.poll();
if (!arr[i].equals("n")) {
TreeNode left = new TreeNode(Integer.parseInt(
                                arr[i]);
p.left = left;
q.offer(left)
{
```

```
if (!arr[++i].equals("n")) {
TreeNode r = new TreeNode (Integer.parseInt(arr[i]));
p.right = r;
q.offer(r);
}
}
return n;
}
```

O(n) O(n)

Arrays:

117) sort 0,1,2 [inplace]:

in: [2,0,2,1,1,0] out = [0 0 1 1 2 2]

```
low = 0, high = len - 1;
for (i = low; i < high; ) {
if (nums[i] == 2) {
swap (nums, high, i);
high--;
}
else if (nums[i] == 0) {
swap (nums, low, i);
i++;
low++;
}
else i++;
}
```

O(n) O(1);

Date: __/__/__ Page No.: ____

1) c) Stock buy & sell : pick a buy price & a sell price to make out profit

in = [7, 1, 5, 3, 6, 4] ans = 5

```
buy = prices[0];
max = Integer.MIN_VALUE;
for (i=1; i<prices.length; i++) {
    if (buy > prices[i]) {
        buy = prices[i];
    }
    max = Math.max(max, buy-prices[i]);
}
return max < 0 ? 0 : max;
}
```

O(n) O(1)

1) g) Rotate matrix 90°:

IMP

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

transpose (matrix)
reflect (matrix)

```
transpose (matrix) {
  n = matrix.length;
  for (i=0; i<n; i++) {
    for (j = i+1; j<n; j++) {
      temp = matrix[i][j]
      matrix[i][j] = matrix[j][i]
      matrix[j][i] = temp;
    }
  }
}
```

```
rifluit (matrix) {
  n = matrix.length;
  for (i=0; i<n; i++) {
   for (j=0; j< n/2; j++) {
     temp = matrix [i] [j]
     matrix [i] [j] = matrix [i] [j n-j-1]
     matrix [i] [n-j-1] = temp;
   }
  }
}
```

O (N*N + N^N) O (1)

20) Merge Intervals:

eg: [[1,3], [2,6], [8,10], [15,18]]

out: [[1,6], [8,10], [15,18]]

```
Collections.sort (interval, (a,b) -> a.get(0) - b.get(0))
ArrayList< List<Integer>> merge = ---;

for (i=0; i< interval.length; i++) {
if (merge.isEmpty() || merge.get(merge.size()-1).get(1) <
                    interval.get(i).get(0)) {
  ArrayList <Integer> v;
  v.add (interval.get(i).get(0))
  v.add(interval.get(i).get(1));
  merge.add(v);
}
else {
merge.get(merge.size()-1).set (1, Math.max( merge.get(merge.
                    size()-1).get(1), interval.get(i).get(1));
}
}
return merge;
}
```

Date: __/__/__ Page No.: ____

$O(N \log N) + O(N) \approx O(N \log N)$ $O(N)$

121) Merge 2 arrays in increasing order : [in place]

arr1 = [1, 2, 3, 0, 0, 0] m = 3

arr2 = [2, 5, 6] n = 3

```
a1 = m-1 , a2 = n-1 , k = (m+n) - 1 ;
while (a1 >= 0 && a2 >= 0) {
  if (arr1[a1]
nums1[k] = (nums1[a1] > nums2[a2]) ? nums1[a1--]:
                                    nums2[a2--];
  k--;
}
while (a2 >= 0) {
  nums1[k--] = nums2[a2--];
}
}
```

O(N) O(1).

(log n) approach using gap method.

122) find duplicate in an array of N+1 integers.

find sum of first n natural numbers

find sum of arr element

take difference. ← ans

O(n)

```
slow = arr[0], fast = arr[0];
do {
  slow = arr[slow]
  fast = arr[arr[fast]]
} while (slow != fast);
```

```
fast = arr[0]
while (slow != fast) {
    slow = arr[slow]
    fast = arr[fast]
}
return slow;
}
```

O(N) O(1)

128) Missing and Repeating number in an array.

[1, 3, 5, 4, 4] ans [2, 4]

logic: take ele, make it ind, mark ind elem as -ve val, if same ele appears the ind is repeating and if a value is +ve the ~~val~~ ind is missing

[1 3 5 4 4], [-1, 3, 5, 4, 4], [-1, 3, -5, 4, 4], [-1, 3, -5, 4, -4], [-1, 3, -5, -4, -4], [-1, 3, -5, -4, -4] r = 4 as repeating

```
for (int i=0; i<n; i++) {
  if (arr.get( ~~arr~~ Math.abs(arr.get(i)) - 1) > 0) {
    x = abs(arr.get(i));
    arr.set(x-1, - arr.get(x-1));
  }
  else { r = abs(arr.get(i)); }
}
for (i=0; i<n; i++) {
  if (arr.get(i) > 0) { m = i+1; break; }
}
ans[] = {m, r}
return ans;
```

O(N) O(1)

Date: __/__/__ Page No. : _____

124) Count inversions :

Cond : 1) $i < j$ 2) $arr[i] > arr[j]$ then inversion

[2 5 1 3 4] = 4

```
mergesort (arr, 0, n-1)

mergsort (arr, left, right) {   long inv = 0;
  if (right > left) {
  inv = mergesort(
  mid = (left + right)/2
  inv+= mergesort (arr, left, mid);
  inv += mergesort (arr, mid+1, right);

  inv += merge (arr, left, mid+1, right);
  } return inv;
}

merge (arr, left, mid, right) {
  i = left, j = mid, k = 0;
  temp [] = new int/long [right - left + 1];
  long inv = 0;
while ( (i < mid) && (j <= right)) {
    if (arr[i] <= arr[j]) {
      temp [k] = arr [i]
        i++; k++;
    }
    else {
      inv += (mid - i)
      arr[k] = arr[j];
      k++; j++;
    }
}
```

```
while (i<mid) {
temp arr[k] = arr[i]
    k++; i++;
}
while (j<=right) {
  temp[k] = arr[j]
    k++; j++;
}          k=0
for (i=left; i<right; i++) {
  { arr[i] = temp[k];
  }
return inv;
}
```

O(N * log(N)) O(N)

12 c) Search element in 2D Matrix:

1	3	5	7
10	11	16	20
23	30	34	60

3

ans = true

```
if (matrix.length==0) return false;
lo = 0  hi = row * col - 1
while (lo <= hi) {
if (mat[  m = (low+high)/2
  if (mat[m/c][m%c] == tar) return true;
  else if (mat[m/c][m%c] < tar) { l = m+1;
  else { hi = m-1;}
}
return false;
```

O(log(m*n)) O(1)

Date: __/__/__ Page No. : _____

12 c) Pow(x, y) (Binary Exponential)

eg: x = 2.0000 n = 10 ans = 1024.00000

2) x = 2.00000 n = -2 ans = $\frac{1}{4}$ = 0.25

```
long nn = n;        ans = 1.0
if (nn < 0)  nn = -nn ;
while (nn > 0) {
  if (nn % 2 == 1) {
     ans = ans * x;
     nn--;
  } else {
     x = x * x;
     nn = nn/2; }
}
if (n < 0)  ans = 1/ans ;
return ans;
}
```

O(log N) O(1)

12 f) Find ele that appears more than ⌊N/2⌋ times.
(Moore's Voting algo)

[3 2 3] → 3

```
cand = 0
count = 0
for (num : nums) {
  if (count == 0)  cand = num;
  if (num == cand)  count++;
  else  count--;
}
return cand;
}
```

O(N) O(1)

128) Majority element II : ele having freq greater than $\lfloor N/3 \rfloor$
(there will be at max 2 element)

```
num1 = -1, num2 = -1, co1 = 0  co2 = 0
for (i=0; i < len; i++) {
  if (num1 == arr[i]) { co1++; }
  else if (num2 == arr[i]) co2++;
  else if (co1 == 0) { num1 = arr[i]; co1 = 1; }
  else if (co2 == 0) { num2 = arr[i]; co2 = 1; }
  else { co1--; co2--; }
}
  co1 = 0  co2 = 0
for (i=0; i< len; i++) {
  if (num1 == arr[i]) co1++;
  else if (num2 == arr[i]) co2++;
}
if (co1 > n/3) ans.add(num1);
if (co2 > n/3) ans.add(num2);
return ans;
}
```

129) Count unique paths from top left to bottom right right and down move allowed.

dp in : 2, 2 ans 2
in : 3, 2 ans 3

```
dp[][] = new int[n+1][m+1];
for (int row[] : dp) Arrays.fill(row, -1);
return count(0, 0, m, n, dp);
```

Date: __/__/__ Page No.: _____

```
count (i, j, m, n, dp) {
    if (i == m-1 && j == n-1) return 1;
    if (i >= m || j >= n) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    else return dp[i][j] = count(i+1, j, m, n, dp)
                         + count(i, j+1, m, n, dp)
}
```

$O(m*N)$ ~~$O(N)$~~ $O(N*m)$

Combinatoric logic: there are $m+n-2$ steps to reach the end. and we need exact $n-1$ or $m-1$ right and down moves respectively.

$\therefore {}^{m+n-2}C_{n-1}$

```
N = m+n-2
r = m-1
res = 1
for (i=1; i<=r; i++) {
    res = res * (N-r+i)/i;
}
return (int) res;
}
```

$O(\min(n-1, m-1))$ $O(1)$

130) Count inversion pairs:
a pair having 1) $i<j$ 2) $a[i] > 2*a[j]$
eg. 1 5 2 3 1 ans = 2
[5, 1] [5, 2]

```
return mergeSort(nums, 0, nums.length-1)

mergeSort(nums, low, high) {
  if (low >= high) return 0;
  int mid = (low+high)/2
  inv = mergeSort(nums, low, mid)
  inv += mergeSort(nums, mid+1, high)
  inv += merge(nums, low, mid, high);
  return inV;
}
merge(nums, low, mid, high) {
  i= low   j=mid+1   cnt=0
for(i=low; i<=mid; i++) {
    while(j<=high && arr[i] > (2* (long) arr[j]))
      j++;
    }
    cnt += j - (mid+1);
}
ArrayList<Integer> temp;
left = low   right = mid+1
while (left <= mid && right <= high) {
    if (arr[left] <= arr[right]
        temp.add(arr[left++]);
    else temp.add(arr[right++]);

while (left <= mid) {
    temp.add(arr[left++]);

while (right <= high) {
    temp.add(arr[right++]);
}
```

```
for (i= low; i<=high ;i++){
    arr[i] = temp.get(i- low);
}
return ans;
}
```

O (N log N) + O(N) + O(N)

131) 2-Sum :

in: [2, 7, 11, 15] tar = 9

```
Map <Integer, Integer> mp:
for (i=0; i< nums.length; i++){
  if (mp.containsKey (target - nums[i])){
    return int[] { i, mp.get (target - nums[i])};
  }
  mp.put(nums[i], i);
}
return int [] {0, 0};
}
```

O(N) O(N)

132) 4-Sum :

```
ArrayList<List<Integer>> res;
if (nums == null || nums.length == 0) return res;
n = nums.length
Arrays.sort(nums);
for (i=0; i<n; i++){
    for (j=i+1; j<n; j++){
```

```
tar = tar - num[i] - num[j]
front = j+1
back = n-1
while (front < back) {
if (num[front] + num[back] < tar) front++;
else if (num[front] + num[back] > tar) back--;
else {
List<Integer> ar = new ArrayList<>();
ar.add(i);
ar.add(j)
ar.add(front)
ar.add(back)
res.add(ar);
while (front < back && num[front] == ar.get(2))
++front
while (front < back && num[back] == ar.get(3))
back--;
}
}
while (j+1 < n && num[j+1] == num[j]) j++;
}
while (i+1 < n && num[i+1] == num[i]) i++;
}
return res;
}
```

O (N^3) $\mathcal{O}(1)$

1.3) Consecutive sequence : O(n)

in : [100, 1, 200, 4, 2, 3] ans = 4 (1 2 3 4)

```
max = 0
HashSet<Integer> hs = new HashSet<>();
for (int i : nums) { hs.add(i); }
for (int n : nums) {
    if (!hs.contains(n-1)) {
        currS = 1
        curr = n;
        while (hs.contains(curr+1)) {
            curr++;
            currS++;
        }
        max = Math.max(max, currS);
    }
}
return max;
O(N)        O(N)
```

134) Length of longest subarray with zero.

[9 -3 3 -1 6 -5] ans = 5

```
HashMap<Integer, Integer> hm;
int maxi = 0, sum = 0;
for (i=0; i<arr.length; i++) {
    sum += arr[i]
    if (sum == 0) { maxi = i+1; }
    else {
        if (hm.contains(sum))
            maxi = Math.max(maxi, i - hm.get(sum));
        else hm.put(sum, i);
        {
    }
return maxi;        O(N)    O(N)
```

135) Count no. of subarray with given xor k.

```
HashMap<Integer, Integer> hm;
cpx = 0, c = 0
for (i=0 ; i< arr.length ; i++){
    cpx = cpx ^ A[i];
    if (hm.containsKey(cpx ^ B){
        c += hm.get(cpx ^ B); }
    if (cpx == B) c++
    if (hm.containsKey(cpx)) {
        hm.put(cpx, hm.get(cpx)+1);
    else  hm.put(cpx, 1);
    }
return c;
}
```

O(N) O(N)

136) longest substring with no repeating character.

```
HashMap < character, Integer> hm;
right = 0   left = 0
while (r < n) {
if (hm.containskey(s.charAt(right))
        left = Math.max(left,
                map.get(s.charAt(i right))+1);
map.put(s.charAt(right), right);
max = Math.max(max, right - left+1);
right++;
}
return max
```

O(N) O(N)

Date: __/__/__ Page No. :

Linked List:

137 Reverse a linked list:

```
curr = head
prev, next = null
while (curr != null) {
    next = curr.next;
    curr.next = prev;
    prev = curr
    curr = next
}
return prev;
```

O(N) O(1)

138) find middle of linked list.

```
slow = head  fast = head
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
}
return slow;
```

O(N) O(1)

139) Merge two sorted Array:

```
if (l1 == null) return l2;
if (l2 == null) return l1;
if (l1.val > l2.val) {
    l2.next = merge(l1, l2.next)
    return l2;
}
```

```
else {
    l1.next = merge(l1.next, l2);
    return l1;
  }
}
```

O(n) O(1)

140) Remove kth node from back :-

```
ListNode s = new ListNode;
s.next = head;
ListNode slow = s, fast = s;
for (i=1; i<=k; i++) {
    fast = fast.next;
}
while (fast.next != null) {
    fast = fast.next;
    slow = slow.next;
}
slow.next = slow.next.next;

return start.next;
}
```

O(N) O(1)

152) Add two numbers represented as LL:

```
ListNode d = new ListNode();
ListNode temp = d;
int carry = 0
while (l1 != null || l2 != null || carry != 0)
```

Date: __/__/__ Page No. : _____

```
{ int sum = 0
  if (l1 != null) {
    sum += l1.val;
    l1 = l1.next;
  }
  if (l2 != null) {
    sum += l2.val;
    l2 = l2.next;
  }
  sum += carry
  carry = sum/10
  ListNode n = new ListNode(sum%10);
  temp.next = n;
  temp = temp.next;
}
return d.next;
}
```

O(max(n,m)) O(max(n,m));

142) Delete give Node in LL:

```
delete(t) {
  if (t == null) return;
  t.num = t.next.num;
  t.next = t.next.next;
  return;
}
```

O(1) O(1)

Graph

143) Clone a graph :

```
clone (Node n) {
HashMap <Integer, Node> hm = new HashMap <> ();
if (n == null) { return null; }
if (hm. containsKey (n.val)) { return hm.get(n.val);

Node clone = new Node (n.val);
hm.put (n.val, clone);
for (Node it : clone.neighbor) {
    clone.neighbor.add ( cloneGraph(it));
}
return clone;
}
```

144) DFS : (for connected and non connected component)

V = vertices e = edges

```
int vis[] = new int[v];
Arrays. fill (vis, 0)
for (i=0; i<v; i++) {
    if (vis[i] == 0) {
        dfs (vis, adj, i);
    }
}

dfs ( vis, adj, node) {
vis [node] = 1;
ans. add(node);
for (int it; adj.get(node)) {
    if (vis[it] == 0) {
        dfs ( vis, adj, it);
    }
}}
```

O(N+e), O(N+E) + O(N) + O(N)

Date: __/__/__ Page No. : _____

145) BFS :

```
int vis[] = new int[V];   Arrays.fill(vis, 0);
Queue<Integer> q = new LinkedList<>();
q.add(0);
vis[0] = 1;
while(!q.isEmpty()){
    int n = q.poll();
    ans.add(n);
    for(int i : adjget(n)){
        if(vis[i] == 0){
            vis[i] = 1;
            q.add(i);
        }
    }
}
return ans;
```

O(N+E) O(N+E) + O(N) + O(N)

146) Cycle in undirected graph (BFS).

```
class Node {
    int f, s;
    Node(fi, se){
        f = fi; s = se;
    }
}
```

```
boolean vis[] = new boolean[V];
Arrays.fill(vis, false);
```

```
for (i=0; i<V; i++) {
    if (vis[i] == false) {
        if (bfs(adj, vis, i)) return true;
    }
}
return false;

bfs ( - - - - - ) {
Queue <Node> q = ...;
q.add(new Node(s, -1));
vis[s] = true;
while (!q.isEmpty()) {
int u = q.peek().f;
int par = q.peek().s;

for (int it : adj.get(u)) {
    if (vis[it] == false){
        q.offer(new Node(it, u));
        vis[it] = true;
    }
    else if (par != it) return true;
}
}
return false;
```

O(N+E) O(N+E) + O(N) + O(N)

Date: ___/___/___ Page No.: ____

147) Cycle in Undirected graph (DFS).

```
boolean vis[] = new boolean[V];
Arrays.fill(vis, false);
for(i=0; i<V; i++) {
  if (vis[i] == false) {
    if (check(i, -1, vis, adj)) return true;
  }
}
return false;
check(node, par, vis[], ArrayList<ArrayList<>> adj) {
vis[node] = true;
for (int i: adj.get(node)) {
  if (vis[i] == false) {
    if (check(i, node, vis, adj) == true)
      return true;
  }
  else if (i != par) return true;
}
return false;
}
```

O(~~E+V~~ N) O(N)

150) Cycle in directed graph (DFS):

```
vis[] = new boolean [n+1];
dvis[] = new boolean [n+1];
Arrays.fill (vis, false);
for (i=0; i<n; i++) {
  if (vis[i] == false) {
    if (check(adj, vis, dvis, i)) return true;
  }
}
return false;
```

```
check(adj, vis, dvis, n) {
 vis[n] = true;
 dvis[n] = true;
 for (int i: adj.get(n)) {
    if (vis[i] == false) {
       if (check(adj, vis, dvis, n)) return true;
    }
    else if (dvis[i] == true) return true;
 } dfs dvis[n] = false;
 return false;
}
```

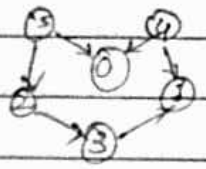
O(V+E) O(V)

151) Topological sort (BFS) + (Cycle in Directed (BFS)).

```
int topo[] = new int[n];
int ind[] = new int[n]

for (i=0; i<n; i++) {
    for (int j: adj.get(i)) {
        ind[j]++;
    }
}

Queue<Integer> q = ;
for (i=0; i<n; i++) {
    if (ind[i] == 0) {
        q.add(i);
    }
}
cnt=0  ind=0
while (!q.isEmpty()) {
```

Date: __/__/__ Page No. : _____

```
n = q.poll();
topo[ind++] = n;
cnt++;
for (it : adj.get(n)) {
    ind[it]--;
    if (ind[it]==0) q.offer(it);
}
}
if (cnt == N) return false;
return true;
}
```



adj: 4 5 2 0 3 1

O(N+E) O(N) + O(N)

152) Topo sort (DFS)

```
vis[] = new int[V];
Arrays.fill(vis, 0);
Stack<Int> st;
for (i=0; i<V; i++) {
    if (vis[i]==0) dfs(adj, vis, st, i);
}
topo[] = new int[V];
for (i=0; i<N; i++) {
    topo[i] = st.pop();
}
return topo;
}
dfs( ) {
vis[n] = 1;
for (int i: adj.get(n)) {
    if (vis[i]==0) dfs(adj, vis, st, i)
}
st.push(n);
}
```

O(N+E), O(N)+O(N)+O(N)

153) Count No. of islands: (8 dir or 4 dir)

eg: 0 1 1 0 0 ans = 3
1 0 0 1 0
0 0 1 0 0
1 0 0 0 1

```
// visited array creation
vis[][] = new int[mat.length][mat[0].length];
for (i=0; i<mat.length; i++) {
  for (j=0; j<mat[0].length; j++) {
    if (mat[i][j]==1) vis[i][j] = true;
    else vis[i][j] = false;
  }
}

count=0;
for (i=0; i<m; i++) {
  for (j=0; j<n; j++) {
    if (mat[i][j] == 1 && vis[i][j] == true) {
      vis[i][j] = false;
      dfs(vis, mat, i, j)
      count++;
    }
  }
}
return count;

dfs(vis, mat, i, j) {
  dx[] = {-1, -1, 0, 1, 1, 1, 0, -1};
  dy[] = {0, 1, 1, 1, 0, -1, -1, -1};
```

```
for(k=0; k<5; k++){
    x = i + dx[k]   y = j + dy[k];
    if (x>=0 && y>=0 && x<mat.length &&
        y<mat[0].length && mat[x][y]==1
        && vis[x][y]==true){
            vis[x][y] = false;
            dfs(mat, vis, x, y);
        }
    }
}
```

O(N*M) O(N*M)

154) Bipartite graph (BFS)

```
{color[] = new int[n];
 Arrays.fill(color, -1);
 for(i=0; i<n; i++){
   if (color[i]==-1){
     if (bfsCheck(adj, color, i)==false)
         return false
   }
 }
 return true;
}
bfsCheck(adj, color, node){
 color[node] = 1
 Queue<Integer> q;
 q.add(node);
 while (!q.isEmpty()){
 n = q.poll();
 for (int i: adj.get(n)){
```

```
        if (color[i] == -1) {
            color[i] = 1 - color[n];
            q.offer(i);
        }
        else if (color[i] == color[n]) return false
    }
}
return true;
}
```

O(N*M) O(N)

155) Bipartite (DFS)

```
color[] = new int[n];
Arrays.fill(color, -1)
for (i = 0; i<n; i++){
    if (color[i] == -1){
        color[i]=1;
        if (!dfscheck(graph, color, i)) return false;
    }
}
return true.

dfscheck(graph, color, node){
    for (int it: graph[node]){
        if (color[it] == -1){
            color[it] = 1 - color[node]}
            if (!dfs(graph, color, it)) return false;
        }
        else if (color[it] == color[node]) return false;
    }
return true;
}
```

O(V+E) O(V)

Date: __/__/__ Page No.: _____

156) Strongly Connected Component (Kosaraju's Algorithm)

steps : 1) topo sort
2) transpose the edges
3) rev topo sort.

```
vis[] = new int[n];
for (i=0; i<n; i++){
    if (vis[i] == 0){
        dfs(st, adj, vis, i);
    }
}
ArrayList<ArrayList<Integer>> tran;
for (i=0; i<n; i++){
    vis[i] = 0;
    for (int it: adj.get(i)){
        tran.get(it).add(i);
    }
}
List<List<Integer>> ans;
while(!st.isEmpty()){
    int node = st.pop();
    if (vis[node] == 0){ ArrayList<Integer> sub;
        revDfs(transpose, vis, sub, node);
        ans.add(sub);
    }
}
return ans;

dfs(stack, adj, vis, i){
    vis[i] = 1;
    for(int it: adj.get(i)){
```

```
        if (vis[it]==0) {
            dfs(st, adj, vis, it)
        }
    }
    st.push(i);
}
                    sub
revDfs (e trans, vis, node) {
    vis[node] = 1;
    sub.add(node)
    for (int it: trans.get(node)) {
        if (vis[it]==0){
            revDfs(trans, vis, sub, it);
        }
    }
}
```

$O[N+E]$

15) Dijkstra's algo:

```
class Node implements Comparator<Node> {
    int v, wei;
    Node() {}
    Node (int _v, _wei) {
        v = _v; wei = _wei;
    }
    int compare (Node n1, Node n2) {
        if (n1.wei > n2.wei) return 1;
        else if (n1.wei == n2.wei) return 0;
        else return -1;
    }
}
```

```
for(i=0; i< edges.length; i++){
    adj.get(edges.get(i).get(0)).add(new Node(
        edges.get(i).get(1), edges.get(i).get(2));
    adj.get(edges.get(i).get(1)).add(new Node(edges.get(i).
        get(0), edges.get(i).get(2));
}
dist[] = new int[V];
Arrays.fill(dist, Integer.MAX_VALUE);
dist[S] = 0;
PriorityQueue<Node> pq = new PriorityQueue<Node>(new
                                                 Node(i));
pq.add(new Node(S, 0));
while(!pq.isEmpty()){
  ver = pq.peek().v;
  wei = pq.peek().wei;
  pq.poll();
  for(Node it : adj.get(ver)){
    if(it.wei + dist[ver] < dist[it.gv]){
        dist[it.gv] = it.wei + dist[ver]
        pq.add(new Node(it.v, it.wei + dist[ver]));
    }
  }
}
return dist;          O((N+E) * log N)        O(N)
```

15) Bellman Ford (Negative Weight)

```
dist[] = new int[V];
Arrays.fill(dist, 10000000);
dist[src] = 0
```


```
for (i=0; i<v; i++) {
    for (j=0; j<m; j++) {
        u = edges.get(j).get(0);
        v = edges.get(j).get(1);
        w = edgs.get(j).get(2);

        if (dist[u] != 10000000 && dist[u] + w < dist[v])

            dist[v] = dist[u] + w;

    }
}
return dist;   or return dist[dest];
```

to check negative cycle in graph.

```
for (j=0; j<m; j++) {
    u = edges.get(j).get(0);
    v = edges.get(j).get(1);
    w = edgs.get(j).get(2);
    if (dist[u] + w < dist[v]) {
        flag = true;
        break;
    }
}
if (flag)  Negative cycle.
```

O(N*E) O(N)

159 All pair shortest path (Floyd-Warshall's)
(contains negative edges)

INF = infinite.

```
for (k=0; k<n; k++) {
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if (mat[i][k] == -1 || mat[k][j]==-1)
                continue;
            else if (mat[i][j] == -1) {
                mat[i][j] = mat[i][k] + mat[k][j];
            }
            else if (mat[i][j] > mat[i][k] +
                                mat[k][j]) {
                mat[i][j] = mat[i][k] + mat[k][j];
            }
        }
    }
}
return mat; or return mat[src][dest];
```

$O(V^3)$ $O(V^2)$

160) Prim's MST:

```
class Node implements Comparator<Node> {
    int v, w;
    Node () {}
    Node (int _v, int _w) { v = _v; w = _w; }
    int Compare (Node n1, Node n2) {
    if (n1.w < n2.w) return -1;
    else if (n1.w == n2.w) return 0;
      else return 1;
}
```

```
                                    edges
for (i=0 ; i<m ; i++) {
    adj.get( int u = edges.get(i).get(0);
             v = edges.get(i).get(1);
             w = edges.get(i).get(2);
    adj.get(u).add(new Node(v, w));
    adj.get(v).add(new Node(u, w));
}
PriorityQueue<Node> pq = new PriorityQueue(new Node());
Key[] = new int[v];        // cost of that vertex.
parent[] = new int[v];     // parent of a vertex.
mst[] = new boolean[v];    // for tracking a vertex

Arrays.fill(Key, 1000000000);
Arrays.fill(parent, -1);
Arrays.fill(mst, false);
Key[0] = 0
pmst[0] = true;
pq.add(new Node(0,0));
while(!pq.isEmpty()){
    v = pq.peek().v;
    w = pq.peek().w;
    pq.poll();
    mst[v] = true;
    for(Node n : adj.get(v)){
        if(mst[n.v] == false && Key[n.v] > n.w){
            parent[n.v] = v;
            Key[n.v] = n.w
            pq.offer(new Node(n.v, Key[n.v]);
        }
    }
}
```

$O(N \log N)$ $O(N)$

161) Kruskals MST :

```
Union (u, v, parent, rank) {
  u = findParent (u, parent);
  v = findParent (v, parent);
  if (rank[v] < rank[u]) {
     parent[v] = u;
  }
  else if (rank[u] < rank[v]) {
     parent[u] = v;
  }
  else {
     parent[u] = v;
     rank[v]++;
  }
}

findParent (u, parent[]) {
   if (u = parent[u]) return u;
   return parent[u] = findParent (parent[u], parent)
}
for (i=0; i<=n; i++) { parent[i]=i; rank[i]=0; }
  for (Node n: adj) {
     if (findParent (n.getU(), parent) != findParent (n.getV(), parent))
     { mst.add(n);
       union (n.getU(), n.getV(), parent, rank);
       mstCost += n.w;
     }
}
```

O(E log E) + O(E * 4 * alpha)
O(N)

Dynamic Programming :

162) Max product subarray :

Ex: [2, -5, -2, -4, 3] ans = 24

sol: -2 * -4 = 8 * 3 = 24

```
{ max = nums[0];
  curMax = 1, curMin = 1;
  for (i=0; i<n; i++) {
    temp = curMax * nums[i];
curMax = Math.max(nums[i], Math.max(temp, curMin * nums
curMin = Math.min(nums[i], Math.min(temp, curMin * nums[i]))

max = Math.max(max, Math.max(curMin, curMax));
}
return max;
}
```

O(N) O(1)

Another sol: traverse array from both dir and whenever prod == 0 prod = 1

163) Longest Increasing subsequence :

1) O(N^2) O(N)

```
dp[] = new int[n]
Arrays.fill(dp, 1);
for (i=1; i<n; i++) {
    for (j=0; j<i; j++) {
```

```
        if (arr[prev] < arr[i])
        dp[i] = Math.max(dp[i], dp[j]+1);
      }
    }

    for (i=0; i<n; i++) {
      max = Math.max(max, dp[i]);
    }
    return max;
```

2) Binary Search.

```
int lowerbound(arr, start, end, ele) {
  while (start < end) {
    // mid = start + (end-start)/2;
    if (ele > arr[mid]) {
      low = mid+1;
    }
    else high = mid;
  }
  return low;
}
int temp[] = new int[n];   ans = 0
for (i=0; i<n; i++) {
  int ind = lowerbound(&temp, 0, ans, arr[i]);
  temp[ind] = arr[i];
  if (ind == ans) ans++;
}
return ans;
}
```

O(N lg N) O(N).

164) LCS :

```
dp[][] = new int[n+1][m+1];
for (i=0; i<=n; i++)  dp[i][0] = 0
for (i=0; i<=m; i++)  dp[0][i] = 0

for (i=1; i<=n; i++) {
    for (j=1; j<=m; j++) {
        if (s.charAt(i-1) == t.charAt(j-1)) {
            dp[i][j] = dp[i-1][j-1] + 1;
        }
        else dp[i][j] = Math.max(dp[i-1][j-1], Math.
                        max(dp[i-1][j], dp[i][j-1]));
    }
}
return dp[n][m];
```

O(n+m) O(N*M)

165) 0/1 Knapsack : n=4 w=5

val=[5 4 8 6] ans = 13

w = [1 2 4 5]

Memoization:

```
dp[][] = new int[n][wt]
for (int row[] : dp) Arrays.fill(row, -1)
return helper(values, weight, n-1, w, dp);

helper(val, wei, ind, w, dp[][]) {
```

```
if (ind == 0) {
  if (wei[0] <= w) return val[0];
  return 0;

if (dp[ind][w] != -1) return dp[ind][w];

int notake = helper(val, wei, ind-1, w, dp)
int take = 0;
  if (wei[ind] <= w) take = val[ind] + helper(val, wei, ind-1, w-wei[ind], dp)
}
return dp[ind][w] = max(take, notTake);
}
```

TC: O(N*W) SC: O(N*W) + O(N) → stack space recursion.

BU Tabulation:

```
dp[][] = new int[n][w+1]
for(int row[] : dp) Arrays.fill(row, 0);

for(int i = wei[0]; i<=w; i++) dp[0][i] = val[0];

for(i=1; i<n; i++) {
for(int w=0; w<=w; w++) {
notTake = dp[i-1][w];
take = 0;
if(w<=w)
if(wei[i] <= w){
take = val[i] + dp[i-1][j - wei[i]];
}
```

```
        dp[i][j] = Max(take, notTake);
    }
}
return dp[n-1][w];
```

O(N*W) O(N*W)

166) Edit Distance : ins = 1 edit = 1 dele = 1

horse ros = 3

Memo :

```
{ if (ind1 < 0) return ind2+1;
  if (ind2 < 0) return ind1+1;

  if (dp[ind1][ind2] != -1) return dp[ind1][ind2];

  if (str1.charAt(ind1) == str2.charAt(ind2))
      return dp[ind1][ind2] = 0 + helper(str1, str2,
                                  ind1-1, ind2-1);
  else dp[ind1][ind2] = 1 + Min( dp.helper(s1, s2, ind1-1,
                                  ind2-1), min(
  helper(s1, s2, ind1-1, ind2), helper(s1, s2, ind1, ind2-1));
}
```

O(N*M) O(N+M) + O(N+M)

Tabul :

```
dp[][] = new int[N+1][M+1];
for (i=0; i<=N; i++) dp[i][0] = i;
for (j=0; j<=M; j++) dp[0][j] = j;
```

```
for (i=1; i<=N; i++) {
 for (j=1; j<=M; j++) {
  if (str1.charAt(i) == str2.charAt(j-1))
    dp[i][j] = 0 + dp[i-1][j-1];
  else {
  dp[i][j] = 1 + Min(dp[i-1][j-1], Min(dp[i-1][j],
                                       dp[i][j-1]));
  }
 }
}
return dp[N][M];
```

O(N*M) O(N*M)

167) Max sum increasing subsequence :

eg: 9 1 2 8 and 11

Memoization :

```
for (i=0; i<n; i++) max = max(max, arr[i])
dp[][] = new int[n][max+1];
for (int row[]: dp) Arrays.fill(row, -1);

return helper(arr, 0, 0, dp)
helper(arr, ind, prev, dp)
 if (ind == arr.length) return 0;

 if (dp[ind][prev] != -1) return dp[ind][prev];
 notake = 0 + helper(arr, ind+1, prev, dp);
 if (prev < arr[ind]) {
    take = arr[ind] + helper(arr, ind+1, arr[ind], dp);
 }
```

return dp[ind][prv] = max(take, notake);

}

SC: O(N*M) + O(N) TC: O(N*M)

eg | 1 | 4 | 8 | 2 | 1 | 4 | 8 | 2 |
   | 1 | 5 | 13 | 3 | 1 | 7 | 15 | 8 |

Tabulation:

```
dp[] = new int[n];
Omax = MIN_VALUE.
for (i=0; i<n; i++){
  max = -1
  for (j=0; j<i; j++){
    if (arr[j] < arr[i]){
      if (max < dp[j]) max = dp[j];
    }
  }
  dp[i] = Max(arr[i], max + arr[i]);
  Omax = Max(Omax, dp[i]);
}
return Omax;
}
```

TC: O(N*N) O(N)

(68) Min Path Sum: eg 5 → 9 → 6 (11 → 5 → 2) ans = 21

Mimo:

```
dp[][] = new int[grid.length][grid[0].length]
for (int row[] : dp) Arrays.fill(row, -1);
return helper(grid, grid.length-1, grid[0].length-1, dp);
```

```
helper (grid, i, j, dp) {
if (i==0 && j==0) return grid[0][0];
if (i<0 or j<0) return (int) Math.pow(10, 9);
if (dp[i][j]!=-1) return dp[i][j];
int left = grid[i][j] + helper(grid, i, j-1, dp);
up right = grid[i][j] + helper(grid, i-1, j, dp);

return dp[i][j] = min(left, up);
}
```

O(n*m) O(N*M) + O((N-1)+(M-1))

Tabulation :

```
dp[][] = new int[grid.length][grid[0].length];
for (i=0; i<n; i++) {
    for (j=0; j<m; j++) {
        if (i==0 && j==0) dp[i][j] = grid[i][j];
        else {
        up = grid[i][j]
        if (i>0) up += dp[i-1][j];
        else up += 1e9;
        left = grid[i][j];
        if (j>0) left += dp[i][j-1];
        else left += 1e9;

        dp[i][j] = min(left, up);
        }
    }
}
return dp[N-1][M-1];
```

O(N*M) O(N*M)

Date: __/__/__ Page No.: _____

169) Coin change : Count ways in which change can be provided using given denomination.

[1 2 5], t = 4 ans = 4 (1111, 112, 22, 13)

Memo :

```
dp[][] = new int[d.length][v+1];
for(int i[] : dp) Arrays.fill(i, -1);
return helper(d, d.length-1, v, dp);
helper(d, ind, v, dp) {
  if(ind == 0) {
    if(v % d[0] == 0) return 1;
    else return 0;
  }
  if(dp[ind][v] != -1) return dp[ind][v];
  notTake = helper(d, ind-1, v, dp);
  if(d[ind] <= v) {
    take = helper(d, ind, v-d[ind], dp);
  }
return dp[ind][v] = (notTake + take);
  }
```

O(N+M) O(N+M) + O(N)

Tab :

```
for(int i[] : dp) Arrays.fill(i, 0);
for(i=0; i<=v; i++) {
    if(i % d[0] == 0) dp[0][i] = 1;
}

for(ind=1; ind<n; ind++) {
    for(j=0; j<=v; j++) {
```

```
not = dp[ind-1][j];
if (j d[ind] <= j) {
    take = dp[ind][j - d[ind]];
}
dp[ind][j] = (take + not);
}
}
return dp[n-1][v];
```

O(N*M) O(N*M)

170) Subset sum :

eg: 6 1 2 1 k=4 ans = true.

Memo :

```
dp[][] = new Boolean[n][k+1];
for (Boolean row[] : dp) Arrays.fill(row, null);
helper(arr, n-1, k, dp)

helper(arr, ind, k, dp)
if (k == 0) return true;
if (ind == ~~arr[0]~~) return arr[ind] == ~~k~~ k;

if (dp[ind][k] != -1) return dp[ind][k];

notTaken = helper(arr, ind-1, k, dp);
take = false
if (arr[ind] <= k) {
    take = helper(arr, ind-1, k - arr[ind], dp);
}
```

Date: __/__/__ Page No.: ____

return dp[ind][k] = take || notTake;
}

O(N*k) O(N*k) + O(N)

Tab:

Arrays.fill(row, false);
for (i=0; i<n; i++) {
dp[i][0] = true;
}

if (arr[0] <= ~~target~~ k) dp[0][arr[0]] = true.

for (int ind = 1; ind<n; ind++) {
for (int tar=1; tar<=k; tar++) {
not = dp[ind][tar]
take = ~~false~~
if (arr[ind] <= tar) {
take = dp[ind-1][tar - arr[ind]];
}
dp[ind][tar] = take||not;
}}
return dp[n][k-1];

O(N*k) O(N*k)

171) cutting rod (MCM val)
arr [2 5 7 8 10] n = 5
index index means piece piece <= n

Memo :

```
dp[][] = new int dp[n][n+1];
for (row[] : dp) Arrays.fill(row, -1);
return helper(arr, n, n, dp);

helper(price[], int ind, int n, dp[][]) {
if (ind == 0) {
    return price[0] * n;
}
if (dp[ind][n] != -1) return dp[ind][n];
notTake = helper(arr, ind-1, n, dp)
take = 0
rod = ind+1
if (rod <= n) {
        take = helper(arr, ind, n-rod, dp) + arr[ind]
}
return dp[ind][n] = Math.max(take, notTake);
}
```

O(N*M) O(N*N) ~ O(N)

Tat :

```
some dp array and initialization
for (i=0; i<=n; i++) dp[0][i] = arr[0]*i};

for (ind=1; ind<n; ind++) {
    for (j=0; j<=n; j++) {
    notTake = dp[ind-1][j]
    take = 0
    rod = ind+1
    if (rod<=j) {
        take = arr[ind] + dp[ind][j-rod];
    }
```

Date: __/__/__ Page No.: ____

```
dp[j-
    dp[ind][j] = Math.max (take, nottake);
  }
 }
return  dp[ind-1][n];
}
```

O(N*N) O(N*N)