**Dec 2nd**

**Q1 : Understanding Stacks**

| public static void main(String[] args) {    Stack<Integer> st = new Stack<>();  st.push(10);  System.out.println(st + " " + st.peek() + " " + st.size());  st.push(20);  System.out.println(st + " " + st.peek() + " " + st.size());  st.push(30);  System.out.println(st + " " + st.peek() + " " + st.size());  st.push(40);  System.out.println(st + " " + st.peek() + " " + st.size());    st.pop();  System.out.println(st + " " + st.peek() + " " + st.size());  st.pop();  System.out.println(st + " " + st.peek() + " " + st.size());  st.pop();  System.out.println(st + " " + st.peek() + " " + st.size());  st.pop();  *// this will throw exception,  // System.out.println(st + " " + st.peek() + " " + st.size());*  System.out.println(st + " " + st.size());  } |
| --- |

Time complexity of push, pop, peek, size -> O(1)

**Q2 : Extra Brackets**

(<https://course.acciojob.com/idle?question=1375f004-d383-4a7e-9716-e1a5e377a2ec>)

| public boolean ExtraBrackets(String exp)  {  int n = exp.length();  Stack<Character> st = new Stack<>();    for(int i = 0; i < n; i++)  {  char ch = exp.charAt(i);  *// 1. ch is not closing*  if(ch != ')') st.push(ch);  *//2. ch is closing*  else   {  *// Nothing is in between => extra bracket*  if(st.size() > 0 && st.peek() == '(') return true;  else {  *// until you encounter an open keep popping the stack*  while(st.size() > 0 && st.peek() != '(') {  st.pop();  }  st.pop(); *// to pop open bracket*  }    }  }  return false; } |
| --- |

TC : O(2N) => O(N) and SC : O(N)

(every element is it being visited at max 2 times, first while pushing another time while time)

**Q3 : Next Greater Element on Right**

(<https://course.acciojob.com/idle?question=73772158-09d5-4636-aa41-def2d3158102>)

| public static long[] nextLargerElement(long[] arr, int n) {   long[] ans = new long[n];  Stack<Integer> st = new Stack<>(); *// will have indices*  for(int i = 0; i < n; i++) {  *// check whether curr ele is NGE or not*  while(st.size() > 0 && arr[i] > arr[st.peek()]) {  ans[st.peek()] = arr[i]; *// fix / store NGE*  st.pop();  }  st.push(i);  }  *// left over element in stack => no NGE for them*  while(st.size() > 0) {  ans[st.peek()] = -1;  st.pop();  }  return ans; } |
| --- |

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

**Dec 7th**

**Q4 : Balanced Brackets**

(<https://course.acciojob.com/idle?question=ea7fc1c8-be76-4490-8a27-b4c5ff4fa51f>)

| char open(char ch) {  if(ch == ')') return '(';  else if(ch == ']') return '[';  else return '{'; }  public void balancedBrackets(String s, int n)  {  Stack<Character> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = s.charAt(i);  if(ch == '(' || ch == '{' || ch == '[') st.push(ch);  else {  if(st.size() > 0 && st.peek() == open(ch)) st.pop();  else {  System.out.println("NO");  return;  }  }  }  if(st.size() == 0) System.out.println("YES");  else System.out.println("NO"); } |
| --- |

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

**Q5 : Balanced Expression**

(<https://course.acciojob.com/idle?question=e16170b9-480d-4bff-be85-dacd2afc2e48>)

| char open(char ch) {  if(ch == ')') return '(';  else if(ch == ']') return '[';  else return '{'; }  boolean expBalanced(String s)  {  int n = s.length();  Stack<Character> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = s.charAt(i);  if(ch == '(' || ch == '{' || ch == '[') st.push(ch);  else if (ch == ')' || ch == '}' || ch == ']') {  if(st.size() > 0 && st.peek() == open(ch)) st.pop();  else return false;  }  }  return (st.size() == 0); } |
| --- |

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

**Q6 : Previous Greater element**

(<https://course.acciojob.com/idle?question=ac88cc75-d94b-411e-b84d-ca0334811442>)

| public static long[] prevGreater(long[] arr, int n) {   long[] ans = new long[n];  Stack<Integer> st = new Stack<>();  for(int i = n - 1; i >= 0; i--)  {  while(st.size() > 0 && arr[i] > arr[st.peek()])  {  ans[st.peek()] = arr[i];  st.pop();  }  st.push(i);  }    while(st.size() > 0)  {  ans[st.peek()] = -1;  st.pop();  }  return ans;  } |
| --- |

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

**Q7 : Stock Span Problem**

(<https://course.acciojob.com/idle?question=dee87292-2cca-4f9c-9501-973000b81a15>)

| public static int[] nextGreaterOnLeftIdx(int[] arr, int n) {   int[] ans = new int[n];  Stack<Integer> st = new Stack<>();  for(int i = n - 1; i >= 0; i--)  {  while(st.size() > 0 && arr[i] > arr[st.peek()])  {  ans[st.peek()] = i;  st.pop();  }  st.push(i);  }    while(st.size() > 0)  {  ans[st.peek()] = -1;  st.pop();  }  return ans;  }   static int[] stockSpan(int[] a)  {  int[] temp = nextGreaterOnLeftIdx(a, a.length);  int[] ans = new int[a.length];  for(int i = 0; i < a.length; i++)  {  int breakpoint = temp[i];  ans[i] = i - breakpoint;  }  return ans; } |
| --- |

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

**Q8 :** **Largest Histogram Area**

(<https://course.acciojob.com/idle?question=50799402-ffd5-4907-9f91-555993ff4b62>)

| public static long[] nextSmallerOnLeftIdx(long[] arr, int n) {   long[] ans = new long[n];  Stack<Integer> st = new Stack<>();  for(int i = n - 1; i >= 0; i--) {  while(st.size() > 0 && arr[i] < arr[st.peek()]) {  ans[st.peek()] = i; st.pop();  }  st.push(i);  }  while(st.size() > 0) {  ans[st.peek()] = -1; st.pop();  }  return ans; } public static long[] nextSmallerOnRightIdx(long[] arr, int n) {   long[] ans = new long[n];  Stack<Integer> st = new Stack<>();  for(int i = 0; i < n; i++) {  while(st.size() > 0 && arr[i] < arr[st.peek()]) {  ans[st.peek()] = i; st.pop();  }  st.push(i);  }  while(st.size() > 0) {  ans[st.peek()] = n; st.pop();  }  return ans; }  public static long maximumArea(long hist[], long n) {  long[] l = nextSmallerOnLeftIdx(hist, (int)n);  long[] r = nextSmallerOnRightIdx(hist, (int)n);  long ans = 0;  for(int i = 0; i < (int)n; i++) {  long area = hist[i] \* (r[i] - l[i] - 1);  ans = Math.max(ans, area);  }  return ans; } |
| --- |

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

**Q10 : Postfix Evaluation And Conversions**

(<https://course.acciojob.com/idle?question=e508251a-37f6-412c-8d06-7e9219a293f7>)

| int evaluatePostfix(String postfix) {  int n = postfix.length();  Stack<Integer> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = postfix.charAt(i);  *// convert char to int*  if(Character.isDigit(ch)) st.push(ch - '0');  else  {  int op1 = st.pop();  int op2 = st.pop();  if(ch == '+') st.push(op2 + op1);  else if(ch == '-') st.push(op2 - op1);  else if(ch == '\*') st.push(op2 \* op1);  else if(ch == '/') st.push(op2 / op1);  }  }  return st.peek(); }  String postfixToInfix(String postfix) {  int n = postfix.length();  Stack<String> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = postfix.charAt(i);  if(ch == '+' || ch == '-' || ch == '/' || ch == '\*')   {  String op1 = st.pop();  String op2 = st.pop();  String res = "(" + op2 + ch + op1 + ")";  st.push(res);  }  else st.push(ch + ""); *// convert character to string*  }  return st.peek(); } |
| --- |

| String postfixToPrefix(String postfix) {  int n = postfix.length();  Stack<String> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = postfix.charAt(i);  if(ch == '+' || ch == '-' || ch == '/' || ch == '\*')   {  String op1 = st.pop();  String op2 = st.pop();  String res = ch + op2 + op1;  st.push(res);  }  else st.push(ch + ""); *// convert character to string*  }  return st.peek(); }  public void evaluation(String exp) {  System.out.println(evaluatePostfix(exp));  System.out.println(postfixToInfix(exp));  System.out.println(postfixToPrefix(exp)); } |
| --- |

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

**DEC 8th**

**Q11 : Reverse Integer**

(<https://course.acciojob.com/idle?question=b72cbf47-64e8-41e6-b6c7-285988367003>)

| public int reverseInteger(int x) {  int rev = 0;  while (x != 0) {  int pop = x % 10;  x /= 10;  if (rev > Integer.MAX\_VALUE/10 || (rev == Integer.MAX\_VALUE / 10 && pop > 7)) return 0;  if (rev < Integer.MIN\_VALUE/10 || (rev == Integer.MIN\_VALUE / 10 && pop < -8)) return 0;  rev = rev \* 10 + pop;  }  return rev;  } |
| --- |

TC : O(LogN) SC : O(1)

**Q12 : Infix to Postfix**

(<https://course.acciojob.com/idle?question=9c94428f-1965-4a4b-b4be-969a7cbc250e>)

| String infixToPostfix(String exp) {  int n = exp.length();  String result = "";  Stack<Character> st = new Stack<>();  for(int i = 0; i < n; i++)  {  char ch = exp.charAt(i);  // 1. operand => add to output string  if(Character.isLetterOrDigit(ch)) result = result + ch;  //2. open bracket => push to stack  else if(ch == '(') st.push(ch);  //3. close bracket => pop until an open bracket is encountered  else if(ch == ')') {  while(st.size() > 0 && st.peek() != '(') {  result = result + st.peek();  st.pop();  }  st.pop(); // pop the open bracket as well  }  //4. operator => check precedence  else {  while(st.size() > 0 && prec(ch) <= prec(st.peek())) {  result = result + st.peek();  st.pop();  }  st.push(ch);  }  }  // 5. If there are any remaining character pop them as well  while(st.size() > 0) {  result = result + st.peek();  st.pop();  }  return result; } |
| --- |

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

**Q13 : Trapping Rain water**

(<https://course.acciojob.com/idle?question=142ae3a2-073f-4620-b1a2-92b3bbc87710>)

| public void TappingWater(int[] arr, int n)  {  int[] leftmax = new int[n];  int[] rightmax = new int[n];    leftmax[0] = Integer.MIN\_VALUE;  for(int i = 1; i < n; i++) {  leftmax[i] = Math.max(leftmax[i - 1], arr[i - 1]);  }    rightmax[n - 1] = Integer.MIN\_VALUE;  for(int i = n - 2; i >= 0; i--) {  rightmax[i] = Math.max(rightmax[i + 1], arr[i + 1]);  }    int water = 0;  for(int i = 1; i <= n - 2; i++) {  int units = Math.min(leftmax[i], rightmax[i]) - arr[i];  if(units > 0) water += units;  }    System.out.print(water); } |
| --- |

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

**Q14 : Merge Intervals**

(<https://course.acciojob.com/idle?question=2d56d7c3-099b-4480-9311-f182fbab85ad>)

| public void merge(int[][] intervals)  {  // sort based on xth column => Arrays.sort(intervals, (a, b) -> Integer.compare(a[x], b[x]));  // 1. sort based on first values;  Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));  int n = intervals.length;    int prevStart = intervals[0][0];  int prevEnd = intervals[0][1];    for(int i = 1; i < n; i++)  {  int currStart = intervals[i][0];  int currEnd = intervals[i][1];    // overlapping case  if(currStart <= prevEnd) prevEnd = Math.max(prevEnd, currEnd);  else {  System.out.println(prevStart + " " + prevEnd);  prevStart = currStart;  prevEnd = currEnd;  }  }    System.out.println(prevStart + " " + prevEnd); } |
| --- |

TC : O(NlogN) SC : O(1)

**Q15 : Sum of Subarray Minimums**

(<https://course.acciojob.com/idle?question=a11eac7c-f247-409b-851e-7e5bc94bc2ca>)

| public static int[] nextSmallerOnLeftIdx(int[] arr, int n) {   int[] ans = new int[n];  Stack<Integer> st = new Stack<>();     for(int i = n - 1; i >= 0; i--) {  while(st.size() > 0 && arr[i] < arr[st.peek()]) {  ans[st.peek()] = i;  st.pop();  }  st.push(i);  }    while(st.size() > 0) {  ans[st.peek()] = -1;  st.pop();  }  return ans; }  public static int[] nextSmallerOnRightIdx(int[] arr, int n) {   int[] ans = new int[n];  Stack<Integer> st = new Stack<>();     for(int i = 0; i < n; i++) {  while(st.size() > 0 && arr[i] <= arr[st.peek()]) {  ans[st.peek()] = i;  st.pop();  }  st.push(i);  }    while(st.size() > 0) {  ans[st.peek()] = n;  st.pop();  }  return ans; }    public long minSubarraySum(int n, int a[])  {  int[] nsl = nextSmallerOnLeftIdx(a, n);  int[] nsr = nextSmallerOnRightIdx(a, n);  long ans = 0;  long M = 1000000007;    // (a + b) % M  // => ((a % M) + (b % M)) % M    for(int i = 0; i < n; i++) {  long num = (long)(i - nsl[i]) \* (long)(nsr[i] - i);  long temp = (num % M \* a[i] % M) % M;  ans = (ans % M + temp % M) % M;  }    return ans; } |
| --- |

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

**Dec 9th**

**Q14 : Understanding Queue**

| public static void main(String[] args)  {  // 1. Initialize  Queue<Integer> q = new LinkedList<>();    // 2. how to add elements  q.add(10);  q.add(20);  q.add(30);  System.out.println(q);    // 3. get front element  System.out.println(q.peek());    // 4. remove front element  q.remove();  System.out.println(q);    q.remove();  System.out.println(q);    q.add(40);  q.remove();  q.add(50);  q.add(60);    System.out.println(q);  System.out.println(q.size()); } |
| --- |

add, remove, peek, size -> O(1)

**Q15 : Design Stack Using Linked List**

(<https://course.acciojob.com/idle?question=42e0af38-ed64-456b-b7a5-43b885320ffc>)

| class StackUsingLinkedlist {   Node top;  StackUsingLinkedlist() { this.top = null; }   public void push(int x)  {  Node temp = new Node(x);  if(top == null) top = temp;  else {  temp.next = top;  top = temp;  }  }   public int peek()  {  if(top == null) return -1;  return top.data;  }   public void pop()  {  if(top == null) return;  top = top.next;  }   public Node display()  {  return top;  } } |
| --- |

TC : O(1) for all operations SC : O(N)

**Q16 : Queue using Linked List**

(<https://course.acciojob.com/idle?question=b3346122-ef12-4cc2-b8aa-4b1d9fdda3ba>)

| class Node {  int data;  Node next;  Node(int data) {  this.data = data;  this.next = null;  } }  class Queue {  Node front;  Node back;  int cnt = 0;  public void push(int value) {  // 1. Make a new node  Node temp = new Node(value);  // 2. check empty case  if(front == null) {  front = temp;  back = temp;  }  else {  back.next = temp;  back = temp;  }  cnt++;  }    public int pop() {  if(front == null) {  back = null;  return -1;  }    int ans = front.data;  front = front.next;  cnt--;  return ans;  }    public int front() {  if(front == null) return -1;  return front.data;  }    public int getSize() {  return cnt;  } } |
| --- |

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

**Q17 : Implement Queue using stack - enQueue/ push O(1)**

(<https://course.acciojob.com/idle?question=89a5f158-cacc-427d-a317-0967668d8f2b>)

| class StackQueue {  Stack<Integer> s1 = new Stack<>();  Stack<Integer> s2 = new Stack<>();    void Push(int x) {  s1.push(x);  }    int Pop() {  if(s1.size() == 0) return -1;   // 1. move s1 -> s2 until s1.size() = 1  while(s1.size() > 1) s2.push(s1.pop());   // 2. s1 is left with one ele which is front  int ans = s1.peek();  s1.pop();   // 2. move s2 -> s1  while(s2.size() > 0) s1.push(s2.pop());    return ans;  } } |
| --- |

TC : push -> O(1) and pop -> O(N) SC : O(N)

**Q18 : Implement Queue using stack - Dequeue / pop O(1)**

(<https://course.acciojob.com/idle?question=06476864-88f7-478c-bff1-94797c7556b1>)

| class StackQueue {  Stack<Integer> s1 = new Stack<>();  Stack<Integer> s2 = new Stack<>();    void Push(int x)  {  // 1. move s1 -> s2;  while(s1.size() > 0) s2.push(s1.pop());    // 2. add curr ele to s1  s1.push(x);   // 3. move s2 -> s1  while(s2.size() > 0) s1.push(s2.pop());  }    int Pop()  {  if(s1.size() == 0) return -1;  return s1.pop();  } } |
| --- |

TC : push -> O(N) and pop -> O(1) SC : O(N)

**Q19 : Implement two Stacks in an Array**

(<https://course.acciojob.com/idle?question=b47e7025-826e-48d5-ab1c-345bc0a1687b>)

| class twoStacks {  int[] arr;  int size;  int top1, top2;   *// Constructor*  twoStacks(int n) {  size = n;  arr = new int[n];  top1 = -1; *// initialize for s1*  top2 = (n / 2) - 1; *// initialize for s2*  }   void push1(int x) {  *// Overflow condition for s1*  if(top1 == (size / 2) - 1) return;  top1++;  arr[top1] = x;  }   void push2(int x) {  *// Overflow condition for s2*  if(top2 == size - 1) return;  top2++;  arr[top2] = x;  }   void pop1() {  *//Underflow condition for s1*  if(top1 == -1) {  System.out.println(-1);  return;  }  System.out.println(arr[top1]);  top1--;  }  void pop2() {  *//Underflow condition for s2*  if(top2 == (size / 2) - 1) {  System.out.println(-1);  return;  }  System.out.println(arr[top2]);  top2--;  } } |
| --- |

TC : O(1) for all operations SC : O(N)

**Q20 : Circular Tour**

(<https://course.acciojob.com/idle?question=59126924-703f-403a-8af9-821d06e3c75a>)

| boolean check(int start, int petrol[], int distance[]) {  int n = petrol.length;  if(start == n) return false; //edge case    int currpetrol = 0;  int idx = start;  int bunks = 0;  while(bunks < n) {  currpetrol += (petrol[idx] - distance[idx]);  idx = (idx + 1) % n;  bunks++;  }  return (currpetrol >= 0); }  int tour(int petrol[], int distance[])  {  int n = petrol.length;  int start = 0;  int end = 0;  int currpetrol = 0;  while(end < n) {  // 1. consideration -> expansion -> add elements  currpetrol += (petrol[end] - distance[end]);  end++;  // 2. contraction -> poping elements from front  while(currpetrol < 0) {  currpetrol -= (petrol[start] - distance[start]);  start++;  }  }  boolean ans = check(start, petrol, distance);  if(ans == true) return start;  return -1; } |
| --- |

TC : O(N) and SC : O(1) \*\*input is not taken properly in portal please change that

**Q21 : Smallest Number Following Pattern**

(<https://course.acciojob.com/idle?question=efcb1e58-c615-48b1-a7ed-def039965808>)

| public String smallestNumber(String str)  {  Stack<Integer> st = new Stack<>();  String result = "";  int num = 1;  for (int i = 0; i < str.length(); i++)  {  char ch = str.charAt(i);  if (ch == 'd') {  st.push(num);  num++;  } else {  st.push(num);  num++;  while (st.size() > 0) {  result += st.pop();  }  }  }   st.push(num);  while (st.size() > 0) {  result += st.pop();  }  return result; } |
| --- |

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

**Dec 11th**

**Q22 : Backspace String Compare**

(<https://course.acciojob.com/idle?question=25c52021-b22e-4b2a-9183-fa026ba80c8b>)

| class Solution {    public static boolean backspaceCompare(String s, String t) {   int n = s.length();  int m = t.length();  Stack<Character> s1 = new Stack<>();  Stack<Character> s2 = new Stack<>();   for(int i = 0; i < n; i++) {  char ch = s.charAt(i);  if(ch == '#' && s1.size() > 0) s1.pop();  else s1.push(ch);  }   for(int i = 0; i < m; i++) {  char ch = t.charAt(i);  if(ch == '#' && s2.size() > 0) s2.pop();  else s2.push(ch);  }   return s1.equals(s2);  } } |
| --- |

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

**Q23 : Print Bracket Number**(<https://course.acciojob.com/idle?question=b35b8b6f-f94e-4fc1-85a5-34d9e486acd7>)

| class Solution {  ArrayList<Integer> barcketNumbers(String s) {   int n = s.length();  ArrayList<Integer> arr = new ArrayList<>();  Stack<Integer> st = new Stack<>();  int bracketNumber = 1;   for(int i = 0; i < n; i++) {   char ch = s.charAt(i);  if(ch == '(') {  arr.add(bracketNumber);  st.push(bracketNumber);  bracketNumber++;  }  else if(ch == ')' && st.size() > 0) {  arr.add(st.pop());  }  }   return arr;  } } |
| --- |

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

**Q24 : Next Highest Height Left**

(<https://course.acciojob.com/idle?question=2b7faf76-32ba-4e55-9c58-726a91f9861c>)

| class Accio {   int[] nextGreaterOnLeftIdx(int[] arr) {   int n = arr.length;  int[] ans = new int[n];  Stack<Integer> st = new Stack<>();   for(int i = n - 1; i >= 0; i--) {  while(st.size() > 0 && arr[i] > arr[st.peek()]) {  ans[st.peek()] = i;  st.pop();  }  st.push(i);  }   while(st.size() > 0) ans[st.pop()] = -1;   return ans;  }    public int[] solve(int[] arr) {  int n = arr.length;  int[] ngol = nextGreaterOnLeftIdx(arr);  for(int i = 0; i < n; i++) {  if(ngol[i] != -1 )  ngol[i] = i - ngol[i];  }   return ngol;  } } |
| --- |

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

**Q25 : Minimum stack**

(<https://course.acciojob.com/idle?question=5435d3a1-ebd0-4b1c-85f8-d4b600f468b6>)

| class Solution {  Stack<Integer> s1;  Stack<Integer> s2;    Solution()  {  s1 = new Stack<Integer>();  s2 = new Stack<Integer>();  }   void push(int x) {  s1.push(x);  if(s2.size() > 0) s2.push(Math.min(s2.peek(), x));  else s2.push(x);  }    int pop() {  if(s1.size() == 0) return -1;  s2.pop();  return s1.pop();  }   int getMin() {  if(s2.size() == 0) return -1;  return s2.peek();  }  } |
| --- |

TC : O(1) all operations and SC : O(N)

**Q26 : Celebrity Problem**

(<https://course.acciojob.com/idle?question=aa54f234-9dd5-4031-af3d-819afac164f7>)

| class Solution {   int findCelebrity(int M[][], int n) {    Stack<Integer> st = new Stack<>();  for(int i = 0; i < n; i++) st.push(i);   while(st.size() > 1) {  int a = st.pop();  int b = st.pop();  *// a knows b => a cant be celebrity so b might be celebrity hence push(b)*  if(M[a][b] == 1) st.push(b); *// b knows a => b cant be celebrity so a might be celebrity hence push(a)*  else if(M[b][a] == 1) st.push(a);  }   if(st.size() == 0) return -1;  *// Confirm whether top of stack is celebrity*  int ans = st.peek();  for(int i = 0; i < n; i++) {  if(M[ans][i] == 1) return -1;  }    return ans;  } } |
| --- |

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

**Q27 : Valid Parenthesis String**

(<https://course.acciojob.com/idle?question=d77837bf-ee1b-44d5-9c46-9942f3756bd8>)

public static boolean checkValidString(int n, String s)

{

Stack<Integer> open = new Stack<>();

Stack<Integer> star = new Stack<>();

for(int i = 0; i < n; i++) {

char ch = s.charAt(i);

if(ch == '(') open.push(i);

else if(ch == '\*') star.push(i);

else if(ch == ')') {

*// first check open then star, if both are empty we cannot balance*

if(open.size() > 0) open.pop();

else if(star.size() > 0) star.pop();

else return false;

}

}

*// if there are no opens we treat all the left over stars as empty*

if(open.size() == 0) return true;

*// If there are opens but no stars we cannot balance the left over opens*

if(open.size() > 0 && star.size() == 0) return false;

while(open.size() > 0 && star.size() > 0) {

int openIdx = open.pop();

int starIdx = star.pop();

*// if open is coming after a star we cannot balance it so return false*

if(openIdx > starIdx) return false;

}

return true;

}

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

**Q28 : Queue using array**

(<https://course.acciojob.com/idle?question=5e1ce738-3090-4c62-a704-6565f15593d6>)

| class Queue {   int size;  int[] arr;  int front;  int back;  int cnt;    public Queue() {  size = 1000;  arr = new int[size];  front = -1;  back = -1;  cnt = 0;  }   public void push(int newElement) {  // overflow  if(back == size - 1) return;    if(front == -1) {  front = 0;  back = 0;  }  else back++;  arr[back] = newElement;  cnt++;  }  public int pop() {  // underflow  if(front == -1) return -1;   int ans = arr[front];  front++;  cnt--;  if(cnt == 0) {  front = -1;  back = -1;  }  return ans;  }    public int front() {  if(front == -1) return -1;  return arr[front];  }    public int size() {  return cnt;  } } |
| --- |

**Dec 12th**

**Q29 : Sliding window maximum**

(<https://course.acciojob.com/idle?question=2da7ad22-cccc-497d-864c-a3ea784e1263>)

static int[] nextGreaterOnRightIdx(int[] arr) {

int n = arr.length;

Stack<Integer> st = new Stack<>();

int[] ans = new int[n];

for(int i = 0; i < n; i++) {

while(st.size() > 0 && arr[i] > arr[st.peek()])

ans[st.pop()] = i;

st.push(i);

}

// left over ele doesnt have nge so assign nge as extreme right

while(st.size() > 0) ans[st.pop()] = n;

return ans;

}

static int[] SlidingWindowMaximum(int n, int k, int[] arr) {

int[] nge = nextGreaterOnRightIdx(arr);

int[] ans = new int[n - k + 1];

int j = 0;

for(int i = 0; i <= n - k; i++) {

// If j is lagging behind i make them equal

if(j < i) j = i;

// keep jumping j to nge[j] within the window

while(nge[j] < i + k) j = nge[j];

// j will be pointing at your window maximum

ans[i] = arr[j];

}

return ans;

}

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

**Q30 : Asteroid collision**

(<https://leetcode.com/problems/asteroid-collision/>)

public int[] asteroidCollision(int[] arr)

{

Stack<Integer> st = new Stack<>();

int n = arr.length;

for(int i = 0; i < n; i++)

{

if(arr[i] > 0) st.push(arr[i]);

else

{

// pop all asteroids which have less weight

while(st.size() > 0 && st.peek() > 0 && st.peek() < -arr[i])

st.pop();

// check equal weight case

if(st.size() > 0 && st.peek() == -arr[i])

st.pop();

// empty stack and same direction negative weights

else if(st.size() == 0 || st.peek() < 0)

st.push(arr[i]);

}

}

int size = st.size();

int[] ans = new int[size];

for(int i = size - 1; i >= 0; i--)

ans[i] = st.pop();

return ans;

}

// Eg : 8 2 6 -8 5 7 -10 1 3 -4

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

**Q31 : Rotting Oranges**

(<https://course.acciojob.com/idle?question=b21cba45-2a97-4492-82f7-5e23ed20ac00>)

public int orangesRotting(int[][] grid)

{

int rows = grid.length;

int cols = grid[0].length;

// We need a queue which stores 2d indexes

Queue<int[]> rotten = new LinkedList<>();

int fresh = 0;

// Add rotten to queue and count fresh

for(int i = 0; i < rows; i++) {

for(int j = 0; j < cols; j++) {

if(grid[i][j] == 1) fresh++;

else if(grid[i][j] == 2) rotten.add(new int[]{i, j});

}

}

// Edge case if there are no fresh oranges no need to process

if(fresh == 0) return 0;

int[][] dirs = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

int time = 0;

// BFS

while(rotten.size() > 0)

{

// Iterate over currlevel and make adjacent rotten

int size = rotten.size();

for(int i = 0; i < size; i++)

{

int[] indices = rotten.remove();

int r = indices[0];

int c = indices[1];

// (r - 1, c) (r, c + 1) (r + 1, c) (r, c - 1)

for(int j = 0; j < 4; j++)

{

int nr = r + dirs[j][0];

int nc = c + dirs[j][1];

// Make sure nr, nc are in bounds and it is a fresh orange

if(nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == 1) {

fresh--;

grid[nr][nc] = 2;

rotten.add(new int[]{nr, nc});

}

}

}

// before going to next level update time also

time++;

}

if(fresh == 0) return time - 1;

return -1;

}

TC : O(rows \* cols) and SC : O(rows \* cols)

**Dec 13th**

**Q32 : Understanding HashMap**

public static void main(String[] args) {

//1. Initialize

Map<String, Integer> hm = new HashMap<>();

//2. Insert

hm.put("India", 135);

hm.put("China", 200);

hm.put("Pak", 30);

hm.put("US", 20);

hm.put("UK", 10);

System.out.println(hm);

//3. updates if key already present

hm.put("Nigeria", 5);

hm.put("US", 30);

System.out.println(hm);

//4. get value of respective key

System.out.println(hm.get("India"));

System.out.println(hm.get("Utopia"));

//5. check whether key is present or not

System.out.println(hm.containsKey("India"));

System.out.println(hm.containsKey("Utopia"));

//6. get all keys present in hashmap

Set<String> keys = hm.keySet();

System.out.println(keys);

//7. Iterating over hashmap

for(String key : hm.keySet()) {

System.out.println(key + " " + hm.get(key));

}

}

**Q33 : Design HashSet**

(<https://course.acciojob.com/idle?question=86402bd0-eeed-4c05-bf51-6ef08065b6c8>)

class Solution {

int[] arr = new int[1000001];

public void add(int key) {

arr[key] = 1;

}

public void remove(int key) {

arr[key] = 0;

}

public boolean contains(int key) {

return (arr[key] == 1);

}

}

TC : O(1) for all operations and SC : O(106) => O(constant)

**Q34 : First Element to occur k times**

(<https://course.acciojob.com/idle?question=cfcb12b8-f817-4be3-8420-48ea92ed19bc>)

**M1 : Using Count Array**

public void firstElementToOccurKTimes(int[] nums, int n, int k) {

int[] cnt = new int[1000001];

for(int i = 0; i < n; i++) {

cnt[arr[i]]++;

if(cnt[arr[i]] == k) {

System.out.println(arr[i]);

return;

}

}

System.out.println(-1);

}

TC : O(N) and SC : O(106) => O(Constant)

**M2 : Using HashMap**

public void firstElementToOccurKTimes(int[] nums, int n, int k) {

Map<Integer, Integer> hm = new HashMap<>();

for(int i = 0; i < n; i++) {

// adding to hashmap

if(hm.containsKey(nums[i])) {

int oldval = hm.get(nums[i]);

hm.put(nums[i], oldval + 1);

}

else hm.put(nums[i], 1);

// check whether freq is k

if(hm.get(nums[i]) == k) {

System.out.println(nums[i]);

return;

}

}

System.out.println(-1);

}

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

**Q35 : Missing Numbers**

(<https://course.acciojob.com/idle?question=560ab8d1-ed6f-45e0-b3be-a0d1c1d56499>)

static void missingNumbers(int n, int arr[], int m, int brr[]) {

int[] cnt1 = new int[10001];

int[] cnt2 = new int[10001];

for(int i = 0; i < n; i++) cnt1[arr[i]]++;

for(int i = 0; i < m; i++) cnt2[brr[i]]++;

boolean found = false;

// ele => [1, 10000] is missing or not

for(int ele = 1; ele <= 10000; ele++) {

// present in 2nd arr but not in 1st

if(cnt2[ele] > 0 && cnt1[ele] == 0) {

System.out.print(ele + " ");

found = true;

}

// present in both but of different frequency

else if(cnt1[ele] > 0 && cnt2[ele] > 0 && cnt1[ele] != cnt2[ele]) {

System.out.print(ele + " ");

found = true;

}

}

if(found == false) System.out.print(-1);

}

TC : O(N + M + 104) => O(N + M) and SC : O(104 + 104) => O(Constant)

**Q36 : Employees and Manager**

(<https://course.acciojob.com/idle?question=7d2ada34-6296-40ca-8bbf-389f729ac8c5>)

int solve(String ceo, Map<String, List<String>> mngr, Map<String, Integer> ans) {

if(mngr.containsKey(ceo) == false) {

ans.put(ceo, 0);

return 1;

}

int cnt = 0;

// Cnt all the employees under ceo

for(String emp : mngr.get(ceo)) {

cnt += solve(emp, mngr, ans);

}

ans.put(ceo, cnt);

return cnt + 1;

}

public void EmpUnderManager(Map<String, String> emp) {

//1. manager -> emp

Map<String, List<String>> mngr = new HashMap<>();

String ceo = "";

for(String employee : emp.keySet()) {

String manager = emp.get(employee);

if(manager.equals(employee)) {

ceo = manager;

continue;

}

// If containsKey update the old list only else make a new list

if(mngr.containsKey(manager)) {

List<String> oldlist = mngr.get(manager);

oldlist.add(employee);

mngr.put(manager, oldlist);

}

else {

List<String> newlist = new ArrayList<>();

newlist.add(employee);

mngr.put(manager, newlist);

}

}

// As we need keys in sorted order

Map<String, Integer> ans = new TreeMap<>();

solve(ceo, mngr, ans);

for(String key : ans.keySet()) {

System.out.println(key + " " + ans.get(key));

}

}

TC : O(Total no.of employees) and SC : O(Total no.of employees)

**Dec 14th**

**Q37 : Problem with given difference**

(<https://course.acciojob.com/idle?question=803b4abc-3829-4b3b-9dab-74da720ff06a>)

public int givenDifference(int []arr, int n, int k)

{

Map<Integer, Integer> hm = new HashMap<>();

for(int i = 0; i < n; i++)

{

if(hm.containsKey(arr[i] + k) || hm.containsKey(arr[i] - k))

return 1;

int oldval = hm.getOrDefault(arr[i], 0);

hm.put(arr[i], oldval + 1);

}

return 0;

}

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

**Q38 : Pair Sum Divisible by K**

(<https://course.acciojob.com/idle?question=0031d548-b5e9-488d-a254-9a9a3536319a>)

public static int countKdivPairs(int arr[], int n, int k)

{

Map<Integer, Integer> hm = new HashMap<>();

int cnt = 0;

for(int i = 0; i < n; i++)

{

int rem = arr[i] % k;

if(rem == 0) {

int val = hm.getOrDefault(rem, 0);

cnt += val;

}

else {

int val = hm.getOrDefault(k - rem, 0);

cnt += val;

}

int oldval = hm.getOrDefault(rem, 0);

hm.put(rem, oldval + 1);

}

return cnt;

}

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

**Q39 : Equilibrium Index**

(<https://course.acciojob.com/idle?question=ca688309-71a6-4c7a-8a45-07ad8817a350>)

static int findEquilibriumIndex(int[] arr)

{

int n = arr.length;

int total = 0;

for(int i = 0; i < n; i++)

total += arr[i];

int leftsum = 0;

for(int i = 0; i < n; i++) {

int rightsum = total - leftsum - arr[i];

if(leftsum == rightsum) return i;

leftsum += arr[i];

}

return -1;

}

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

**Q40 : Largest subarray with 0 sum**

(<https://course.acciojob.com/idle?question=2ee2a709-fb2f-4acd-b328-a7a74a556edb>)

public int maxLen(int arr[])

{

int n = arr.length;

Map<Integer, Integer> hm = new HashMap<>(); // (sum, idx)

int sum = 0;

int maxlen = 0;

hm.put(0, -1); // to handle for continous sum = 0

for(int i = 0; i < n; i++)

{

sum += arr[i];

// 1. check for sum in hm

if(hm.containsKey(sum)) {

int len = i - hm.get(sum);

maxlen = Math.max(len, maxlen);

}

// 2. only put for first time so as to get largest (more gap)

else hm.put(sum, i);

}

return maxlen;

}

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

**Q41 : Subarray Sum Equals K**

(<https://leetcode.com/problems/subarray-sum-equals-k/description/>)

public int subarraySum(int[] nums, int k)

{

int n = nums.length;

int sum = 0;

int cnt = 0;

Map<Integer, Integer> hm = new HashMap<>(); // (sum, freq)

hm.put(0, 1); // if continuos sum == k

for(int i = 0; i < n; i++)

{

sum += nums[i];

if(hm.containsKey(sum - k)) {

cnt += hm.get(sum - k);

}

int oldval = hm.getOrDefault(sum, 0);

hm.put(sum, oldval + 1);

}

return cnt;

}

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

**Dec 15th**

**Q42 : Subarray Sum Divisible by k**

(<https://course.acciojob.com/idle?question=6b0355db-2e09-4afa-8be4-045d710113fb>)

public int subarraysDivByK(int[] nums, int k)

{

int n = nums.length;

Map<Integer, Integer> hm = new HashMap<>();

int sum = 0;

int cnt = 0;

hm.put(0, 1); // to handle continuous sum % k == 0

for(int i = 0; i < n; i++)

{

sum += nums[i];

int rem = sum % k;

if(rem < 0) rem = rem + k;

int val = hm.getOrDefault(rem, 0);

cnt += val;

hm.put(rem, val + 1);

}

return cnt;

}

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

**Q43 : Group Anagrams**

(<https://leetcode.com/problems/group-anagrams/>)

String sortStr(String s) {

// convert string to arr

char[] arr = s.toCharArray();

// sort the array

Arrays.sort(arr);

// convert arr to string

return new String(arr);

}

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> hm = new HashMap<>();

int n = strs.length;

for(int i = 0; i < n; i++) {

String original = strs[i];

String sorted = sortStr(original);

if(hm.containsKey(sorted)) {

List<String>oldlist = hm.get(sorted);

oldlist.add(original);

hm.put(sorted, oldlist);

}

else {

List<String>newlist = new ArrayList<>();

newlist.add(original);

hm.put(sorted, newlist);

}

}

List<List<String>> ans = new ArrayList<>();

for(String key : hm.keySet()) {

ans.add(hm.get(key));

}

return ans;

}

TC : O(N\*MLogM) and SC : O(N) [ N -> strs.length, M -> strs[i].length()]

**Q44 : Substring With K Unique characters**

(<https://course.acciojob.com/idle?question=10944e43-a4d3-4974-9ea9-02aa61d602ee>)

public static int longestkSubstr(int n, int k, String s)

{

int start = 0;

int end = 0;

int unique = 0;

int ans = -1;

int[] freq = new int[123];

// ('a' -> 'z') [97, 122]

while(end < n) {

// 1. expansion

freq[s.charAt(end)]++;

if(freq[s.charAt(end)] == 1) unique++;

end++;

// 2. contraction

while(start < end && unique > k) {

freq[s.charAt(start)]--;

if(freq[s.charAt(start)] == 0) unique--;

start++;

}

// 3. calculation

if(unique == k)

ans = Math.max(ans, end - start);

}

return ans;

}

TC : O(N) and SC : O(123) => O(Constant)

**Q45 : Distinct Window**

(<https://course.acciojob.com/idle?question=9cc8f33f-8879-4406-8eee-193ccc59fbac>)

public static String smallestkSubstr(int n, int k, String s)

{

int start = 0;

int end = 0;

int unique = 0;

int ans = Integer.MAX\_VALUE;

int[] freq = new int[123];

int ansStart = -1;

int ansEnd = -1;

while(end < n) {

// 1. expansion

freq[s.charAt(end)]++;

if(freq[s.charAt(end)] == 1) unique++;

end++;

// 2. contraction

while(start < end && unique == k) {

// 3. calculation

if(ans > end - start) {

ans = end - start;

ansStart = start;

ansEnd = end;

}

freq[s.charAt(start)]--;

if(freq[s.charAt(start)] == 0) unique--;

start++;

}

}

if(ansStart == -1) return "";

return s.substring(ansStart, ansEnd);

}

public static String DistinctWindow(String s)

{

int n = s.length();

int[] cnt = new int[123];

int distinct = 0;

for(int i = 0; i < n; i++) {

cnt[s.charAt(i)]++;

if(cnt[s.charAt(i)] == 1) distinct++;

}

return smallestkSubstr(n, distinct, s);

}

TC : O(N) and SC : O(123) => O(Constant)

**Q46 : Minimum Window Substring**

(<https://course.acciojob.com/idle?question=8c817174-62b5-46e8-8cf7-00cc0d0ffa47>)

boolean isSatisfied(int[] sfreq, int[] tfreq) {

for(int i = 0; i < 123; i++) {

if(tfreq[i] > sfreq[i]) return false;

}

return true;

}

public String minWindow(String s, String t) {

int n = s.length(), m = t.length();

int[] tfreq = new int[123];

for(int i = 0; i < m; i++) tfreq[t.charAt(i)]++;

int[] sfreq = new int[123];

int start = 0, end = 0;

int ans = Integer.MAX\_VALUE;

int ansStart = -1, ansEnd = -1;

while(end < n) {

sfreq[s.charAt(end)]++;

end++;

while(start < end && isSatisfied(sfreq, tfreq)) {

if(ans > end - start) {

ans = end - start;

ansStart = start;

ansEnd = end;

}

sfreq[s.charAt(start)]--;

start++;

}

}

if(ansStart == -1) return "";

return s.substring(ansStart, ansEnd);

}

TC : (123 \* N) and SC : O(123) => O(Constant)

**Dec 16th**

**Q47 : Longest subarray with equal frequency**

(<https://course.acciojob.com/idle?question=617c656f-342f-4dd1-b9e0-5c1469fad4a7>)

\*\*Try this as well => (<https://practice.geeksforgeeks.org/problems/equal-0-1-and-23208/1>)

public static int longestSubarray(int[] arr)

{

int n = arr.length;

Map<String, Integer> hm = new HashMap<>();

int zeros = 0, ones = 0, twos = 0;

int ans = 0;

hm.put("0#0", -1);

for(int i = 0; i < n; i++)

{

if(arr[i] == 0) zeros++;

else if(arr[i] == 1) ones++;

else twos++;

int diff10 = ones - zeros;

int diff21 = twos - ones;

String key = diff10 + "#" + diff21;

if(hm.containsKey(key))

ans = Math.max(ans, i - hm.get(key));

else hm.put(key, i);

}

return ans;

}

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

**Q48 : LRU Cache**

(<https://leetcode.com/problems/lru-cache/>)

class Node {

int appName;

int state;

Node next;

Node prev;

Node(int appName, int state) {

this.appName = appName;

this.state = state;

next = prev = null;

}

}

class LRUCache {

Node front, back; // (MRU, LRU)

int size, capacity; // (length, max Recent pages)

Map<Integer, Node> hm; // (appName - address)

public LRUCache(int capacity) {

this.capacity = capacity;

this.size = 0;

front = back = null;

hm = new HashMap<>();

}

public int get(int appName) {

//See whether it is present in recent apps

if(hm.containsKey(appName)) {

// open the app i.e; bring to front

Node app = hm.get(appName);

moveTofront(app);

return app.state;

}

return -1;

}

private void moveTofront(Node app) {

// 1. If app is already at front

if(app == front) return;

// 2. If app is at back

if(app == back) back = back.next;

Node nextApp = app.next;

Node prevApp = app.prev;

if(nextApp != null) nextApp.prev = prevApp;

if(prevApp != null) prevApp.next = nextApp;

front.next = app;

app.prev = front;

app.next = null;

front = app;

}

private void addAtFront(Node app) {

if(front == null) {

front = back = app;

return;

}

front.next = app;

app.prev = front;

app.next = null;

front = app;

}

private void removeBack() {

if(back == null) return;

int backAppName = back.appName;

Node nextApp = back.next;

if(nextApp != null) nextApp.prev = null;

back.next = null;

back = nextApp;

}

public void put(int appName, int state) {

if(hm.containsKey(appName)) {

Node app = hm.get(appName);

moveTofront(app);

app.state = state;

}

else {

Node app = new Node(appName, state);

hm.put(appName, app);

addAtFront(app);

size++;

}

// if your recent apps > given capacity

if(size > capacity) {

hm.remove(back.appName);

removeBack();

size--;

}

}

}

TC : O(1) for all operations and SC : O(Capacity)

**Q49 : Longest Subsequence With Difference One**

(<https://course.acciojob.com/idle?question=1ca1c62e-19c2-4ef6-9f62-0fb652b756dc>)

public int longestSubsequence(int[] arr)

{

HashMap<Integer, Integer> hm = new HashMap<>();

int ans = 1;

int n = arr.length;

for(int i = 0; i < n; i++)

{

int len = 1;

if(hm.containsKey(arr[i] + 1))

len = Math.max(len, hm.get(arr[i] + 1) + 1);

if(hm.containsKey(arr[i] - 1))

len = Math.max(len, hm.get(arr[i] - 1) + 1);

hm.put(arr[i], len);

ans = Math.max(ans, len);

}

return ans;

}

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

**Q50 : Count Number of Pairs With Absolute Difference K**

(<https://course.acciojob.com/idle?question=c47351e9-e120-488d-a193-0fdc5ab7a56b>)

public int findPairs(int[] A, int k)

{

int n = A.length;

HashMap<Integer, Integer> hm = new HashMap<>();

int ans = 0;

// build your Freq Map

for(int i = 0; i < n; i++) {

hm.put(A[i], hm.getOrDefault(A[i], 0) + 1);

}

// iterate on key set so that it doesnt have any dups

// handle edge case when k = 0

for(Integer a : hm.keySet()) {

if(k == 0) {

if(hm.get(a) > 1) ans++;

}

else {

if(hm.containsKey(a + k) && hm.get(a + k) > 0) ans++;

if(hm.containsKey(a - k) && hm.get(a - k) > 0) ans++;

}

// as the number is used and it should not be used for further pairs

hm.put(a, 0);

}

return ans;

}

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

**Dec 17th**

**Q51 : Avoid Flood in the city**

(<https://leetcode.com/problems/avoid-flood-in-the-city/>)

private int search(int[] rains, int start, int end) {

// return first day between [start, end]

for(int i = start; i <= end; i++) {

if(rains[i] == 0) return i;

}

return -1;

}

private int searchOPT(TreeMap<Integer, Integer> thm, int start) {

// ceil gives just greater or equal

Integer dryIdx = thm.ceilingKey(start);

if(dryIdx == null) return -1;

return dryIdx;

}

public int[] avoidFlood(int[] rains)

{

int n = rains.length;

Map<Integer, Integer> hm = new HashMap<>();

TreeMap<Integer, Integer> thm = new TreeMap<>();

int[] ans = new int[n];

for(int i = 0; i < n; i++) {

int lakeNo = rains[i];

if(lakeNo > 0) {

// lake is previously filled

if(hm.containsKey(lakeNo)) {

// day when lakeNo is previously filled

int prevIdx = hm.get(lakeNo);

// check for dry day in between

int dryIdx = searchOPT(thm, prevIdx + 1);

if(dryIdx != -1) {

// at dry day current lake will be dried

ans[dryIdx] = lakeNo;

// at curr day it will rain because we dried it

ans[i] = -1;

// as we used that dry day mark it used and remove

rains[dryIdx] = Integer.MIN\_VALUE;

thm.remove(dryIdx);

}

// if no chance to dry then flood

else return new int[]{};

}

// lake is not present in hm => lake is not filled

else ans[i] = -1;

hm.put(lakeNo, i); // update (lakeNo, dayFilled)

}

else thm.put(i, 1); // store dry days in tree map

}

// Any leftover dry days make them 1

for(int i = 0; i < n; i++) {

if(rains[i] == 0) ans[i] = 1;

}

return ans;

}

TC : O(NLogN) and SC : O(N)

**Q52 : Rabbits in a forest**

(<https://leetcode.com/problems/rabbits-in-forest/>)

public int numRabbits(int[] answers)

{

int n = answers.length;

int[] cnt = new int[1001];

for(int i = 0; i < n; i++)

cnt[answers[i]]++;

int ans = 0;

for(int num = 0; num <= 1000; num++)

{

int freq = cnt[num];

int groups = (int)Math.ceil((freq \* 1.0)/ (num + 1));

ans += (groups \* (num + 1));

}

return ans;

}

TC : O(N) and SC : O(1001) => O(Constant)

**Q53 : Longest substring without repeating characters**

(<https://leetcode.com/problems/longest-substring-without-repeating-characters/>)

boolean isRepeated(int[] freq)

{

for(int i = 0; i < 256; i++) {

if(freq[i] > 1) return true;

}

return false;

}

public int lengthOfLongestSubstring(String s)

{

int n = s.length();

int[] freq = new int[256];

int start = 0, end = 0;

int ans = 0;

while(end < n)

{

freq[s.charAt(end)]++;

end++;

while(isRepeated(freq))

{

freq[s.charAt(start)]--;

start++;

}

ans = Math.max(ans, end - start);

}

return ans;

}

TC : O(256 \* N) and SC : O(256) => O(Constant)

**M2 : Using Repeat variable**

public int lengthOfLongestSubstring(String s)

{

int n = s.length();

int[] freq = new int[256];

int start = 0, end = 0;

int ans = 0, repeat = 0;

while(end < n)

{

freq[s.charAt(end)]++;

if(freq[s.charAt(end)] > 1) repeat++;

end++;

while(start < end && repeat > 0)

{

if(freq[s.charAt(start)] > 1) repeat--;

freq[s.charAt(start)]--;

start++;

}

ans = Math.max(ans, end - start);

}

return ans;

}

TC : O(N) and SC : O(256) => O(Constant)

**Dec 18th**

**Q54 : Minimum Size Subarray Sum**

(<https://leetcode.com/problems/minimum-size-subarray-sum/>)

public int minSubArrayLen(int target, int[] nums)

{

int n = nums.length;

int start = 0;

int end = 0;

int sum = 0;

int ans = Integer.MAX\_VALUE;

while(end < n) {

sum += nums[end];

end++;

while(start < end && sum >= target) {

ans = Math.min(ans, end - start);

sum -= nums[start];

start++;

}

}

if(ans == Integer.MAX\_VALUE) return 0;

return ans;

}

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

**Q55 : Shortest Subarray with Sum at Least K**

(<https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/>)

public int shortestSubarray(int[] nums, int k) {

int n = nums.length;

long sum = 0;

Deque<long[]> dq = new LinkedList<>(); // (idx, sum)

int ans = Integer.MAX\_VALUE;

for(int i = 0; i < n; i++)

{

sum += nums[i]; // s2

// Maintain sum in deque as increasing order

while(dq.size() > 0 && dq.peekLast()[1] >= sum) {

dq.removeLast();

}

dq.addLast(new long[]{i, sum});

if(sum >= k) ans = Math.min(ans, i + 1); // check s2 >= k

// check any possible s2 - s1 >= k

while(dq.size() > 0 && sum - dq.peekFirst()[1] >= k) {

ans = Math.min(ans, i - (int)dq.peekFirst()[0]);

dq.removeFirst();

}

}

if(ans == Integer.MAX\_VALUE) return -1;

return ans;

}

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

**Q56 : Understanding Binary search**

(<https://practice.geeksforgeeks.org/problems/binary-search-1587115620/1>)

**M1 : Iterative**

int binarysearch(int arr[], int n, int ele)

{

int start = 0;

int end = n - 1;

while(start <= end) {

int mid = (start + end) / 2;

if(ele == arr[mid]) return mid;

else if(ele > arr[mid]) start = mid + 1;

else end = mid - 1;

}

return -1;

}

TC : O(Log2N) and SC : O(1)

**M2 : Recursive**

int bs(int arr[], int start, int end, int ele)

{

if(start > end) return -1;

int mid = (start + end) / 2;

if(ele == arr[mid]) return mid;

else if(ele > arr[mid]) return bs(arr, mid + 1, end, ele);

return bs(arr, start, mid - 1, ele);

}

int binarysearch(int arr[], int n, int ele) {

return bs(arr, 0, n - 1, ele);

}

TC : O(Log2N) and SC : O(Log2N) [Recursive Stack Space]

**Q57 : Floor and Ceil in a sorted array**

(<https://course.acciojob.com/idle?question=127fb51b-1c5d-4e3a-a2db-7e6ae72fe535>)

**M1 : Using Ans variable**

private static int floor(int[] arr, int ele) {

int start = 0;

int end = arr.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) return arr[mid];

else if(arr[mid] < ele) {

ans = arr[mid];

start = mid + 1;

}

else end = mid - 1;

}

return ans;

}

private static int ceil(int[] arr, int ele) {

int start = 0;

int end = arr.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) return arr[mid];

else if(arr[mid] > ele) {

ans = arr[mid];

end = mid - 1;

}

else start = mid + 1;

}

return ans;

}

public static int[] floorAndCeil(int ele, int[] arr) {

int f = floor(arr, ele);

int c = ceil(arr, ele);

return new int[]{f, c};

}

**M2 : Using start and end**

public static int[] floorAndCeil(int ele, int[] arr)

{

int n = arr.length;

int start = 0;

int end = n - 1;

if(arr[end] < ele) return new int[]{arr[end], -1};

else if(ele < arr[start]) return new int[]{-1, arr[start]};

while(start <= end)

{

int mid = (start + end) / 2;

if(ele == arr[mid]) return new int[]{ele, ele};

else if(ele > arr[mid]) start = mid + 1;

else end = mid - 1;

}

return new int[]{arr[end], arr[start]};

}

TC : O(Log2N) and SC : O(1)

**Q58 : Count 1 in sorted binary array**

(<https://course.acciojob.com/idle?question=98e9bbba-6f59-4585-a38c-6f9bd3cd972a>)

private static int firstZero(int arr[], int n) {

int start = 0;

int end = n - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == 1) start = mid + 1;

else {

ans = mid;

end = mid - 1;

}

}

return ans;

}

private static int lastOne(int arr[], int n) {

int start = 0;

int end = n - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == 1) {

ans = mid;

start = mid + 1;

}

else end = mid - 1;

}

return ans;

}

static int count1(int size, int arr[]) {

/\*int idx = firstZero(arr, size);

if(idx == -1) return size;

return idx;\*/

int idx = lastOne(arr, size);

return idx + 1;

}

TC : O(Log2N) and SC : O(1)

**Q59 : Sorted Insert Position**

(<https://course.acciojob.com/idle?question=bff80545-9861-4ac9-9c16-e4729299bd09>)

private static int ceil(int[] arr, int ele) {

int start = 0;

int end = arr.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) return mid;

else if(arr[mid] > ele) {

ans = mid;

end = mid - 1;

}

else start = mid + 1;

}

return ans;

}

public static int searchInsert(int[] a, int b) {

int idx = ceil(a, b);

if(idx == -1) return a.length;

return idx;

}

TC : O(Log2N) and SC : O(1)

**Dec 20th**

**Q60 : Find First and Last Position of Element in Sorted Array**

(<https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/>)

private int firstOccur(int[] arr, int ele) {

int start = 0;

int end = arr.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) {

ans = mid;

end = mid - 1;

}

else if(ele > arr[mid]) start = mid + 1;

else end = mid - 1;

}

return ans;

}

private int lastOccur(int[] arr, int ele) {

int start = 0;

int end = arr.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) {

ans = mid;

start = mid + 1;

}

else if(ele > arr[mid]) start = mid + 1;

else end = mid - 1;

}

return ans;

}

public int[] searchRange(int[] nums, int target) {

int first = firstOccur(nums, target);

if(first == -1) return new int[]{-1, -1};

int last = lastOccur(nums, target);

return new int[]{first, last};

}

TC : O(LogN) and SC : O(1)

**Q61 : Search a 2D Matrix**

(<https://leetcode.com/problems/search-a-2d-matrix/>)

private int floor(int[][] matrix, int ele) {

int start = 0;

int end = matrix.length - 1;

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(matrix[mid][0] == ele) return mid;

else if(matrix[mid][0] < ele) {

ans = mid;

start = mid + 1;

}

else end = mid - 1;

}

return ans;

}

private boolean binarySearch(int[][] matrix, int target, int row) {

int start = 0;

int end = matrix[row].length - 1;

while(start <= end) {

int mid = (start + end) / 2;

if(matrix[row][mid] == target) return true;

else if(target > matrix[row][mid]) start = mid + 1;

else end = mid - 1;

}

return false;

}

public boolean searchMatrix(int[][] matrix, int target) {

int row = floor(matrix, target); // to find correct row O(Log rows)

if(row == -1) return false;

if(matrix[row][0] == target) return true;

// perform bs on that row O(Log cols)

return binarySearch(matrix, target, row);

}

TC : O(Log rows + Log cols) and SC : O(1)

**M2 : Virtual 1D**

public boolean searchMatrix(int[][] matrix, int ele) {

int rows = matrix.length;

int cols = matrix[0].length;

int start = 0;

int end = (rows \* cols) - 1;

while(start <= end) {

int mid = (start + end) / 2;

int r = mid / cols;

int c = mid % cols;

if(matrix[r][c] == ele) return true;

else if(ele > matrix[r][c]) start = mid + 1;

else end = mid - 1;

}

return false;

}

TC : O(Log (rows \* cols)) => O(Log rows + Log cols) and SC : O(1)

**Q62 : Search in Rotated Sorted Array**

(<https://leetcode.com/problems/search-in-rotated-sorted-array/>)

**M1 : Finding max element and then BS**

private int maxIdx(int[] arr) {

int n = arr.length;

int start = 0;

int end = n - 1;

while(start <= end) {

int mid = (start + end) / 2;

if(mid + 1 < n && arr[mid] > arr[mid + 1]) return mid;

else if(arr[start] <= arr[mid]) start = mid + 1;

else end = mid - 1;

}

return n - 1;

}

private int binarySearch(int[] arr, int ele, int start, int end) {

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == ele) return mid;

else if(ele > arr[mid]) start = mid + 1;

else end = mid - 1;

}

return -1;

}

public int search(int[] arr, int target) {

int idx = maxIdx(arr);

int n = arr.length;

if(arr[idx] == target) return idx;

else if(arr[0] <= target && target < arr[idx])

return binarySearch(arr, target, 0, idx);

return binarySearch(arr, target, idx + 1, n - 1);

}

**M2 : Single Binary search based on graph**

public int search(int[] arr, int target) {

int start = 0;

int end = arr.length - 1;

while(start <= end) {

int mid = (start + end) / 2;

if(arr[mid] == target) return mid;

// 1st line

else if(arr[start] <= arr[mid]) {

// [start, mid] is sorted

if(arr[start] <= target && target < arr[mid]) end = mid - 1;

else start = mid + 1;

}

// 2nd line

else {

// [mid, end] is sorted

if(arr[mid] < target && target <= arr[end]) start = mid + 1;

else end = mid - 1;

}

}

return -1;

}

TC : O(LogN) and SC : O(1) for both methods

**Q63 : Snapshot Array (Doubt)**

(<https://leetcode.com/problems/snapshot-array/>)

class SnapshotArray {

TreeMap<Integer, Integer> arr[];

int snap;

public SnapshotArray(int length) {

arr = new TreeMap[length];

for(int i = 0; i < length; i++) {

arr[i] = new TreeMap<>();

}

snap = 0;

}

public void set(int index, int val) {

arr[index].put(snap, val);

}

public int snap() {

snap++;

return snap - 1;

}

public int get(int index, int snap\_id) {

Integer floor = arr[index].floorKey(snap\_id);

if(floor == null) return 0;

return arr[index].get(floor);

}

}

TC : O(LogS) for set and get, O(1) for snap

SC : O(S)

Here S = Total no.of snaps

**Dec 21st**

**Q64 : Square root** (<https://leetcode.com/problems/sqrtx/>)

(<https://course.acciojob.com/idle?question=71891482-69b9-4bd3-a1ae-1945179ee04f>)

public int mySqrt(int x) {

long start = 0;

long end = x;

long ans = 0;

while(start <= end) {

long mid = (start + end) / 2;

if(mid \* mid == x) return (int)mid;

else if(mid \* mid < x) {

ans = mid;

start = mid + 1;

}

else end = mid - 1;

}

return (int)ans;

}

**M2 : Without using long**

public int mySqrt(int x) {

if(x == 0) return 0;

int start = 1;

int end = x;

int ans = 0;

while(start <= end) {

int mid = start + ((end - start) / 2);

if(mid == x / mid) return mid;

else if(mid < x / mid) {

ans = mid;

start = mid + 1;

}

else end = mid - 1;

}

return ans;

}

**Q65 : Triangular Number**

(<https://course.acciojob.com/idle?question=c73a5b20-1518-4498-8222-bce4f230b463>)

static boolean TriangularNumber(int N){

long start = 1;

long end = N;

while(start <= end) {

long mid = (start + end) / 2;

long num = (mid \* (mid + 1)) / 2;

if(num == N) return true;

else if(num > N) end = mid - 1;

else start = mid + 1;

}

return false;

}

TC : O(LogN) and SC : O(1)

**Q66 : Minimum Number of Days to Make m Bouquets**

(<https://leetcode.com/problems/minimum-number-of-days-to-make-m-bouquets/>)

(<https://course.acciojob.com/idle?question=d203d755-cbed-4c79-910b-d77dec37f33b>)

private boolean isPossible(int[] bloomDay, int currDay, int m, int k) {

int flowers = 0;

int boques = 0;

int n = bloomDay.length;

for(int i = 0; i < n; i++) {

if(bloomDay[i] <= currDay) flowers++; // consecutive flowers

else flowers = 0;

if(flowers == k) {

flowers = 0;

boques++;

if(boques == m) return true;

}

}

return false;

}

public int minDays(int[] bloomDay, int m, int k) {

int minDay = Integer.MAX\_VALUE;

int maxDay = Integer.MIN\_VALUE;

int n = bloomDay.length;

// Flowers are not enough for making a boque

if(m \* k > n) return -1;

for(int i = 0; i < n; i++) {

minDay = Math.min(minDay, bloomDay[i]);

maxDay = Math.max(maxDay, bloomDay[i]);

}

int start = minDay; // (trick) start = 0

int end = maxDay; // end = (int)1e9

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

if(isPossible(bloomDay, mid, m, k)) {

ans = mid;

end = mid - 1;

}

else start = mid + 1;

}

return ans;

}

TC : O(NLog(maxEle - minEle)) and SC : O(1)

**Q67 : Capacity To Ship Packages Within D Days**

(<https://leetcode.com/problems/capacity-to-ship-packages-within-d-days/>)

(<https://course.acciojob.com/idle?question=ea8f08aa-d041-4639-a8c6-d3235d8bd34f>)

private boolean isPossible(int[] weights, int truckWeight, int days) {

int n = weights.length;

int currWeight = 0;

int currDay = 1;

for(int i = 0; i < n; i++) {

// if you use trick, handle edge case

// if(weights[i] > truckWeight) return false;

currWeight += weights[i];

// currWeight > truckWeight move to next day

if(currWeight > truckWeight) {

currWeight = weights[i];

currDay++;

}

}

return (currDay <= days);

}

public int shipWithinDays(int[] weights, int days) {

int maxLoad = Integer.MIN\_VALUE;

int totalLoad = 0;

int n = weights.length;

for(int i = 0; i < n; i++) {

maxLoad = Math.max(maxLoad, weights[i]);

totalLoad += weights[i];

}

int start = maxLoad; // (trick) start = 0;

int end = totalLoad; // end = (int)1e9 = 10 power 9

int ans = -1;

while(start <= end) {

int mid = (start + end) / 2;

// Transfer (weights) using (mid) kg truck in (days)

if(isPossible(weights, mid, days)) {

ans = mid;

end = mid - 1;

}

else start = mid + 1;

}

return ans;

}

TC : O(NLog(sum - maxEle)) and SC : O(1)

**Dec 22nd**

**Q68 : Preorder, Inorder, Postorder**

([Binary Tree Inorder Traversal - LeetCode](https://leetcode.com/problems/binary-tree-inorder-traversal/))

([Binary Tree Preorder Traversal - LeetCode](https://leetcode.com/problems/binary-tree-preorder-traversal/))

([Binary Tree Postorder Traversal - LeetCode](https://leetcode.com/problems/binary-tree-postorder-traversal/))

class Solution {

List<Integer> ans = new ArrayList<>();

private void inorder(TreeNode root) {

if(root == null) return;

inorder(root.left);

ans.add(root.val);

inorder(root.right);

}

public List<Integer> inorderTraversal(TreeNode root) {

inorder(root);

return ans;

}

}

class Solution {

List<Integer> ans = new ArrayList<>();

private void preorder(TreeNode root) {

if(root == null) return;

ans.add(root.val);

preorder(root.left);

preorder(root.right);

}

public List<Integer> preorderTraversal(TreeNode root) {

preorder(root);

return ans;

}

}

class Solution {

private void postorder(TreeNode root, List<Integer> ans) {

if(root == null) return;

postorder(root.left, ans);

postorder(root.right, ans);

ans.add(root.val);

}

public List<Integer> postorderTraversal(TreeNode root) {

List<Integer> ans = new ArrayList<>();

postorder(root, ans);

return ans;

}

}

\*\*Instead of declaring ans as global you can also pass it to function, its just an another way of writing the code.

TC : O(3N) => O(N) and SC : O(N) => Recursive stack space

**Q69 : Size, Sum, height of Binary Tree**

(<https://course.acciojob.com/idle?question=a8e99a89-cde0-4f26-9570-49dff12b7624>)

(<https://course.acciojob.com/idle?question=336f7410-3904-4cf9-be1b-47773cfae7b4>)

(<https://course.acciojob.com/idle?question=57d8deff-acbe-407a-b504-6fe20f94770e>)

**M1 : Forming the answer (preorder version)**

class Solution

{

static int size = 0;

static int sum = 0;

static int maxDepth = 0;

private static void solveSize(Node root) {

if(root == null) return;

size++;

solveSize(root.left);

solveSize(root.right);

}

public static int getSize(Node root) {

size = 0;

solveSize(root);

return size;

}

private static void solveSum(Node root) {

if(root == null) return;

sum += root.data;

solveSum(root.left);

solveSum(root.right);

}

public static int getSum(Node root) {

sum = 0;

solveSum(root);

return sum;

}

private static void solveHeight(Node root, int currDepth) {

if(root == null) return;

maxDepth = Math.max(maxDepth, currDepth);

solveHeight(root.left, currDepth + 1);

solveHeight(root.right, currDepth + 1);

}

public static int getHeight(Node root) {

maxDepth = 0;

solveHeight(root, 1);

return maxDepth;

}

}

**M2 : Getting the answer (Postorder version)**

class Solution

{

public static int getSize(Node root) {

if(root == null) return 0;

int lsize = getSize(root.left);

int rsize = getSize(root.right);

return lsize + rsize + 1;

}

public static int getSum(Node root) {

if(root == null) return 0;

int lsum = getSum(root.left);

int rsum = getSum(root.right);

return lsum + rsum + root.data;

}

public static int getHeight(Node root) {

if(root == null) return 0;

int lheight = getHeight(root.left);

int rheight = getHeight(root.right);

return Math.max(lheight, rheight) + 1;

}

}

For all functions in both Methods, TC : O(3N) => O(N) and SC : O(N) => Recursive stack space

**Q70 : Balanced Binary Tree**

(<https://course.acciojob.com/idle?question=45eae2bf-6488-4ec5-85ab-598fc10d1647>)

private int height(TreeNode root) {

if(root == null) return 0;

int lh = height(root.left);

int rh = height(root.right);

return Math.max(lh, rh) + 1;

}

public boolean isBalanced(TreeNode root) {

if(root == null) return true;

// For every Node, abs(left height - right height) <= 1

int lh = height(root.left);

int rh = height(root.right);

if(Math.abs(lh - rh) > 1) return false;

return isBalanced(root.left) && isBalanced(root.right);

}

TC : O(N2) => at every node it is calling height() and SC : O(N) => Recursive stack space

**DEC 23rd**

**Q71 : Binary Tree Level Order Traversal**

**(**<https://leetcode.com/problems/binary-tree-level-order-traversal/>)

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

// Template for all BFS -> Graphs

// BFS -> Breadth First Search, DFS -> Depth First Search

List<List<Integer>> ans = new ArrayList<>();

if(root == null) return ans;

// Start with adding the root

Queue<TreeNode> q = new LinkedList<>();

q.add(root);

while(q.size() > 0) {

// In every level perform RPA for every node

// remove, print, add child

int size = q.size();

List<Integer> level = new ArrayList<>();

for(int i = 0; i < size; i++) {

TreeNode temp = q.remove();

level.add(temp.val);

if(temp.left != null) q.add(temp.left);

if(temp.right != null) q.add(temp.right);

}

ans.add(level);

}

return ans;

}

}

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

**Q72 : Left view of a Binary Tree**

**(**<https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1>)

class Tree

{

ArrayList<Integer> leftView(Node root)

{

ArrayList<Integer> ans = new ArrayList<>();

if(root == null) return ans;

Queue<Node> q = new LinkedList();

q.add(root);

while(q.size() > 0) {

int size = q.size();

for(int i = 0; i < size; i++) {

Node temp = q.remove();

//First Node of every level

if(i == 0) ans.add(temp.data);

if(temp.left != null) q.add(temp.left);

if(temp.right != null) q.add(temp.right);

}

}

return ans;

}

}

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

**Q73 : Right view of a Binary Tree**

**(**<https://practice.geeksforgeeks.org/problems/right-view-of-binary-tree/1>)

class Tree

{

ArrayList<Integer> leftView(Node root)

{

ArrayList<Integer> ans = new ArrayList<>();

if(root == null) return ans;

Queue<Node> q = new LinkedList();

q.add(root);

while(q.size() > 0) {

int size = q.size();

for(int i = 0; i < size; i++) {

Node temp = q.remove();

//Last Node of every level

if(i == size - 1) ans.add(temp.data);

if(temp.left != null) q.add(temp.left);

if(temp.right != null) q.add(temp.right);

}

}

return ans;

}

}

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

**Q74 : Vertical order traversal**

(<https://www.interviewbit.com/problems/vertical-order-traversal-of-binary-tree/>)

class Pair {

TreeNode node;

int scale;

Pair(TreeNode node, int scale) {

this.node = node;

this.scale = scale;

}

}

public class Solution {

public ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode root) {

ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

if(root == null) return ans;

// (Scale, List of node values)

Map<Integer, ArrayList<Integer>> hm = new HashMap<>();

Queue<Pair> q = new LinkedList<>();

int maxScale = Integer.MIN\_VALUE;

int minScale = Integer.MAX\_VALUE;

// (TreeNode, Scale)

q.add(new Pair(root, 0));

while(q.size() > 0) {

int size = q.size();

for(int i = 0; i < size; i++) {

Pair info = q.remove();

TreeNode temp = info.node;

int currScale = info.scale;

maxScale = Math.max(maxScale, currScale);

minScale = Math.min(minScale, currScale);

if(hm.containsKey(currScale)) {

ArrayList<Integer> oldlist = hm.get(currScale);

oldlist.add(temp.val);

hm.put(currScale, oldlist);

}

else {

ArrayList<Integer> newlist = new ArrayList<>();

newlist.add(temp.val);

hm.put(currScale, newlist);

}

if(temp.left != null) q.add(new Pair(temp.left, currScale - 1));

if(temp.right != null) q.add(new Pair(temp.right, currScale + 1));

}

}

for(int i = minScale; i <= maxScale; i++) {

ans.add(hm.get(i));

}

return ans;

}

}

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

**Q75 : Top view of Binary Tree**

(<https://practice.geeksforgeeks.org/problems/top-view-of-binary-tree/1>)

class Pair {

Node node;

int scale;

Pair(Node node, int scale) {

this.node = node;

this.scale = scale;

}

}

class Solution

{

static ArrayList<Integer> topView(Node root)

{

ArrayList<Integer> ans = new ArrayList<>();

if(root == null) return ans;

// (Scale, top node value)

Map<Integer, Integer> hm = new HashMap<>();

Queue<Pair> q = new LinkedList<>();

int maxScale = Integer.MIN\_VALUE;

int minScale = Integer.MAX\_VALUE;

// (TreeNode, Scale)

q.add(new Pair(root, 0));

while(q.size() > 0) {

int size = q.size();

for(int i = 0; i < size; i++) {

Pair info = q.remove();

Node temp = info.node;

int currScale = info.scale;

maxScale = Math.max(maxScale, currScale);

minScale = Math.min(minScale, currScale);

if(!hm.containsKey(currScale)) {

hm.put(currScale, temp.data);

}

if(temp.left != null)

q.add(new Pair(temp.left, currScale - 1));

if(temp.right != null)

q.add(new Pair(temp.right, currScale + 1));

}

}

for(int i = minScale; i <= maxScale; i++) {

ans.add(hm.get(i));

}

return ans;

}

}

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

**Q76 : Bottom view of Binary Tree**

(<https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1>)

class Pair {

Node node;

int scale;

Pair(Node node, int scale) {

this.node = node;

this.scale = scale;

}

}

class Solution

{

static ArrayList<Integer> topView(Node root)

{

ArrayList<Integer> ans = new ArrayList<>();

if(root == null) return ans;

// (Scale, top node value)

Map<Integer, Integer> hm = new HashMap<>();

Queue<Pair> q = new LinkedList<>();

int maxScale = Integer.MIN\_VALUE;

int minScale = Integer.MAX\_VALUE;

// (TreeNode, Scale)

q.add(new Pair(root, 0));

while(q.size() > 0) {

int size = q.size();

for(int i = 0; i < size; i++) {

Pair info = q.remove();

Node temp = info.node;

int currScale = info.scale;

maxScale = Math.max(maxScale, currScale);

minScale = Math.min(minScale, currScale);

hm.put(currScale, temp.data);

if(temp.left != null)

q.add(new Pair(temp.left, currScale - 1));

if(temp.right != null)

q.add(new Pair(temp.right, currScale + 1));

}

}

for(int i = minScale; i <= maxScale; i++) {

ans.add(hm.get(i));

}

return ans;

}

}

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

**Q77 : Boundary Traversal**

(<https://practice.geeksforgeeks.org/problems/boundary-traversal-of-binary-tree/1>)

class Solution

{

void leftBoundary(Node root, List<Integer> path) {

if(root == null) return;

if(root.left != null) {

path.add(root.data);

leftBoundary(root.left, path);

}

else if(root.right != null) {

path.add(root.data);

leftBoundary(root.right, path);

}

}

void leafNodes(Node root, List<Integer> path) {

if(root == null) return;

if(root.right == null && root.left == null) {

path.add(root.data);

return;

}

leafNodes(root.left, path);

leafNodes(root.right, path);

}

void rightBoundary(Node root, List<Integer> path) {

if(root == null) return;

if(root.right != null) {

path.add(root.data);

rightBoundary(root.right, path);

}

else if(root.left != null) {

path.add(root.data);

rightBoundary(root.left, path);

}

}

ArrayList <Integer> boundary(Node root) {

ArrayList<Integer> ans = new ArrayList<>();

if(root == null) return ans;

// edge if not written going down the code root gets added two times

if(root.left == null && root.right == null) {

ans.add(root.data);

return ans;

}

List<Integer> left = new ArrayList<>();

leftBoundary(root.left, left);

List<Integer> right = new ArrayList<>();

rightBoundary(root.right, right);

List<Integer> leafs = new ArrayList<>();

leafNodes(root, leafs);

ans.add(root.data);

for(int i = 0; i < left.size(); i++) ans.add(left.get(i));

for(int i = 0; i < leafs.size(); i++) ans.add(leafs.get(i));

for(int i = right.size() - 1; i >= 0; i--) ans.add(right.get(i));

return ans;

}

}

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

**Dec 25th**

**Q78 : Cousins in a Binary Tree (Contest)**

(<https://leetcode.com/problems/cousins-in-binary-tree/>)

class Solution {

int depthA = -1, depthB = -1;

int parentA = -1, parentB = -1;

private void solve(int parent, TreeNode root, int A, int B, int depth) {

if(root == null) return;

if(root.val == A) {

depthA = depth;

parentA = parent;

return;

}

if(root.val == B) {

depthB = depth;

parentB = parent;

return;

}

solve(root.val, root.left, A, B, depth + 1);

solve(root.val, root.right, A, B, depth + 1);

}

public boolean isCousins(TreeNode root, int A, int B) {

solve(-1, root, A, B, 0);

return (depthA == depthB) && (parentA != parentB);

}

}

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

**Q79 : Maximum running time of N computers (Contest)**

(<https://leetcode.com/problems/maximum-running-time-of-n-computers/>)

class Solution {

private boolean isPossible(int[] batteries, long time, int computers) {

long charge = 0;

int cnt = 0;

for(int i = 0; i < batteries.length; i++) {

charge += batteries[i];

if(charge >= time) {

cnt++;

if(cnt == computers) return true;

charge = charge - time;

}

}

return false;

}

public long maxRunTime(int n, int[] batteries) {

long start = 0;

long end = (long)1e14;

long ans = -1;

Arrays.sort(batteries);

while(start <= end) {

long mid = (start + end) / 2;

if(isPossible(batteries, mid, n)) {

ans = mid;

start = mid + 1;

}

else end = mid - 1;

}

return ans;

}

}

TC : O(NLogN + NLog(1e14)) and SC : O(1)

**Q80 : Root to Node Path**

(<https://course.acciojob.com/idle?question=a6cd9b9f-b898-4093-bcd6-de49aff55f4b>)

private boolean rootToNodePath(Node root, int target, ArrayList<Integer> ans) {

if(root == null) return false;

if(root.data == target) {

ans.add(root.data);

return true;

}

boolean res = rootToNodePath(root.left, target, ans);

res = res || rootToNodePath(root.right, target, ans);

if(res == true) ans.add(root.data);

return res;

}

public ArrayList<Integer> solve(Node root,int b) {

ArrayList<Integer> ans = new ArrayList<>();

rootToNodePath(root, b, ans);

Collections.reverse(ans);

return ans;

}

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

**Q81 : Path sum**

(<https://leetcode.com/problems/path-sum/>)

class Solution {

public boolean hasPathSum(TreeNode root, int targetSum) {

if(root == null) return false;

if(root.left == null && root.right == null) {

return ((targetSum - root.val) == 0);

}

boolean res = hasPathSum(root.left, targetSum - root.val);

res = res || hasPathSum(root.right, targetSum - root.val);

return res;

}

}

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

**Q82 : Path sum II**

(<https://leetcode.com/problems/path-sum-ii/>)

void solve(TreeNode root, int targetSum, List<List<Integer>> ans, List<Integer> path)

{

if(root == null) return;

if(root.left == null && root.right == null) {

if(targetSum - root.val == 0) {

path.add(root.val);

List<Integer> temp = new ArrayList<>(path); // O(N)

ans.add(temp);

path.remove(path.size() - 1);

}

return;

}

path.add(root.val); // we have added => O(1)

solve(root.left, targetSum - root.val, ans, path);

solve(root.right, targetSum - root.val, ans, path);

path.remove(path.size() - 1); // we must only remove => O(1)

}

public List<List<Integer>> pathSum(TreeNode root, int targetSum) {

List<List<Integer>> ans = new ArrayList<>();

List<Integer> path = new ArrayList<>();

solve(root, targetSum, ans, path);

return ans;

}

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

**Q83 : Lowest Common Ancestor of a Binary Tree**

(<https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/>)

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

List<TreeNode> path1 = new ArrayList<>();

List<TreeNode> path2 = new ArrayList<>();

rootToNodePath(root, p, path1);

rootToNodePath(root, q, path2);

// these paths are in reverse order

// so iterate from end

int i = path1.size() - 1;

int j = path2.size() - 1;

TreeNode LCA = null;

while(i >= 0 && j >= 0) {

if(path1.get(i) != path2.get(j))

break;

LCA = path1.get(i);

i--;

j--;

}

return LCA;

}

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

**Q84 : All Nodes Distance K in Binary Tree**

(<https://leetcode.com/problems/all-nodes-distance-k-in-binary-tree/>)

private void getKLevelDown(TreeNode root, TreeNode blockNode, int k, List<Integer> ans) {

if(root == null || root == blockNode) return;

if(k == 0) {

ans.add(root.val);

return;

}

getKLevelDown(root.left, blockNode, k - 1, ans);

getKLevelDown(root.right, blockNode, k - 1, ans);

}

public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {

List<TreeNode> path = new ArrayList<>();

rootToNodePath(root, target, path);

TreeNode blockNode = null;

List<Integer> ans = new ArrayList<>();

for(int i = 0; i < path.size(); i++) {

getKLevelDown(path.get(i), blockNode, k - i, ans);

blockNode = path.get(i);

}

return ans;

}

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

**Q85 : Burning Tree**

(<https://practice.geeksforgeeks.org/problems/burning-tree/1>)

class Solution

{

private static boolean rootToNodePath(Node root, int target, List<Node> ans) {

if(root == null) return false;

if(root.data == target) {

ans.add(root);

return true;

}

boolean res = rootToNodePath(root.left, target, ans);

res = res || rootToNodePath(root.right, target, ans);

if(res == true) ans.add(root);

return res;

}

private static int height(Node root, Node blockNode) {

if(root == null || blockNode == root) return -1;

return 1 + Math.max(height(root.left, blockNode), height(root.right, blockNode));

}

public static int minTime(Node root, int target)

{

List<Node> path = new ArrayList<>();

rootToNodePath(root, target, path);

Node blockNode = null;

int time = 0;

for(int i = 0; i < path.size(); i++) {

int h = height(path.get(i), blockNode);

time = Math.max(time, i + h);

blockNode = path.get(i);

}

return time;

}

}

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

**Q86 : Diameter of Binary Tree**

(<https://leetcode.com/problems/diameter-of-binary-tree/>)

class Solution {

// int[] => {height, diameter}

private int[] solve(TreeNode root) {

if(root == null) return new int[]{-1, 0};

int[] leftHD = solve(root.left);

int[] rightHD = solve(root.right);

int[] ans = new int[2];

ans[0] = Math.max(leftHD[0], rightHD[0]) + 1;

int md = leftHD[0] + rightHD[0] + 2;

ans[1] = Math.max(md, Math.max(leftHD[1], rightHD[1]));

return ans;

}

public int diameterOfBinaryTree(TreeNode root) {

int[] ans = solve(root);

return ans[1];

}

}

**Q87 : Binary Tree Maximum Path Sum**

(<https://leetcode.com/problems/binary-tree-maximum-path-sum/>)

class Solution {

// int[] => {maxPathSum, maxSumTillRoot}

private int[] solve(TreeNode root) {

if(root == null) return new int[]{Integer.MIN\_VALUE, 0};

int[] left = solve(root.left);

int[] right = solve(root.right);

int case1 = Math.max(left[0], right[0]);

int leftBranch = left[1] + root.val;

int rightBranch = right[1] + root.val;

// case2

int maxSumTillRoot = Math.max(root.val, Math.max(leftBranch, rightBranch));

int case3 = left[1] + root.val + right[1];

int[] ans = new int[2];

ans[0] = Math.max(case1, Math.max(maxSumTillRoot, case3));

ans[1] = maxSumTillRoot;

return ans;

}

public int maxPathSum(TreeNode root) {

int[] ans = solve(root);

return ans[0];

}

}

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