Day1:

1. Pre and suffix SUM:

int[] arr = {1, 2, 3, 4, 5};

int n = arr.length;

int[] prefixSum = new int[n];

prefixSum[0] = arr[0];

for (int i = 1; i < n; i++) {

prefixSum[i] = prefixSum[i - 1] + arr[i];

}

// Print the prefix sum array

for (int num : prefixSum) {

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

}

System.out.println();

int[] postFix=new int[n];

postFix[n-1]=arr[n-1];

for(int index=n-2;index>=0;index--){

postFix[index]=postFix[index+1]+arr[index];

}

System.out.println(Arrays.toString(arr));

System.out.println(Arrays.toString(postFix));

2. Rotation of elements of array- left and right:

public static void rotateArrayAntiClock(String[] arr, int k) {

int n = arr.length;

String[] rotatedArr = new String[n];

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

rotatedArr[(i - k+ n) % n] = arr[i];

System.out.println((i - k+ n) % n);

}

// Copy the rotated elements back to the original array

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

System.out.print(rotatedArr[i]+" ");

}

System.out.println();

}

public static void rotateArrayClock(String[] arr, int k) {

int n = arr.length;

String[] rotatedArr = new String[n];

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

rotatedArr[(i + k) % n] = arr[i];

System.out.println((i+k)%n);

}

for (int i = 0; i < n; i++) {System.out.print(rotatedArr[i]+" ");}

System.out.println();

}

public static void main(String[] args) {

String[] frameworks={"Node", "Django", "Vue", "Express", "Spring", "Flask", "Angular", "React", "Hibernate"};

System.out.println(Arrays.toString(frameworks));

rotateArrayClock(frameworks,2);

rotateArrayAntiClock(frameworks,2);

}

3. Finding equilibrium index of an array:

public static int equilibriumIndex(int[] arr){

int cumulateSum=0;

int westSum=0;

for(int each:arr){

cumulateSum+=each;

}

for(int index=0;index<arr.length;index++){

cumulateSum-=arr[index];

if(westSum==cumulateSum)

return index;

westSum+=arr[index];

}

return -1;

}

public static void main(String[] args) {

int[] arr = {-7, 1, 5, 2, -4, 3, 0};

System.out.println(equilibriumIndex(arr));

}

4. Inversion count of an array

Input: A[] = [1, 9, 6, 4, 5]

Output: The inversion count is 5

There are 5 inversions in the array: (9, 6), (9, 4), (9, 5), (6, 4), (6, 5)

public static int inversionCount(int[] arr){

int inversions=0;

for(int current=0;current<arr.length-1;current++){

for(int adjucent=current+1;adjucent<arr.length;adjucent++){

if(arr[current]>arr[adjucent]){inversions++;}

}

}

return inversions;

}

public static void main(String[] args) {

int[] arr = {45,91,10,67,21,6,11};

System.out.println(inversionCount(arr));

}

5. Mike has arranged a small party for the inauguration of his new startup. He has invited all of his fellow employees who are N in number. These employees are indexed with an array starting from 1 to N. In this startup, everyone knows each other’s salary. We will represent salary by an integer value. Mike has to arrange tables, where he will accommodate everyone. But he is a little thrifty in that, he wants to adjust everyone in as few tables as he can. Tables of various seating are available. Let’s say the cost of renting each table is K. All the employees have to seat in the order of the index. The only problem is that the employees with the same salary can get into arguments which can ruin the party.  
Mike came across the term inefficiency of arrangement, which can be defined as the sum of the cost of tables + the total number of people getting into arguments. Given the number of employees, N, and their salaries in array a[ ], he wants to find the optimal inefficiency, i.e., the smallest possible value for the inefficiency of arranging the N employees.

public static int findOptimalInefficiency(int N, int K, int[] salaries) {

Arrays.sort(salaries);

int[] dp = new int[N + 1]; dp[0] = 0;

for (int i = 1; i <= N; i++) {

int sum = 0, arguments = 0;dp[i] = Integer.MAX\_VALUE;

for (int j = 1; j <= i; j++) {

sum += salaries[j - 1]; int difference = sum - j \* salaries[j - 1];

arguments += Math.max(0, difference); dp[i] = Math.min(dp[i], dp[j - 1] + K + arguments);

}

}

return dp[N]; }

public static void main(String[] args) {

int[] arr = {5, 4, 3, 2, 1};

System.out.println(findOptimalInefficiency(arr.length,2,arr));

}

6. Maximum Subarray Sum using Divide and Conquer

public static void findMaxSumSubArray(int[] arr){

int maxSum=Integer.MIN\_VALUE, subSum=0;

for(int current=0;current<arr.length;current++){

subSum=0;

for(int sub=current;sub<arr.length;sub++){

subSum+=arr[sub];

//System.out.println(arr[sub]+" "+subSum);

if(subSum>maxSum){

maxSum=subSum;

}

}

}

System.out.println(maxSum);

}

public static void main(String[] args) {

int[] arr = {45,91,10,67,21,6,11};

findMaxSumSubArray(arr);

int[] nums = { 2, -4, 1, 9, -6, 7, -3 };

findMaxSumSubArray(nums);

}

7. Coin change using greedy approach:

public static int coinChange(int[] coins, int amount) {

Arrays.sort(coins);

int coinCount = 0;

int index = coins.length - 1;

while (amount > 0 && index >= 0) {

if (coins[index] <= amount) {

System.out.println(coins[index]);

amount -= coins[index];

coinCount++;

} else {index--;}

}

return amount==0?coinCount:-1;

}

public static void main(String[] args) {

int[] coins = {1, 2, 5};

System.out.println(coinChange(coins, 11));

coins=new int[]{5,10};

System.out.println(coinChange(coins, 50));

}

8. Mahesh and Suresh are playing a new game “Checkers“. This is a very simple game but becomes challenging when more expert players are playing. Below is the description of the game and rules: The game is played by 2 players. This game consists of an N\*M matrix. Each of the cells is background lit by lights. And these cells are either Green or Black. The green and black cells are randomly lit and will be represented with 1’s and 0’s respectively. Green cells are the cells that need to be captured. Black cells cannot be captured. Everyone is in the race to capture the maximum number of cells possible.

In a single chance, a player can capture all those adjacent cells which share an edge. Once there is no adjacent edge the chance breaks and the next player will play.

Mahesh always starts the game and Suresh is second.

Both players are playing optimally, find out how many cells Suresh captures.

public static int maxCellsCaptured(int[][] gameBoard) {

int n = gameBoard.length;

int m = gameBoard[0].length;

int[][] maheshCaptured = new int[n][m];

int[][] sureshCaptured = new int[n][m];

// mahesh check down and right direction from the current

for (int i = n - 1; i >= 0; i--) {

for (int j = m - 1; j >= 0; j--) {

if (gameBoard[i][j] == 1) {

maheshCaptured[i][j] = 1 + Math.max(

i + 1 < n ? maheshCaptured[i + 1][j] : 0,

j + 1 < m ? maheshCaptured[i][j + 1] : 0

);

}

}

}

// suresh check up and left direction from the current

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

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

if (gameBoard[i][j] == 1) {

sureshCaptured[i][j] = 1 + Math.max(

i - 1 >= 0 ? sureshCaptured[i - 1][j] : 0,j - 1 >= 0 ? sureshCaptured[i][j - 1] : 0

);

}

}

}

for(int[] row:maheshCaptured){

System.out.println(Arrays.toString(row));

}

for(int[] row:sureshCaptured){

System.out.println(Arrays.toString(row));

}

return Math.min(maheshCaptured[0][0], sureshCaptured[0][0]);

}

public static void main(String[] args) {

int[][] gameBoard = {

{0, 1, 1},

{1, 1, 0},

{1, 0, 1}

};

int sureshCellsCaptured = maxCellsCaptured(gameBoard);

System.out.println("Suresh captures " + sureshCellsCaptured + " cells.");

}

9. A chocolate factory is packing chocolates into packets. The chocolate packets here represent an array arrt of N number of integer values. The task is to find the empty packets(0) of chocolate and push it to the end of the conveyor belt(array).

public static void moveEmptyPacketsToEnd(int[] arr) {

int n = arr.length, nonEmptyIndex = 0;

// Traverse the array and move non-empty packets to the front

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

if (arr[i] != 0) {

arr[nonEmptyIndex] = arr[i];

System.out.println(nonEmptyIndex+" "+Arrays.toString(arr));nonEmptyIndex++;

}

}

// Fill the remaining slots with empty packets

while (nonEmptyIndex < n) {

arr[nonEmptyIndex] = 0;

nonEmptyIndex++;

}

}

public static void main(String[] args) {

int[] chocolatePackets = {2, 0, 3, 0, 1, 0, 5};

moveEmptyPacketsToEnd(chocolatePackets);

System.out.println(Arrays.toString(chocolatePackets));

}

10. Airport security officials confiscated several items of the passengers at the security checkpoint. All the items have been dumped into a huge box (array). Each item possesses a certain amount of risk[0,1,2]. Here, the risk severity of the items represents an array[] of N number of integer values. The task here is to sort the items based on their levels of risk in the array. The risk values range from 0 to 2.

public static void sortItemsByRisk(int[] arr) {

int n = arr.length;

int[] count = new int[3]; // Count of items with risk 0, 1, and 2

// Count the frequency of each risk level

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

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

}

System.out.println(Arrays.toString(count));

int index = 0;

// Reconstruct the sorted array based on risk levels

for (int i = 0; i < 3; i++) {

while (count[i] > 0) {

arr[index++] = i;count[i]--;

}

}

}

public static void main(String[] args) {

int[] items = {1, 0, 2,2,1, 1,2,2,1, 0, 2, 1, 0, 2};

sortItemsByRisk(items);

System.out.println(Arrays.toString(items));

}

11. Given N gold wires, each wire has a length associated with it. At a time, only two adjacent small wires are assembled at the end of a large wire and the cost of forming is the sum of their length. Find the minimum cost when all wires are assembled to form a single wire.

Note: using greedy approach

Example:

[1, 1, 6, 6, 7, 8]

[1, 6, 6, 7, 8, 8] 2

[1, 6, 7, 8, 8, 8] 9

[1, 7, 8, 8, 8, 8] 16

[1, 8, 8, 8, 8, 8] 24

[1, 9, 8, 8, 8, 8] 33

Minimum cost to assemble all gold wires: 33

public static int minimumCostToAssembleWires(int[] wires) {

int n = wires.length;

Arrays.sort(wires); // Sort the wires in ascending order

System.out.println(Arrays.toString(wires));

int totalCost = 0;

while (n > 1) {

int cost = wires[0] + wires[1]; // Merge the two shortest wires

totalCost += cost;

// Remove the two shortest wires and insert the merged wire

wires[1] = cost;

for (int i = 2; i < n; i++) {

wires[i - 1] = wires[i];

}

System.out.println(Arrays.toString(wires)+" "+totalCost);

n--; // Decrease the number of wires

}

return totalCost;

}

public static void main(String[] args) {

int[] wires = {7, 6, 8, 6, 1, 1};

int minCost = minimumCostToAssembleWires(wires);

System.out.println("Minimum cost to assemble all gold wires: " + minCost);

}

12. The Climbing Stairs problem

This is one of the most popular coding problems which can be solved using the Dynamic Programming technique. In this problem, you are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. The question is, in how many distinct ways can you climb to the top?

Note: Given n will be a positive integer.

Example 1:

Input: 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step.

public static int climbStairsRecursion(int n) {

if (n <= 0) {return 0;

} else if (n == 1) {return 1;

} else if (n == 2) {

return 2;

} else {

System.out.println(n);

return climbStairs(n - 1) + climbStairs(n - 2);

}

}

public static int climbStairs(int n) {

if (n <= 0) {return 0;}

else if (n == 1) {return 1;}

else if (n == 2) {return 2;}

int[] dp = new int[n + 1];

dp[1] = 1;dp[2] = 2;

for (int i = 3; i <= n; i++) {

dp[i] = dp[i - 1] + dp[i - 2];

}

return dp[n];

}

public static void main(String[] args) {

int n = 5; // Number of stairs

int ways = climbStairs(n);

System.out.println("Number of ways: " + ways);

}