Coding practice Problems:

1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

   Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11

   Explanation: The subarray {7, -1, 2, 3} has the largest sum 11. Input: arr[] = {-2, -4} Output: –2 Explanation: The subarray {-2} has the largest sum -2.

   Input: arr[] = {5, 4, 1, 7, 8} Output: 25 Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

CODE:

```
import java.util.*;

public class Kadane{

        public static void main(String [] args){

                Scanner sc = new Scanner(System.in);

                System.out.print("Enter your Size");

                int n = sc.nextInt();

                int [] arr = new int[n];

                System.out.print("Enter your Elements");

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


                        arr[i] = sc.nextInt();

                }


                int ans = arr[0], tot = arr[0];

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


                        tot= Math.max(arr[i]+tot,arr[i]);

                        ans=Math.max(ans,tot);

                }

                System.out.println(ans);
```

```
        }


}
```

// TIME COMPLEXITY : O(N)

// SPACE COMPLEXITY : O(1)

2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

   Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180

   Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180

   Input: arr[] = {-1, -3, -10, 0, 60} Output: 60 Explanation: The subarray with maximum product is {60}.

CODE :

```
import java.util.*;
public class MaxSubArray{
        public static void main(String [] args){
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter your Size");
                int n = sc.nextInt();
                int [] arr = new int[n];
                System.out.print("Enter your Elements");
                for(int i=0;i<n;i++){

                        arr[i] = sc.nextInt();
                }

                int ps =1,ans=arr[0],ls=1;
                for(int i=1;i<n;i++){
                        ps*=arr[i];
                        ls*=arr[n-i-1];
                        ans=Math.max(ans,Math.max(ps,ls));
                        if (ps==0) ps=1;
                        if(ls==0) ls=1;
```

```
            }
            System.out.println(ans);


        }


}

// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(1)
```

3. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1. Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0 Output : 4

   Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3 Output : -1

   Input : arr[] = {50, 10, 20, 30, 40}, key = 10 Output : 1

CODE :

```
import java.util.*;

public class Rotated{
        public static void main(String [] args){
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter your Size");
                int n = sc.nextInt();
                int [] arr = new int[n];
                System.out.print("Enter your Elements");
                for(int i=0;i<n;i++){

                        arr[i] = sc.nextInt();
                }
                System.out.print("Enter your Search Element");
                int x = sc.nextInt();
```

```
                int l = 0,ans= -1, h=n-1;
                while(l<=h){
                        int mid = (l+h)/2;
                        // System.out.println(mid);
                        if(arr[mid] == x){
                                ans = mid;
                                break;
                        }
                        else if(arr[mid]>x){
                                if(arr[l]<=x && x<=arr[mid]){
                                        h=mid-1;
                                }
                                else l=mid+1;
                        }
                        else{
                                if(arr[mid]<=x && x<=arr[h]){
                                        l=mid+1;
                                }
                                else h=mid-1;
                        }

                }
                System.out.println("Your Answer : "+ans);


        }


}

// TIME COMPLEXITY : O(LOG(N))
```

// SPACE COMPLEXITY : O(1)

4. Container with Most Water

   Input: arr = [1, 5, 4, 3] Output: 6 Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 * 2 = 6

   Input: arr = [3, 1, 2, 4, 5] Output: 12 Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12

CODE :

```
import java.util.*;

public class Water{

	public static void main(String [] args){

		Scanner sc = new Scanner(System.in);

		System.out.print("Enter your Size");

		int n = sc.nextInt();

		int [] arr = new int[n];

		System.out.print("Enter your Elements");

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


			arr[i] = sc.nextInt();

		}


		int l = 0,ans= 0, h=n-1;

		while(l<=h){

			if(arr[l]<=arr[h]){

				ans=Math.max(ans,arr[l]*(h-l));

				l++;

			}
```

```
                else{
                        ans=Math.max(ans,arr[h]*(h-l));
                        h--;

                }
        }
        System.out.println("Your Answer : "+ans);

    }

}

// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(1)
```

5. Find the Factorial of a large number

   Input: 100 Output:
   933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000 00

   Input: 50 Output:
   30414093201713378043612608166064768844377641568960512000000000000

CODE:

```
import java.util.*;
import java.math.*;
public class Factorial{
        public static void main(String [] args){
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter your Size");
                int  n = sc.nextInt();
```

```
                BigInteger ans =  new BigInteger("1");


                for(int i=1;i<n;i++){
                        ans= ans.multiply(BigInteger.valueOf(i));
                }
                System.out.println("Your Answer : "+ans)
        }
}


// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(1)
```

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

   Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image.

   Input: arr[] = {3, 0, 2, 0, 4} Output: 7 Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

   Input: arr[] = {1, 2, 3, 4} Output: 0 Explanation : We cannot trap water as there is no height bound on both sides

   Input: arr[] = {10, 9, 0, 5} Output: 5 Explanation : We trap 0 + 0 + 5 + 0 = 5

CODE :

```
import java.util.*;
public class Trap{
        public static void main(String [] args){
                Scanner sc = new Scanner(System.in);
                int n;
                System.out.print("Enter size");
                n = sc.nextInt();
                int [] arr = new int[n];
                System.out.print("Enter elements");
                for(int i=0;i<n;i++){
```

```
                arr[i] = sc.nextInt();

            }
            int i=0,j=n-1,ans =0;
            while(i<j){
              if(arr[i]<=arr[j]){

                int tmp = arr[i];
                while (i<j && tmp>=arr[i]){
                    ans+=tmp-arr[i++];

                }

              }
              else{
              int tmp = arr[j];
                while (i<j && tmp>=arr[j]){
                    ans+=tmp-arr[j--];

                }
              }

            }
            System.out.println("Your Answer "+ans);


        }
        // TIME COMPLEXITY : O(N);
        // SPACE COMPLEXITY: O(1);

}
```

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

   Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3 Output: 2 Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

   Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5 Output: 7 Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

CODE:

```
import java.util.*;

public class Chocolate{

	public static void main(String [] args){

		Scanner sc = new Scanner(System.in);

		int n,m,ans = Integer.MAX_VALUE;

		System.out.print("Enter size");

		n = sc.nextInt();

		System.out.print("Enter Students size");

		m = sc.nextInt();

		int [] arr = new int[n];

		System.out.print("Enter elements");

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

			arr[i] = sc.nextInt();


		}

		Arrays.sort(arr);

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

			ans=Math.min(ans,arr[i+m-1]-arr[i]);

		}

		System.out.println("Your Answer "+ans);



	}
```

// TIME COMPLEXITY : O(NLOG(N));

// SPACE COMPLEXITY: O(1);

8. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

   Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]] Output: [[1, 4], [6, 8], [9, 10]] Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

   Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]] Output: [[1, 6], [7, 8]] Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

CODE :

```
#include <bits/stdc++.h>
using namespace std;
 #define vvi vector<vector<int>>
 int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.precision(10);
    int n;
    cin>>n;
    vvi arr(n,vector<int>(2,0)),ans;
    for(int i=0;i<n;i++) {
      for(int j=0;j<2;j++) {
          cin>>arr[i][j];
      }
    }
    sort(arr.begin(), arr.end());
    int i=1,fs =arr[0][0],ls=arr[0][1],j=0;
    ans.push_back(arr[0]);
    while(i< n) {
```

```
        int f= arr[i][0], l= arr[i][1];
        if(ls>f){
            ans[j][1]= max(l,ls);
        }else{
            j++;
            ls=l;
            ans.push_back({f,l});
        }
        i++;
    }
    for(int i=0;i<ans.size();i++) {
        for(int j=0;j<2;j++){
            cout<<ans[i][j]<<" ";
        }
    }

    return 0;
}
```

9. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
int main()
{
   ios::sync_with_stdio(false);
   cin.tie(0);
```

```
cout.precision(10);
int n, m, col = 0,row=0;
cin >> n >> m;
vvi arr(n, vector<int>(m, 0)), ans;
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < m; j++)
   {
      cin >> arr[i][j];
   }
}
for(int i=0;i<n;i++) { if(arr[i][0]==1) row=1;}
for(int i=0;i<m;i++) { if(arr[0][i] ==1) col=1;}

for(int i=1;i<n;i++){
   for(int j=1;j<m;j++){
      if(arr[i][j]){
         arr[i][0]=1;
         arr[0][j]=1;
      }
   }
}
for(int i=1;i<n;i++){
   for(int j=1;j<m;j++){
      if(arr[i][0] || arr[0][j]) {
         arr[i][j]=1;
      }
   }
}

if(row){
```

```
    for(int i=0;i<n;i++) arr[i][0]=1;
  }
  if(col){
    for(int j=0;j<m;j++) arr[0][j]=1;
  }

  return 0;
}
// TIME COMPLEXITY : O(N^2)
// SPACE COMPLEXITY : O(1)
```

10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

    Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

    Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}} Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11 Explanation: The output is matrix in spiral format.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
int main()
{
  int n,m;
  cin >> n >> m;
  vvi arr(n, vector<int>(m, 0));
  for (int i = 0; i < n; i++){
    for(int j=0;j<m;j++){
      cin >> arr[i][j];
    }
```

```
    }

    int t=0,l=0,r=m-1,b=n-1;
    while (t<=b && l<=r){
        for(int i=l;i<=r;i++){
            cout<<arr[t][i]<<" ";
        }
        t+=1;
        for(int i=t;i<=b;i++){
            cout<<arr[i][r]<<" ";
        }
        r-=1;
        if(t<=b){
            for(int i=r;i>=l;i--){
                cout<<arr[b][i]<<" ";
            }
        }
        b-=1;
        if(l<=r){
            for(int i=b;i>=t;i--){
                cout<<arr[i][l]<<" ";
            }
        }
        l+=1;
    }
    return 0;
}
// TIME COMPLEXITY : O(N^2)
// SPACE COMPLEXITY : O(1)
```

13. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

Input: str = “((()))()()” Output: Balanced

Input: str = “())((())” Output: Not Balanced

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
int main()
{
   string s;
   cin>>s;
   int n = s.length();
   stack<int> st;
   bool b = false;
   for(int i=0; i<n; i++){
      if(s[i] =='('){
         st.push(i);
      }
      else{
         if(st.empty()){
            b = true;
            break;
         }
         st.pop();
      }
   }
   if(b) cout<<"Not Balanced"<<endl;
   else cout<<"Balanced"<<endl;
   return 0;
```

```
}
// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N)
```

14. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.
Input: s1 = “geeks” s2 = “kseeg” Output: true Explanation: Both the string have same characters with same frequency. So, they are anagrams.
Input: s1 = “allergy” s2 = “allergic” Output: false Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams.
Input: s1 = “g”, s2 = “g” Output: true Explanation: Characters in both the strings are same, so they are anagrams.

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
int main()
{
   string s1, s2;
   cin >> s1;
   cin >> s2;
   int n = s1.length(), m = s2.length();
   if (n != m)
   {
      cout << "False" << endl;
      return 0;
   }
   unordered_map<char, int> mp;
```

```
    for (int i = 0; i < n; i++)
    {
        mp[s1[i]]++;
        mp[s2[i]]--;
    }
    bool ans = true;
    for (auto it : mp)
    {
        if (it.second != 0)
        {
            ans = false;
            break;
        }
    }
    if (ans)
        cout << "True" << endl;
    else
        cout << "False" << endl;

    return 0;
}
// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N)
```

15. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.
Input: str = “forgeeksskeegfor” Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

CODE :

```
#include <bits/stdc++.h>
```

```
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
int main()
{
  string s;
  cin >> s;
  int n = s.length(),maxLen=0,start=0;
  if(n ==0 ) {
   cout << "Empty String" << endl;
   return 0;
  }
   for(int i=0; i<n; i++){
      for(int j=0;j<=1;j++){
         //Expansion center algo i have used for both even and odd palindromes
         int l=i;
         int h=i+j;
         while (l>=0 && h<n && s[l]==s[h]){
            if(h-l+1 > maxLen){
               start =l;
               maxLen = h-l+1;
            }
            l--;
            h++;
         }
      }
   }
   cout << "Longest Palindrome substring: " << s.substr(start,maxLen) << endl;

   return 0;
```

}

// TIME COMPLEXITY : O(N^2)

// SPACE COMPLEXITY : O(1)

16. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.
Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”] Output: gee Explanation: “gee” is the longest common prefix in all the given strings.
Input: arr[] = [“hello”, “world”] Output: -1 Explanation: There‟s no common prefix in the given strings.

CODE :

```
#include <bits/stdc++.h>

using namespace std;

#define vvi vector<vector<int>>

#define vi vector<int>

#define pb push_back

#define ll long long

int main()

{

   int n;

   cin>>n;

   vector<string> lst(n);

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

      cin>> lst[i];

   }

   int l=INT_MAX;

   string c="";

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

      int tmp = lst[i].length();

      if(tmp<l){

         c=lst[i];

         l=tmp;

      }
```

```
    }
    // cout<<l<<c<<"\n";
    for(int i = 0; i < l; i++){
        for(int j = 0; j < n; j++){
            if(c[i]!=lst[j][i]){
                if (i==0) {
                    cout<<-1;
                    return 0;
                }
                cout<<c.substr(0,i);
                return 0;
            }
        }
    }
    cout<<c<<"\n";

    return 0;
}
// TIME COMPLEXITY : O(N^2)
// SPACE COMPLEXITY : O(1)
```

17. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.
Input : Stack[] = [1, 2, 3, 4, 5] Output : Stack[] = [1, 2, 4, 5]
Input : Stack[] = [1, 2, 3, 4, 5, 6] Output : Stack[] = [1, 2, 4, 5, 6]

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
```

```
void f(stack<int>& st,int mid){
    if(mid==0){
        st.pop();
        return;
    }
    int t =st.top();
    st.pop();
    f(st, mid-1);
    st.push(t);
}

void p(stack<int>& st,int ind,int n){
    if(ind==n)
        return;
    int t = st.top();
    st.pop();
    p(st, ind+1,n);
    cout<<t<<" ";

}
int main()
{
    int n;
    cin>>n;
    int mid = n/2;
    stack<int> st;
    for(int i=0; i<n; i++){
        int x;
        cin>>x;
        st.push(x);
```

```
    }
    f(st, mid);
    p(st,0,n-1);



    return 0;
}
// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N) recursion stack space
```

18. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [4 , 5 , 2 , 25] Output: 4 5 2 –> 5 –> 25 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side

Input: arr[] = [13 , 7, 6 , 12] Output: 13 –> 7 -1 –> 12 6 12 –> 12 –> -1 Explanation: 13 and 12 don‟t have any element greater than them present on the right side

CODE :

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define vi vector<int>
#define pb push_back
#define ll long long
int main()
{
  int n;
  cin>>n;
   vi v(n);
   for(int i=0; i<n; i++) cin>>v[i];
   stack<int> st;
   vi ans;
```

```
    for(int i=n-1; i>=0; i--){
        if(st.empty()) {
            ans.pb(-1);
            st.push(v[i]);
        }
        else{
            while(!st.empty() && st.top()<=v[i]) st.pop();
            if(st.empty()) ans.pb(-1);
            else ans.pb(st.top());
            st.push(v[i]);
        }
    }
    reverse(ans.begin(), ans.end());
    for(auto i:ans) cout<< i<<" ";
    return 0;
}
// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N)
```

19. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

CODE :

```
#include <bits/stdc++.h>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
```

```
void printRightView(TreeNode* root) {
    if (!root) return;
    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; i++) {
            TreeNode* current = q.front();
            q.pop();
            if (i == levelSize - 1) {
                cout << current->val << " ";
            }
            if (current->left) q.push(current->left);
            if (current->right) q.push(current->right);
        }
    }
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(6);
    root->right->right->right = new TreeNode(7);

    cout << "Right view of tree: ";
    printRightView(root);
    cout << endl;
```

```
    return 0;
}
// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N)
```

20. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

CODE :

```
#include <bits/stdc++.h>
using namespace std;

class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int maxDepth(TreeNode* root) {
    if (!root) return 0;
    int l = maxDepth(root->left);
    int r = maxDepth(root->right);
    return max(l, r) + 1;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
```

```
    root->left->right = new TreeNode(5);
    cout << "Depth of tree: " << maxDepth(root) << endl;
    return 0;
}

// TIME COMPLEXITY : O(N)
// SPACE COMPLEXITY : O(N)
```