****

**Technical Assessment**

**Question no 1:**

**Given a matrix of size R\*C. Traverse the matrix in spiral form.**

**Example 1:**

**Input:**

**R = 4, C = 4**

**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**

**Explanation:**

**Example 2:**

**Input:**

**R = 3, C = 4**

**matrix[][] = {{1, 2, 3, 4},**

**{5, 6, 7, 8},**

**{9, 10, 11, 12}}**

**Output:**

**1 2 3 4 8 12 11 10 9 5 6 7**

**Explanation:**

**Applying same technique as shown above,**

**output for the 2nd testcase will be**

**1 2 3 4 8 12 11 10 9 5 6 7.**

**Your Task: You dont need to read input or print anything. Complete the function spirallyTraverse() that takes matrix, R and C as input parameters and returns a list of integers denoting the spiral traversal of matrix.   Expected Time Complexity: O(R\*C) Expected Auxiliary Space: O(R\*C)  Constraints: 1 <= R, C <= 100 0 <= matrixi <= 100**

**Solution:**

**Assumption:** R&C are defined globally and using that the matrix is created.

**void spirallyTraverce**(int r, int c, int matrix[R][C])

{

if (r<1 || c>100){

cout<<" Not in the range. ";

return;

}

else{

int i, k = 0, l = 0;

while (k < r && l < c) {

for (i = l; i < c; ++i) {

cout << matrix[k][i] << " ";

}

k++;

for (i = k; i < r; ++i) {

cout << matrix[i][c - 1] << " ";

}

c--;

if (k < r) {

for (i = c - 1; i >= l; --i) {

cout << matrix[r - 1][i] << " ";

}

r--;

}

if (l < c) {

for (i = r - 1; i >= k; --i) {

cout << matrix[i][l] << " ";

}

l++;

}

}

}

}

**Question: 2**

Given **N** activities with their start and finish day given in array **start[ ]** and **end[ ]**. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a given day. **Note :**Duration of the activity includes both starting and ending day.

**Example 1:**

**Input:**

N = 2

start[] = {2, 1}

end[] = {2, 2}

**Output:**

1

**Explanation:**

A person can perform only one of the

given activities.

**Example 2:**

**Input:**

N = 4

start[] = {1, 3, 2, 5}

end[] = {2, 4, 3, 6}

**Output:**

3

**Explanation:**

A person can perform activities 1, 3

and 4.

**Your Task :** You don't need to read input or print anything. Your task is to complete the function ***activityselection()*** which takes array **start[ ],** array **end[ ]** and integer **N** as input parameters and returns the maximum number of activities that can be done.

**Expected Time Complexity** : O(N \* Log(N)) **Expected Auxilliary Space**: O(N)

**Constraints:** 1 ≤ N ≤ 2\*105 1 ≤ start[i] ≤ end[i] ≤ 109

**Solution:**

**Assumption:** Both matrices will be passed of same length.

void activityselection (int start[], int end[], int n)

{

if (n<1 || n>210 ){

cout<<"Not in the range. ";

return;

}

int i=0, j,count=1;

for (j = 1; j < n; j++)

{

if (start[j] >= end[i])

{

count=count+1;

i = j;

}

}

cout<<"A person can perform activities: "<<count;

}

**Question no 3**

Given a string, find the minimum number of characters to be inserted to convert it to palindrome. For Example: ab: Number of insertions required is 1. **b**ab or aba aa: Number of insertions required is 0. aa abcd: Number of insertions required is 3. **dcb**abcd

**Example 1:**

**Input:** str = "abcd"

**Output:** 3

**Explanation:** Inserted character marked

with bold characters in **dcb**abcd

**Example 2:**

**Input:** str = "aa"

**Output:** 0

**Explanation:**"aa" is already a palindrome.

**Your Task:** You don't need to read input or print anything. Your task is to complete the function **countMin()** which takes the string **str**as inputs and returns the answer.  **Expected Time Complexity:** O(N2), N = |str| **Expected Auxiliary Space:** O(N2)  **Constraints:** 1 ≤ |str| ≤ 103 str contains only lower case alphabets.

**Solution:**

**Assumptions:** I can use an extra function of isPalindrome.

bool isPalindrome(string s, int start, int end)

{

while (start < end)

{

if (s[start++] != s[end--])

return false;

}

return true;

}

void countMin(string s)

{

int n = s.length();

if (n<1 || n>103){

cout<<"Out of range. ";

return;

}

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

{

if (isPalindrome(s,0,i))

{

cout<<"Minimum characters required to make it palindrome is "<<(n-i-1)<<endl;

break;

}

}

}

**Question no 4**

In a candy store, there are **N** different types of candies available and the prices of all the N different types of candies are provided to you. You are now provided with an attractive offer. You can buy a single candy from the store and get at most **K** other candies ( all are different types ) for free. Now you have to answer two questions. Firstly, you have to find what is the **minimum amount of money** you have to spend to buy all the**N**different candies. Secondly, you have to find what is the **maximum amount of money** you have to spend to buy all the N different candies. In both the cases you must utilize the offer i.e. you buy one candy and get **K**other candies for free.

**Example 1:**

**Input:**

N = 4

K = 2

candies[] = {3 2 1 4}

**Output:**

3 7

**Explanation:**

As according to the offer if you buy

one candy you can take at most two

more for free. So in the first case,

you buy the candy which costs 1 and

takes candies worth 3 and 4 for free,

also you buy candy worth 2 as well.

So **min cost** : 1+2 =3.

In the second case, you can buy the

candy which costs 4 and takes candies

worth 1 and 2 for free, also you need

to buy candy worth 3 as well.

So **max cost :** 3+4 =7.

**Example 2:**

**Input:**

N = 5

K = 4

candies[] = {3 2 1 4 5}

**Output:**

1 5

**Explanation:**

For minimimum cost buy the candy with

the cost 1 and get all the other candies

for free.

For maximum cost buy the candy with

the cost 5 and get all other candies

for free.

**Your Task:** You don't need to read input or print anything. Your task is to complete the function **candyStore()** which takes the array candies[], its size Nand an integer Kas input parameters and returns the minimum amount and maximum amount of money to buy all candies according to the offer.  **Expected Time Complexity:** O(NLogN) **Expected Auxiliary Space:** O(1)

**Constraints:** 1 <= **N**<= 100000  0 <= **K** <= N-1 1 <= **candies[i]** <= 10000

**Solution:**

**Assumption:** Algorithm library included.

int minMoney(int arr[], int n, int k)

{

if(n<1 || n>100000 || k<0 || k>n-1 ){

cout<<"Out of range. "<<endl;

return 0;

}else{

int minCandy = 0;

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

minCandy = minCandy + arr[i];

n = n - k;

}

return minCandy;

}

}

int maxMoney(int arr[], int n, int k)

{

if(n<1 || n>100000 || k<0 || k>n-1 ){

cout<<"Out of range. "<<endl;

return 0;

}

else{

int maxCandy= 0, p = 0;

for (int i = n - 1; i >= p; i--)

{

maxCandy = maxCandy + arr[i];

p = p + k;

}

return maxCandy;

}

}