# DOMAIN WINTER CAMP

*Department of Computer Science and Engineering*

Name:- Aman Pal UID:- 22BCS10188 Section:- 22KPIT-901-A

**DAY-5**

**Q.1. Find First and Last Position of Element in Sorted Array. Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return [-1, -1].**

**Program Code:-**

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Solution
{
public:
  vector<int> searchRange(vector<int> &nums, int target)
  {
    vector<int> result = {-1, -1};
    int left = 0, right = nums.size() - 1;

    // Find the first position
    while (left <= right)
    {
      int mid = left + (right - left) / 2;
      if (nums[mid] >= target)
      {
        right = mid - 1;
      }
      else
      {
        left = mid + 1;
      }
```

```
    }

    if (left >= nums.size() || nums[left] != target)
    {
        return result;
    }

    result[0] = left;

    // Find the last position
    right = nums.size() - 1;
    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        if (nums[mid] <= target)
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }

    result[1] = right;
    return result;
  }
};

int main()
{
  Solution solution;

  // Take input from the user
  int n, target;
  cout << "Enter the number of elements in the array: ";
  cin >> n;

  vector<int> nums(n);
  cout << "Enter the elements of the array in sorted order: ";
  for (int i = 0; i < n; ++i)
  {
    cin >> nums[i];
  }

  cout << "Enter the target value: ";
  cin >> target;

  vector<int> result = solution.searchRange(nums, target);
```

```
    cout << "First and last positions of " << target << ": [" << result[0] << ", " <<
result[1] << "]" << endl;

    return 0;
}
```

**Output:-**

```
le } ; if ($?) { .\tempCodeRunnerFile }
Enter the number of elements in the array: 5
Enter the elements of the array in sorted order:  4 5 6 7 8
Enter the target value: 7
First and last positions of 7: [3, 3]
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 5>
```

**Q.2. Pair Sum Closet to 0. Given an integer array of N elements. You need to find the maximum sum of two elements such that sum is closest to zero.**

**Program Code:-**

```
    #include
    <iostream>
#include <vector>
#include <cmath>
#include <limits.h>
#include <algorithm>
using namespace std;

class Solution
{
public:
    int closestPairSum(vector<int> &arr)
    {
        int n = arr.size();
        if (n < 2)
            return 0;

        sort(arr.begin(), arr.end());
        int left = 0, right = n - 1;
        int closestSum = INT_MAX;
        int resultSum = 0;
```

```
    while (left < right)
    {
      int sum = arr[left] + arr[right];


      if (abs(sum) < closestSum)
      {
        closestSum = abs(sum);
        resultSum = sum;
      }
      if (sum < 0)
      {
        left++;
      }
      else
      {
        right--;
      }
    }

    return resultSum;
  }
};

int main()
{
  Solution solution;

  int n;
  cout << "Enter the number of elements in the array: ";
  cin >> n;

  vector<int> arr(n);
  cout << "Enter the elements of the array: ";
  for (int i = 0; i < n; ++i)
  {
    cin >> arr[i];
  }

  int result = solution.closestPairSum(arr);
  cout << "Sum of two elements closest to zero: " << result << endl;

  return 0;
}
```

**Output:-**

```
Enter the number of elements in the array: 5
Enter the elements of the array: 7 6 1 2 8
Sum of two elements closest to zero: 3
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 5>
```

**Q3. Find the Kth Smallest Sum of a Matrix With Sorted Rows.**

You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.

You are allowed to choose exactly one element from each row to form an array.

Return the kth smallest array sum among all possible arrays.

**Program code-**

include iostream

```
#include <vector>
#include <queue>
#include <set>
using namespace std;

class Solution
{
public:
  int kthSmallestSum(vector<vector<int>> &mat, int k)
  {
    int m = mat.size(), n = mat[0].size();
    priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>> pq;
    set<vector<int>> seen;

    vector<int> init(m, 0);
    int initialSum = 0;
    for (int i = 0; i < m; ++i)
    {
      initialSum += mat[i][0];
    }

    vector<int> initialState = {initialSum};
    initialState.insert(initialState.end(), init.begin(), init.end());
    pq.push(initialState);
    seen.insert(init);
```

```
    while (k--)
    {
      auto curr = pq.top();
      pq.pop();

      int sum = curr[0];
      vector<int> indices(curr.begin() + 1, curr.end());

      if (k == 0)
        return sum;

      for (int i = 0; i < m; ++i)
      {
        if (indices[i] + 1 < n)
        {
          auto next = indices;
          next[i]++;
          if (seen.insert(next).second)
          {
            int newSum = sum - mat[i][indices[i]] + mat[i][next[i]];
            vector<int> nextState = {newSum};
            nextState.insert(nextState.end(), next.begin(), next.end());
            pq.push(nextState);
          }
        }
      }
    }

    return -1; // Should not reach here
  }
};

int main()
{
  Solution solution;

  // Take input from the user
  int m, n, k;
  cout << "Enter the number of rows in the matrix: ";
  cin >> m;
  cout << "Enter the number of columns in the matrix: ";
  cin >> n;

  vector<vector<int>> mat(m, vector<int>(n));
  cout << "Enter the elements of the matrix row by row: \n";
  for (int i = 0; i < m; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
```

```
            cin >> mat[i][j];
        }
    }

    cout << "Enter the value of k: ";
    cin >> k;

    int result = solution.kthSmallestSum(mat, k);
    cout << "The " << k << "th smallest sum of the matrix is: " << result << endl;

    return 0;
}
```

**Output:-**

```
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 5> cd
Enter the number of rows in the matrix: 2
Enter the number of columns in the matrix: 1
Enter the elements of the matrix row by row:
1 2 1 6 4 5
Enter the value of k: The 1th smallest sum of the matrix is: 3
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 5>
```

**Q4.**

**Max Chunks To Make Sorted II**

You are given an integer array arr.

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

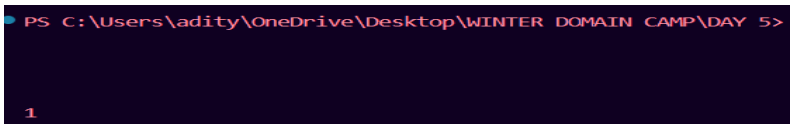
**Return the largest number of chunks we can make to sort the array.**

**Program Code:-**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxChunksToSorted(vector<int> &arr)
{
    int n = arr.size();

    // Create an array to store the minimum value from the right
```

```cpp
    vector<int> minRight(n);
    minRight[n - 1] = arr[n - 1];

    // Fill the minRight array
    for (int i = n - 2; i >= 0; --i)
    {
        minRight[i] = min(arr[i], minRight[i + 1]);
    }

    // Initialize variables
    int maxLeft = arr[0];
    int chunks = 0;

    // Traverse through the array and count valid splits
    for (int i = 0; i < n - 1; ++i)
    {
        maxLeft = max(maxLeft, arr[i]);
        if (maxLeft <= minRight[i + 1])
        {
            chunks++;
        }
    }

    // Add the last chunk
    return chunks + 1;
}

int main()
{
    vector<int> arr = {5, 4, 3, 2, 1};
    cout << maxChunksToSorted(arr) << endl; // Output: 1
    return 0;
}
```

**Output:-**




## Q.5. Median of Two Sorted Arrays.

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

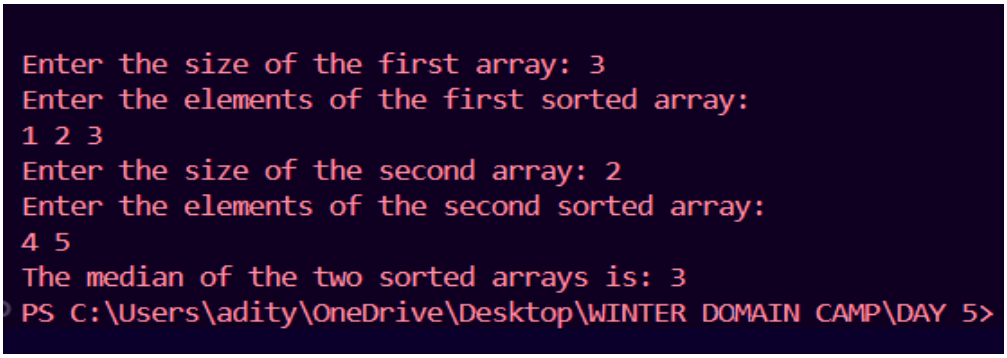
The overall run time complexity should be O(log (m+n)).

**Program Code:-**

```
#include
<iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

double findMedianSortedArrays(vector<int> &nums1, vector<int> &nums2)
{
  if (nums1.size() > nums2.size())
  {
    return findMedianSortedArrays(nums2, nums1); // Ensure nums1 is the smaller array
  }

  int m = nums1.size();
  int n = nums2.size();
  int low = 0, high = m;

  while (low <= high)
  {
    int partitionX = (low + high) / 2;
    int partitionY = (m + n + 1) / 2 - partitionX;

    int maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
    int minX = (partitionX == m) ? INT_MAX : nums1[partitionX];

    int maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];
    int minY = (partitionY == n) ? INT_MAX : nums2[partitionY];

    if (maxX <= minY && maxY <= minX)
```

```
    {
      if ((m + n) % 2 == 0)
      {
        return (max(maxX, maxY) + min(minX, minY)) / 2.0;
      }
      else
      {
        return max(maxX, maxY);
      }
    }
    else if (maxX > minY)
    {
      high = partitionX - 1;
    }
    else
    {
      low = partitionX + 1;
    }
  }

  throw invalid_argument("Input arrays are not sorted or invalid.");
}

int main()
{
  int m, n;
  cout << "Enter the size of the first array: ";
  cin >> m;
  vector<int> nums1(m);
  cout << "Enter the elements of the first sorted array:\n";
  for (int i = 0; i < m; ++i)
  {
    cin >> nums1[i];
  }

  cout << "Enter the size of the second array: ";
  cin >> n;
  vector<int> nums2(n);
  cout << "Enter the elements of the second sorted array:\n";
  for (int i = 0; i < n; ++i)
  {
    cin >> nums2[i];
  }

  double median = findMedianSortedArrays(nums1, nums2);
  cout << "The median of the two sorted arrays is: " << median << endl;

  return 0;
}
```

**Output:-**


Enter the size of the first array: 3
Enter the elements of the first sorted array:
1 2 3
Enter the size of the second array: 2
Enter the elements of the second sorted array:
4 5
The median of the two sorted arrays is: 3
PS C:\Users\adity\OneDrive\Desktop\WINTER DOMAIN CAMP\DAY 5>