

Name: Tamia Naeem

Roll Number: AI-004

Subject: DSA

Course Code: CT-159

Exercise

1. Write a C++ program to copy data of a 2D array in a 1D array using Column Major Order.

Source Code

```
#include<iostream>

using namespace std;

int main () {
int a1[2][2] = {1,3,5,7};

int a2[4];

int k=0;

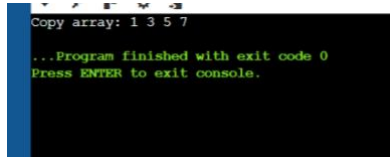
for (int i=0; i<2; i++) {
    for (int j=0; j<2; j++) {
a2[k] = a1[i][j];
k++;
    }
}

cout << "Copy array: " << endl;

for (int i=0; i<4; i++) {
cout << a2[i] << " ";
}

    return 0;
}
```

Output



2. Write a program to calculate the GPA of students of all subjects of a single semester. Assume all the courses have the same credit hour (let's assume 3 credit hours).

	Data Structure	Programming for AI	Digital Logic Design	Probability and Statistics	Finance and Accounting
Ali	3.66	3.33	4.0	3.0	2.66
Hiba	3.33	3.0	3.66	3.0	--
Asma	4.0	3.66	2.66	--	--
Zain	2.66	2.33	4.0	--	--
Faisal	3.33	3.66	4.0	3.0	3.33

Source Code

```
#include <iostream>

#include <string>

using namespace std;

float CalculateGpa(float total, int NoCourses) {

    return total / (NoCourses * 3);

}

int main() {

    string a1[5] = {"Ali", "Hiba", "Asma", "Zain", "Faisal"};

    float gpa[5][5] = {

        {3.66, 3.33, 4.0, 3.0, 2.66},

        {3.33, 3.0, 3.66, 3.0, -1},

        {4.0, 3.66, 2.66, -1, -1},

        {2.66, 2.33, 4.0, -1, -1},

        {3.33, 3.66, 4.0, 3.0, 3.33}
```

```

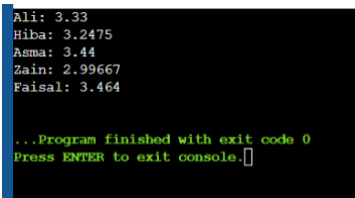
};

for (int i = 0; i < 5; i++) {
    int n = 0;
    float t = 0;
    for (int j = 0; j < 5; j++) {
        if (gpa[i][j] != -1) { // Checking if the grade is valid
            t += gpa[i][j] * 3; // Multiply GPA by 3 (assuming each course has 3 credit hours)
            n++;
        }
    }
    cout << a1[i] << ": " << CalculateGpa(t, n) << endl;
}

return 0;
}

```

Output



```

Ali: 3.33
Hiba: 3.2475
Asma: 3.44
Zain: 2.99667
Faisal: 3.464

...Program finished with exit code 0
Press ENTER to exit console.

```

3. The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

For example, for arr = [2,3,4], the median is 3.

For example, for arr = [2,3], the median is $(2 + 3) / 2 = 2.5$.

Implement the MedianFinder class:

MedianFinder() initializes the MedianFinder object.

void addNum(int num) adds the integer num from the data stream to the data structure.

double findMedian() returns the median of all elements so far. Answers within 10⁻⁵ of the actual answer will be accepted.

Example 1:

Input: ["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]

[[], [1], [2], [], [3], []]

Output: [null, null, null, 1.5, null, 2.0]

Explanation

```
MedianFinder medianFinder = new MedianFinder(); medianFinder.addNum(1); // arr = [1]
medianFinder.addNum(2); // arr = [1, 2] medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3); // arr[1, 2, 3] medianFinder.findMedian(); // return 2.0
```

Constraints: -105 ≤ num ≤ 105

There will be at least one element in the data structure before calling findMedian. At most 5 * 10⁴ calls will be made to addNum and findMedian.

Source Code

```
#include <iostream>

#include <algorithm>

using namespace std;

class MedianFinder {
private:
    int nums[50000];
    int size;

public:
    MedianFinder() : size(0) {}

    void addNum(int num) {
        int pos = upper_bound(nums, nums + size, num) - nums;
        for (int i = size; i > pos; --i) {
```

```

        nums[i] = nums[i - 1];
    }
    nums[pos] = num;
    size++;
}

double findMedian() {
    if (size % 2 == 1) {
        return nums[size / 2];
    } else {
        return (nums[size / 2 - 1] + nums[size / 2]) / 2.0;
    }
}

};

int main() {
    MedianFinder medianFinder;
    medianFinder.addNum(1);
    medianFinder.addNum(2);
    cout << medianFinder.findMedian() << endl;
    medianFinder.addNum(3);
    cout << medianFinder.findMedian() << endl;
    return 0;
}

```

Output

```

1.5
2
...Program finished with exit code 0
Press ENTER to exit console.

```

4. Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return -1. You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1: Input: `nums = [-1,0,3,5,9,12]`, `target = 9`, Output: 4

Explanation: 9 exists in `nums` and its index is 4

Example 2: Input: `nums = [-1,0,3,5,9,12]`, `target = 2`, Output: -1 Explanation: 2 does not exist in `nums` so return -1 Constraints:

$1 \leq \text{nums.length} \leq 104$

$-104 < \text{nums}[i], \text{target} < 104$

All the integers in `nums` are unique.

Source Code

```
#include <iostream>

#include <algorithm>

using namespace std;

bool SearchAlgo(int nums[], int size, int target) {

    int start = 0, end = size - 1;

    while (start <= end) {

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

        if (nums[mid] == target) return true;

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

        else end = mid - 1;

    }

    return false;

}

int main() {

    int nums[] = {1, 7, 5, 9, 44, 12, 4, 80, 123};
```

```

int size = sizeof(nums) / sizeof(nums[0]);

sort(nums, nums + size);

int target1 = 78, target2 = 44;

bool ans1 = SearchAlgo(nums, size, target1);

bool ans2 = SearchAlgo(nums, size, target2);

cout << "Target " << target1 << (ans1 ? " is found." : " is not found.") << endl;

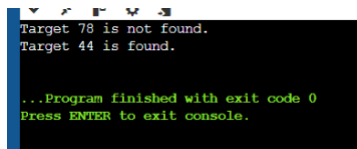
cout << "Target " << target2 << (ans2 ? " is found." : " is not found.") << endl;

return 0;

}

```

Output



```

Target 78 is not found.
Target 44 is found.

...Program finished with exit code 0
Press ENTER to exit console.

```

5. nums is sorted in ascending order. You are given an $m \times n$ integer matrix with the following two properties: Each row is sorted in non-decreasing order. The first integer of each row is greater than the last integer of the previous row. Given an integer target, return true if target is in matrix or false otherwise. You must write a solution in $O(\log(m * n))$ time complexity.

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3, Output: true Constraints: $m == \text{matrix.length}$, $n == \text{matrix}[i].\text{length}$, $1 \leq m$, $n \leq 100$, $-104 \leq \text{matrix}[i][j]$, $\text{target} \leq 104$

Source Code

```

#include <iostream>

using namespace std;

bool searchMatrix(int matrix[][4], int m, int n, int target) {

    int start = 0, end = m * n - 1;

    while (start <= end) {

```

```

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

        int mid_value = matrix[mid / n][mid % n];

        if (mid_value == target) return true;

        else if (mid_value < target) start = mid + 1;

        else end = mid - 1;

    }

    return false;
}

int main() {

    int matrix[3][4] = {

        {1, 3, 5, 7},

        {10, 11, 16, 20},

        {23, 30, 34, 60}

    };

    int target = 3;

    bool result = searchMatrix(matrix, 3, 4, target);

    if (result) {

        cout << "Target found in matrix." << endl;

    } else {

        cout << "Target not found in matrix." << endl;

    }

    return 0;

}

```

Output

Target found in matrix.

...Program finished with exit code 0
Press ENTER to exit console.