Tamia Naeem
AI-004

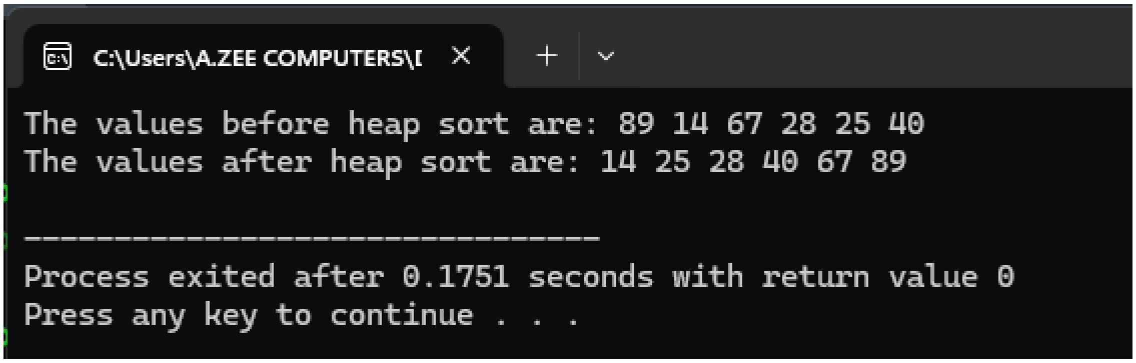
EXERCISE:

1. Implement Heap Sort by using the steps listed in the Lab manual on Page#05.

SOURCE CODE:

```
#include <iostream>

#include <vector>

using namespace std;

class Tamia_004 {

public:

    void heapSort(vector<int>& n) {

        int s = n.size();

        for (int i = s/2 - 1; i >= 0; i--) {

            heapify(n, s, i);

        }

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

            swap(n[0], n[i]);

            heapify(n, i, 0);

        }

    }

private:
    void heapify(vector<int>& n, int s, int root) {
        int large = root;
        int left = 2 * root + 1;
        int right = 2 * root + 2;
        if (left < s && n[left] > n[large]) large = left;
        if (right < s && n[right] > n[large]) large = right;
        if (large != root) {
            swap(n[root], n[large]);
            heapify(n, s, large);
        }
    }
};
int main() {
    Tamia_004 T;
    vector<int> n;
    n.push_back(89);
    n.push_back(14);
    n.push_back(67);
    n.push_back(28);
    n.push_back(25);

    n.push_back(40);



    cout << "The values before heap sort are: ";

    for (int i = 0; i < n.size(); i++)

        cout << n[i] << " ";

    cout << endl;

    T.heapSort(n);

    cout << "The values after heap sort are: ";

    for (int i = 0; i < n.size(); i++)

        cout << n[i] << " ";

    cout << endl;

    return 0;

}
```

OUTPUT:



```
The values before heap sort are: 89 14 67 28 25 40
The values after heap sort are: 14 25 28 40 67 89

--------------------------------
Process exited after 0.1751 seconds with return value 0
Press any key to continue . . .
```

2. Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

Example 1: Input: nums = [1,1,1,2,2,3], k = 2, Output: [1,2]

Example 2: Input: nums = [1], k = 1, Output: [1]

SOURCE CODE:

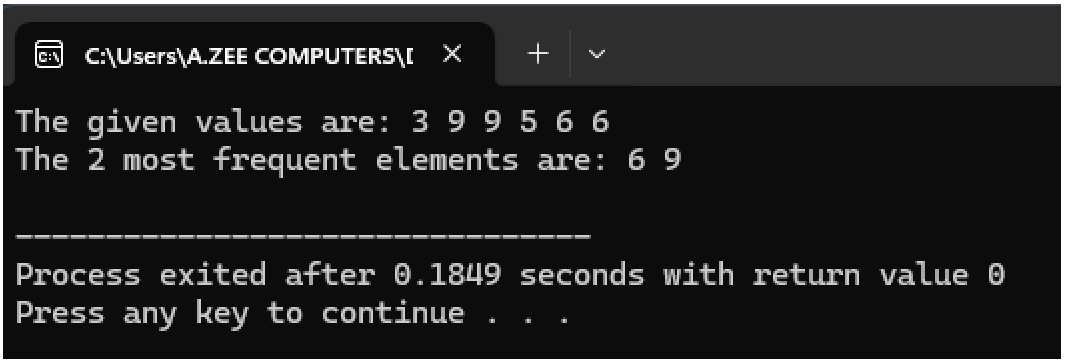
```
#include <iostream>

#include <vector>

using namespace std;

class Tamia_004 {

public:

   vector<int> FQ(vector<int>& nums, int k) {

       vector<int> result;

       if (nums.empty() || k == 0) return result;

       int maxFreq = 0;

       vector<int> frequency(1001, 0);
```

```
        for (int num : nums) {
            frequency[num]++;
            maxFreq = max(maxFreq, frequency[num]);
        }
        vector<vector<int>> QF(maxFreq + 1);
        for (int i = 0; i < frequency.size(); i++) {
            if (frequency[i] > 0) {
                QF[frequency[i]].push_back(i);
            }
        }
        for (int i = maxFreq; i >= 0; i--) {
            if (QF[i].empty()) continue;
            for (int num : QF[i]) {
                result.push_back(num);
                if (result.size() == k) return result;
            }
        }
        return result;
    }
};
int main() {
```

Tamia Naeem
AI-004

```
    Tamia_004 T;

    vector<int> n;

    int values[] = {3, 9, 9, 5, 6, 6};

    for (int i = 0; i < 6; i++) {

        n.push_back(values[i]);

    }

     int k = 2;

     cout << "The given values are: ";

    for (int i = 0; i < n.size(); i++)

        cout << n[i] << " ";

    cout << endl;

     vector<int> result = T.FQ(n, k);

    cout << "The " << k << " most frequent elements are: ";

    for (int i = 0; i < result.size(); i++) {

        cout << result[i] << " ";

    }

    cout << endl;

    return 0;

}
```

OUTPUT:



```
The given values are: 3 9 9 5 6 6
The 2 most frequent elements are: 6 9

--------------------------------
Process exited after 0.1849 seconds with return value 0
Press any key to continue . . .
```

3. Given a string s, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string. Return the sorted string.If there are multiple answers, return any of them.

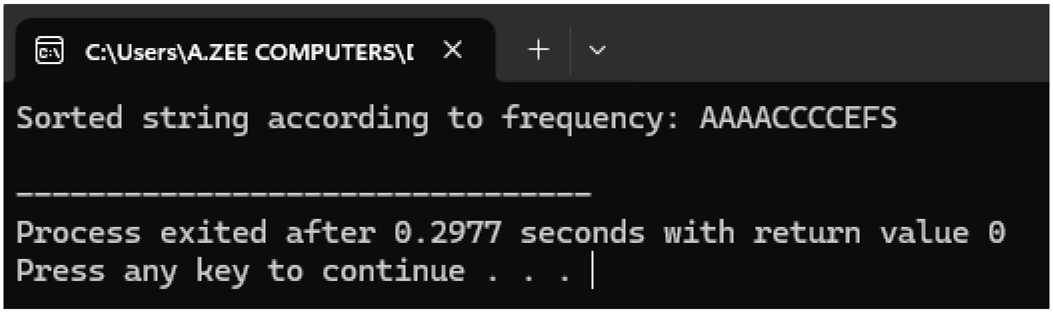
Example 1: Input: s = "tree", Output: "eert"

Example 2: Input: s = "cccaaa", Output: "aaaccc"

SOURCE CODE:

```
#include <iostream>

#include <vector>

#include<algorithm>

using namespace std;

class Tamia_004 {

public:

  string FQSort(string str) {

    vector<int> count(256, 0);

    for (char ch : str) {
        count[ch]++;
    }
    vector<pair<int, char>> freqChars;
    for (int i = 0; i < 256; i++) {
        if (count[i] > 0) {
            freqChars.push_back({count[i], char(i)});
        }
    }
    sort(freqChars.begin(), freqChars.end(), [](pair<int, char> a, pair<int, char> b) {
        return a.first > b.first;
    });
    string sortedStr;
    for (auto& [freq, ch] : freqChars) {
        sortedStr += string(freq, ch);
    }
    return sortedStr;
  }
};
int main() {
  Tamia_004 T;
  string input = "CFCSCECAAAA"";
  string output = T.FQSort(input);

  cout << "Sorted string according to frequency: " << output << endl;

  return 0;

}
```

OUTPUT:



```
Sorted string according to frequency: AAAACCCCEFS

--------------------------------
Process exited after 0.2977 seconds with return value 0
Press any key to continue . . .
```

4. There are n workers. You are given two integer arrays quality and wage where quality[i] is the quality of the it h worker and wage[i] is the minimum wage expectation for the ith worker. We want to hire exactly k workers to form a paid group. To hire a group of k workers, we must pay them according to the following rules: Every worker in the paid group must be paid at least theirminimum wage expectation. In the group, each worker's pay must be directly proportional to their quality. This means if a worker’s quality is double that of another worker in the group, then they must be paid twice as much as the other worker. Given the integer k, return the least amount of money needed to form a paid group satisfying the above conditions. Answers within 10^-5 of the actual answer will be accepted.

Example 1: Input: quality = [10,20,5], wage = [70,50,30], k = 2, Output: 105.00000

Explanation: We pay 70 to 0th worker and 35 to 2nd worker.

Example 2: Input: quality = [3,1,10,10,1], wage = [4,8,2,2,7], k = 3, Output: 30.66667

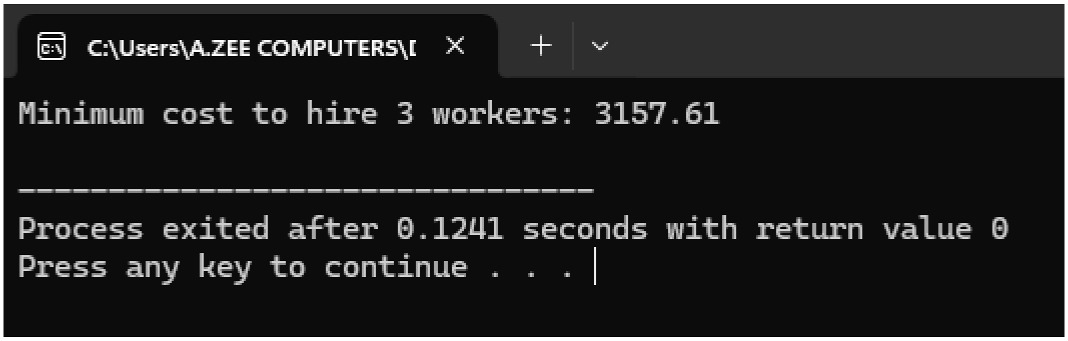
Explanation: We pay 4 to 0th worker, 13.33333 to 2nd and 3rd workers separately.

SOURCE CODE:

Tamia Naeem
AI-004

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

using namespace std;

class Tamia_004 {

public:

    double cost(vector<int>& q, vector<int>& wage, int k) {

        int n = q.size();

        vector<double> ratios(n);

        // Calculate the wage-to-quality ratio for each worker

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

            ratios[i] = (double)wage[i] / q[i];

        }

        // Sort workers based on their wage-to-quality ratio

        vector<int> indices(n);

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

            indices[i] = i;

        }

        sort(indices.begin(), indices.end(), [&](int a, int b) {

            return ratios[a] < ratios[b];

        });

        // Max-heap to maintain the k workers with the highest qualities

        priority_queue<int> maxHeap;
        int sumQ = 0;

        double minCost = INT_MAX;

        // Calculate the cost for the first k workers

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

            sumQ += q[indices[i]];

            maxHeap.push(q[indices[i]]);

        }

        minCost = ratios[indices[k - 1]] * sumQ;

        // Iterate through the remaining workers and adjust the heap

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

            sumQ -= maxHeap.top();  // Remove the largest quality from the heap

            maxHeap.pop();

            sumQ += q[indices[i]];  // Add the current worker's quality

            maxHeap.push(q[indices[i]]);

            // Calculate and update the minimum cost

            minCost = min(minCost, ratios[indices[i]] * sumQ);

        }

        return minCost;

    }

};

int main() {
    Tamia_004 T;

    vector<int> q = {95, 76, 84, 67};

    vector<int> wage = {1200, 1150, 1000, 860};

    int k = 3;

    cout << "Minimum cost to hire " << k << " workers: " << T.cost(q, wage, k) << endl;

    return 0;

}
```

OUTPUT:



```
Minimum cost to hire 3 workers: 3157.61

--------------------------------
Process exited after 0.1241 seconds with return value 0
Press any key to continue . . .
```

5. The median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values. For examples, if arr = [2,3,4], the median is 3. For examples, if arr = [1,2,3,4], the median is (2 + 3) / 2 = 2.5. You are given an integer array nums and an integer k. There is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the median array for each window in the original array. Answers within 10^-5 of the actual value will be accepted.

Example 1: Input: nums = [1,3,-1,-3,5,3,6,7], k = 3,

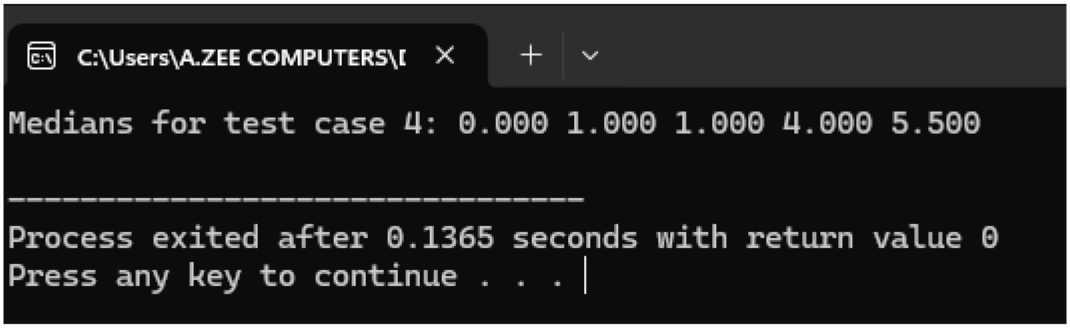
Output: [1.00000,-1.00000,-1.00000,3.00000,5.00000,6.00000]

SOURCE CODE:

Tamia Naeem
AI-004

```
#include <bits/stdc++.h>
using namespace std;
class Tamia_004 {
public:
    vector<double> SlidingWindow(vector<int>& nums, int k) {
        vector<double> medians;
        priority_queue<int> maxHeap; // Max-heap for the lower half
        priority_queue<int, vector<int>, greater<int>> minHeap; // Min-heap for the upper half
        // Lambda to rebalance the heaps
        auto balanceHeaps = [&]() {
            if (maxHeap.size() > minHeap.size() + 1) {
                minHeap.push(maxHeap.top());
                maxHeap.pop();
            } else if (minHeap.size() > maxHeap.size()) {
                maxHeap.push(minHeap.top());
                minHeap.pop();
            }
        };
        // Lambda to add a number to the heaps
        auto addNum = [&](int num) {
            if (maxHeap.empty() || num <= maxHeap.top()) {
                maxHeap.push(num);
            } else {
                minHeap.push(num);
            }
            balanceHeaps();
        };
        // Lambda to remove a number from the heaps
        auto removeNum = [&](int num) {
            if (!maxHeap.empty() && num <= maxHeap.top()) {
                // Create a temporary max heap without the number to be removed
                priority_queue<int> temp;
                while (!maxHeap.empty()) {
                    if (maxHeap.top() != num) temp.push(maxHeap.top());
                    maxHeap.pop();
                }
                maxHeap = temp;
            } else {
                // Create a temporary min heap without the number to be removed
                priority_queue<int, vector<int>, greater<int>> temp;
                while (!minHeap.empty()) {
                    if (minHeap.top() != num) temp.push(minHeap.top());
                    minHeap.pop();
                }
                minHeap = temp;
            }
            balanceHeaps();
        };
        // Process each sliding window
        for (int i = 0; i < nums.size(); i++) {
            addNum(nums[i]);
            if (i >= k - 1) {
                // Calculate median
                if (maxHeap.size() > minHeap.size()) {
                    medians.push_back(maxHeap.top());
                } else {
                    medians.push_back((maxHeap.top() + minHeap.top()) / 2.0);
                }
                // Remove the element going out of the sliding window
                removeNum(nums[i - k + 1]);
            }
        }
        return medians;
    }
};
int main() {
    Tamia_004 T;
    vector<int> n = {1, 3, -1, -3, 5, 3, 6, 7};
    int Size = 4;
    vector<double> median = T.SlidingWindow(n, Size);
    cout << "Medians for test case 4: ";
    for (double med : median) {
        cout << fixed << setprecision(3) << med << " ";
    }
    cout << endl;
    return 0;

}
```

OUTPUT:



```
Medians for test case 4: 0.000 1.000 1.000 4.000 5.500

--------------------------------
Process exited after 0.1365 seconds with return value 0
Press any key to continue . . .
```