# Assignment 1 DAA

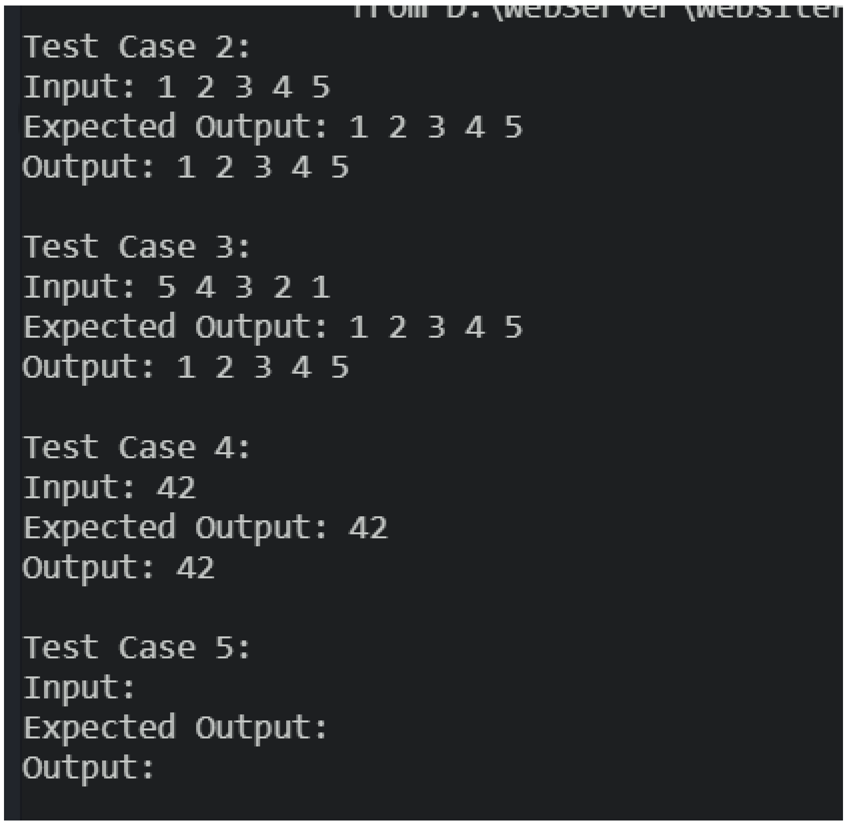
Name: Kshitij Chandrakar
Batch: 49
SAP: 500124827

## Q2:

## Code:

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

const int TotalUsers = 4;

void allocateBandwidth(const string users[], const int userBandwidth[], const
int userValues[], const int totalBandwidth) {
    double VBRatio[TotalUsers];

    // Calculate value-to-bandwidth ratios
    for (int i = 0; i < TotalUsers; ++i) {
        VBRatio[i] = static_cast<double>(userValues[i]) / userBandwidth[i];
    }

    // Sort users by value-to-bandwidth ratio
    for (int i = 0; i < TotalUsers - 1; ++i) {
        for (int j = 0; j < TotalUsers - i - 1; ++j) {
            if (VBRatio[j] < VBRatio[j + 1]) {
                swap(VBRatio[j], VBRatio[j + 1]);
                swap(users[j], users[j + 1]);
                swap(userBandwidth[j], userBandwidth[j + 1]);
                swap(userValues[j], userValues[j + 1]);
            }
        }
    }

    int remainingBandwidth = totalBandwidth;
    double totalValue = 0.0;
    double allocation[TotalUsers] = {0};

    // Allocate bandwidth
    for (int i = 0; i < TotalUsers; ++i) {
        if (remainingBandwidth > 0) {
            double allocated = min(static_cast<double>(userBandwidth[i]),
static_cast<double>(remainingBandwidth));
            totalValue += (allocated / userBandwidth[i]) * userValues[i];
            allocation[i] = allocated;
            remainingBandwidth -= allocated;
        }
    }

    // Print the results
    cout << "Maximum Value Achieved: " << totalValue << endl;
    cout << "Bandwidth Allocation:" << endl;
    for (int i = 0; i < TotalUsers; ++i) {
        cout << "User " << users[i] << ": " << allocation[i] << " MB" << endl;
    }
}

int main() {
    vector<tuple<string[TotalUsers], int[TotalUsers], int[TotalUsers], int,
double, vector<double>>> testCases = {
        {
            {"U1", "U2", "U3", "U4"},
            {40, 70, 30, 50},
            {50, 90, 45, 60},
            100,
            96.4286,
            {40, 60, 0, 0} // Expected allocations
        },
        {
            {"U1", "U2", "U3", "U4"},
            {40, 70, 30, 50},
            {50, 90, 45, 60},
            50,
            49.2857,
            {40, 10, 0, 0} // Expected allocations
        },
        {
            {"U1", "U2", "U3", "U4"},
            {40, 70, 30, 50},
            {50, 90, 45, 60},
            70,
            70.0,
            {40, 30, 0, 0} // Expected allocations
        },
        {
            {"U1", "U2", "U3", "U4"},
            {40, 70, 30, 50},
            {50, 90, 45, 60},
            30,
            30.0,
            {30, 0, 0, 0} // Expected allocations
        },
        {
            {"U1", "U2", "U3", "U4"},
            {40, 70, 30, 50},
            {50, 90, 45, 60},
            0,
            0.0,
            {0, 0, 0, 0} // Expected allocations
        }
    };

    for (size_t i = 0; i < testCases.size(); ++i) {
        string users[TotalUsers];
        int userBandwidth[TotalUsers];
        int userValues[TotalUsers];
        int totalBandwidth;
        double expectedValue;
        vector<double> expectedAllocations;

        tie(users, userBandwidth, userValues, totalBandwidth, expectedValue,
expectedAllocations) = testCases[i];

        cout << "Test Case " << i + 1 << ":\n";

        cout << "Input:\n";
        cout << "Users: ";
        for (const auto& user : users) {
            cout << user << " ";
        }
        cout << "\nUser Bandwidth: ";
        for (const auto& bw : userBandwidth) {
            cout << bw << " ";
        }
        cout << "\nUser Values: ";
        for (const auto& value : userValues) {
            cout << value << " ";
        }
        cout << "\nTotal Bandwidth: " << totalBandwidth << endl;
        allocateBandwidth(users, userBandwidth, userValues, totalBandwidth);
        cout << "Expected Output: Maximum Value Achieved: " << expectedValue
<< endl;
        cout << "Expected Bandwidth Allocation: ";
        for (const auto& alloc : expectedAllocations) {
            cout << alloc << " MB ";
        }
        cout << endl;
        cout << endl;
    }
    return 0;
}
```

## Screenshot:



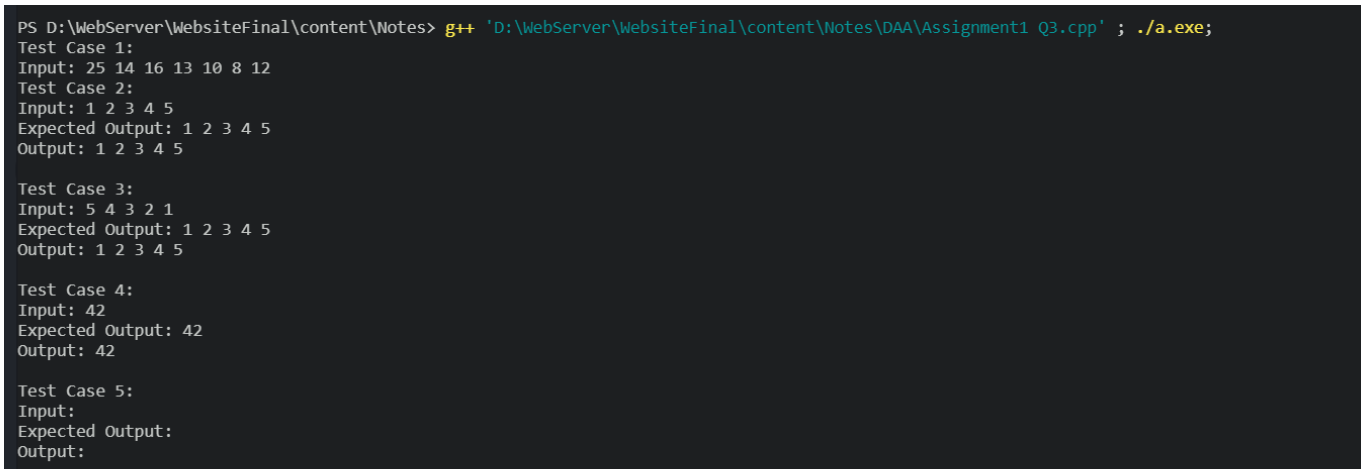
```
Test Case 2:
Input: 1 2 3 4 5
Expected Output: 1 2 3 4 5
Output: 1 2 3 4 5

Test Case 3:
Input: 5 4 3 2 1
Expected Output: 1 2 3 4 5
Output: 1 2 3 4 5

Test Case 4:
Input: 42
Expected Output: 42
Output: 42

Test Case 5:
Input:
Expected Output:
Output:
```

## Q3:

## Code:

```cpp
#include <iostream>
#include <vector>
using namespace std;
void insertionSort(vector<int>& arr) {
    for (size_t i = 1; i < arr.size(); ++i) {
        int key = arr[i];
        size_t j = i - 1;

        while (j < arr.size() && j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
// Function to print an array
void printArray(const vector<int>& arr) {
    for (const int& num : arr) {
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    vector<pair<vector<int>, vector<int>>> testCases = {
        {{25, 14, 16, 13, 10, 8, 12}, {8, 10, 12, 13, 14, 16, 25}},  // Random
unsorted array
        {{1, 2, 3, 4, 5}, {1, 2, 3, 4, 5}},                         // Already
sorted array
        {{5, 4, 3, 2, 1}, {1, 2, 3, 4, 5}},                         // Reverse
sorted array
        {{42}, {42}},                                                // Single
element array
        {{}, {}}                                                     // Empty
array
    };

    for (size_t i = 0; i < testCases.size(); ++i) {
        vector<int> inputArray = testCases[i].first;
        vector<int> expectedOutput = testCases[i].second;
        cout << "Test Case " << i + 1 << ":\n";
        cout << "Input: ";
        printArray(inputArray);
        insertionSort(inputArray);
        cout << "Expected Output: ";
        printArray(expectedOutput);
        cout << "Output: ";
        printArray(inputArray);

        cout << endl;
    }

    return 0;
}
```

## Screenshot:



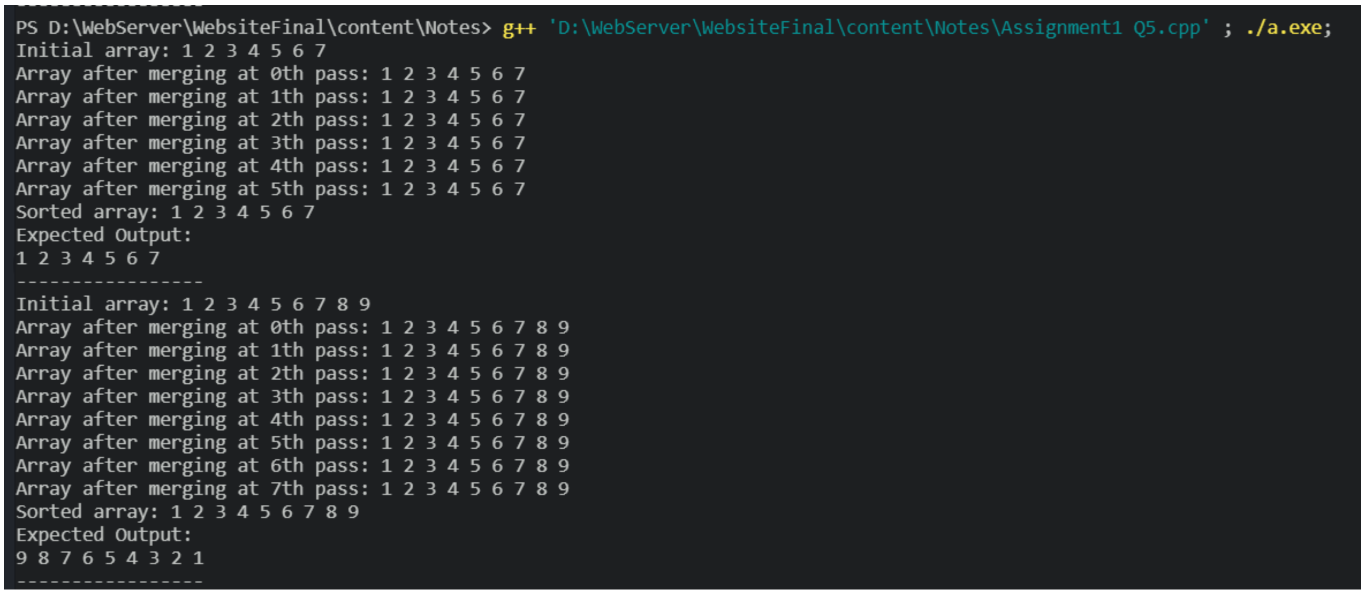
```
PS D:\WebServer\WebsiteFinal\content\Notes> g++ 'D:\WebServer\WebsiteFinal\content\Notes\DAA\Assignment1 Q3.cpp' ; ./a.exe;
Test Case 1:
Input: 25 14 16 13 10 8 12
Test Case 2:
Input: 1 2 3 4 5
Expected Output: 1 2 3 4 5
Output: 1 2 3 4 5

Test Case 3:
Input: 5 4 3 2 1
Expected Output: 1 2 3 4 5
Output: 1 2 3 4 5

Test Case 4:
Input: 42
Expected Output: 42
Output: 42

Test Case 5:
Input:
Expected Output:
Output:
```

## Q5:

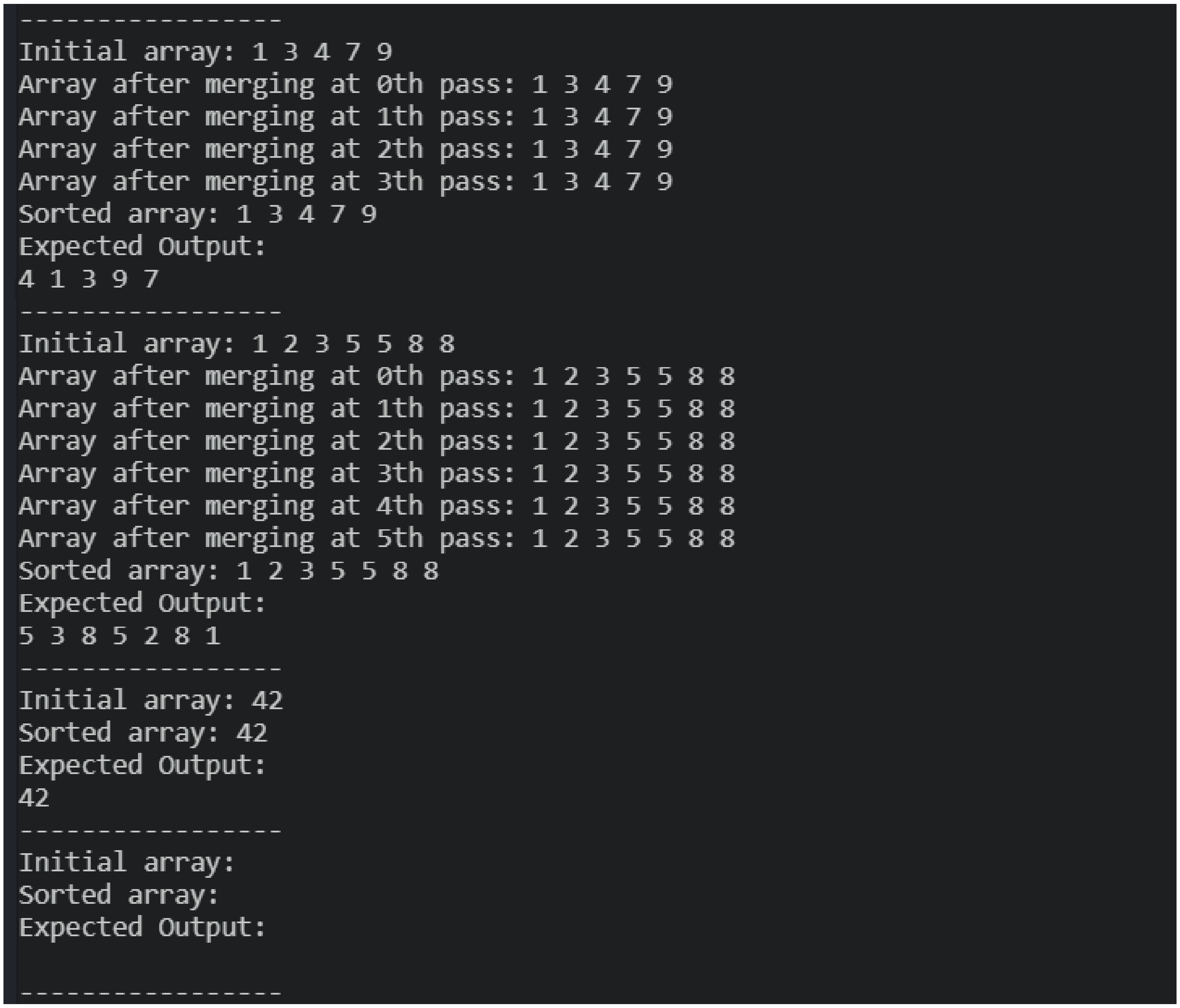
## Code:

```cpp
#include <iostream>
#include <vector>
int pass = 1;
void merge(std::vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    std::vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0;
    int j = 0;
    int k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    std::cout << "Array after merging at " << pass << "th pass: ";
    pass++;
    for (int m = 0; m < arr.size(); m++) {
        std::cout << arr[m] << " ";
    }
    std::cout << std::endl;
}

void mergeSort(std::vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}


int testcases( std::vector<int> arr){
  std::cout << "Initial array: ";
  for (const int& num : arr) {
      std::cout << num << " ";
  }
  std::cout << std::endl;

  mergeSort(arr, 0, arr.size() - 1);

  std::cout << "Sorted array: ";
  for (const int& num : arr) {
      std::cout << num << " ";
  }
  std::cout << std::endl;

  return 0;
}
int main() {
  std::vector<std::pair<std::vector<int>, std::vector<int>>> testCases = {
      {{1, 2, 3, 4, 5, 6, 7}, {1, 2, 3, 4, 5, 6, 7}},
      {{9, 8, 7, 6, 5, 4, 3, 2, 1}, {1, 2, 3, 4, 5, 6, 7, 8, 9}},
      {{4, 1, 3, 9, 7}, {1, 3, 4, 7, 9}},
      {{5, 3, 8, 5, 2, 8, 1}, {1, 2, 3, 5, 5, 8, 8}},
      {{42}, {42}},
      {{}, {}}
  };
    int count = 6;
    for (size_t i = 0; i < count; i++) {
      pass = 0;
      testcases(testCases[i].second);
      std::cout << "Expected Output:" << '\n';
      for (const int& num : testCases[i].first) {
          std::cout << num << " ";
      }
      std::cout << "\n-----------------\n";
    }
        return 0;
}
```

## Screenshot:



```
PS D:\WebServer\WebsiteFinal\content\Notes> g++ 'D:\WebServer\WebsiteFinal\content\Notes\Assignment1 Q5.cpp' ; ./a.exe;
Initial array: 1 2 3 4 5 6 7
Array after merging at 0th pass: 1 2 3 4 5 6 7
Array after merging at 1th pass: 1 2 3 4 5 6 7
Array after merging at 2th pass: 1 2 3 4 5 6 7
Array after merging at 3th pass: 1 2 3 4 5 6 7
Array after merging at 4th pass: 1 2 3 4 5 6 7
Array after merging at 5th pass: 1 2 3 4 5 6 7
Sorted array: 1 2 3 4 5 6 7
Expected Output:
1 2 3 4 5 6 7
----------------
Initial array: 1 2 3 4 5 6 7 8 9
Array after merging at 0th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 1th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 2th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 3th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 4th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 5th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 6th pass: 1 2 3 4 5 6 7 8 9
Array after merging at 7th pass: 1 2 3 4 5 6 7 8 9
Sorted array: 1 2 3 4 5 6 7 8 9
Expected Output:
9 8 7 6 5 4 3 2 1
----------------
```



```
----------------
Initial array: 1 3 4 7 9
Array after merging at 0th pass: 1 3 4 7 9
Array after merging at 1th pass: 1 3 4 7 9
Array after merging at 2th pass: 1 3 4 7 9
Array after merging at 3th pass: 1 3 4 7 9
Sorted array: 1 3 4 7 9
Expected Output:
4 1 3 9 7
----------------
Initial array: 1 2 3 5 5 8 8
Array after merging at 0th pass: 1 2 3 5 5 8 8
Array after merging at 1th pass: 1 2 3 5 5 8 8
Array after merging at 2th pass: 1 2 3 5 5 8 8
Array after merging at 3th pass: 1 2 3 5 5 8 8
Array after merging at 4th pass: 1 2 3 5 5 8 8
Array after merging at 5th pass: 1 2 3 5 5 8 8
Sorted array: 1 2 3 5 5 8 8
Expected Output:
5 3 8 5 2 8 1
----------------
Initial array: 42
Sorted array: 42
Expected Output:
42
----------------
Initial array:
Sorted array:
Expected Output:

----------------
```