22 OCTOBER 2025

# DAA PROJECT PART 1

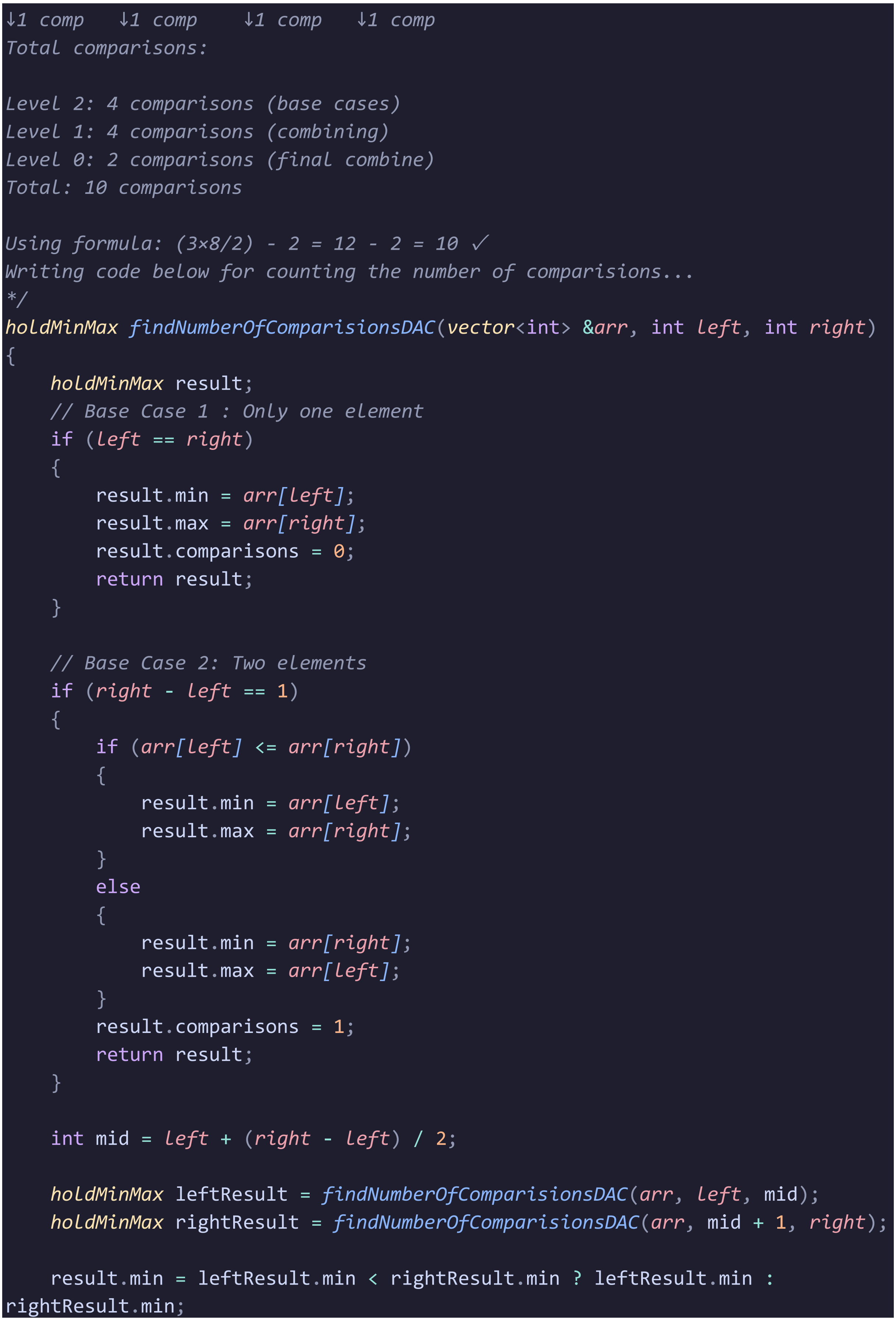
## INSTRUCTOR: SIR MUHAMMAD KASHIF

IBAD UR REHMAN
MUHAMMAD MUSTAFA
AAQIB ZAHID

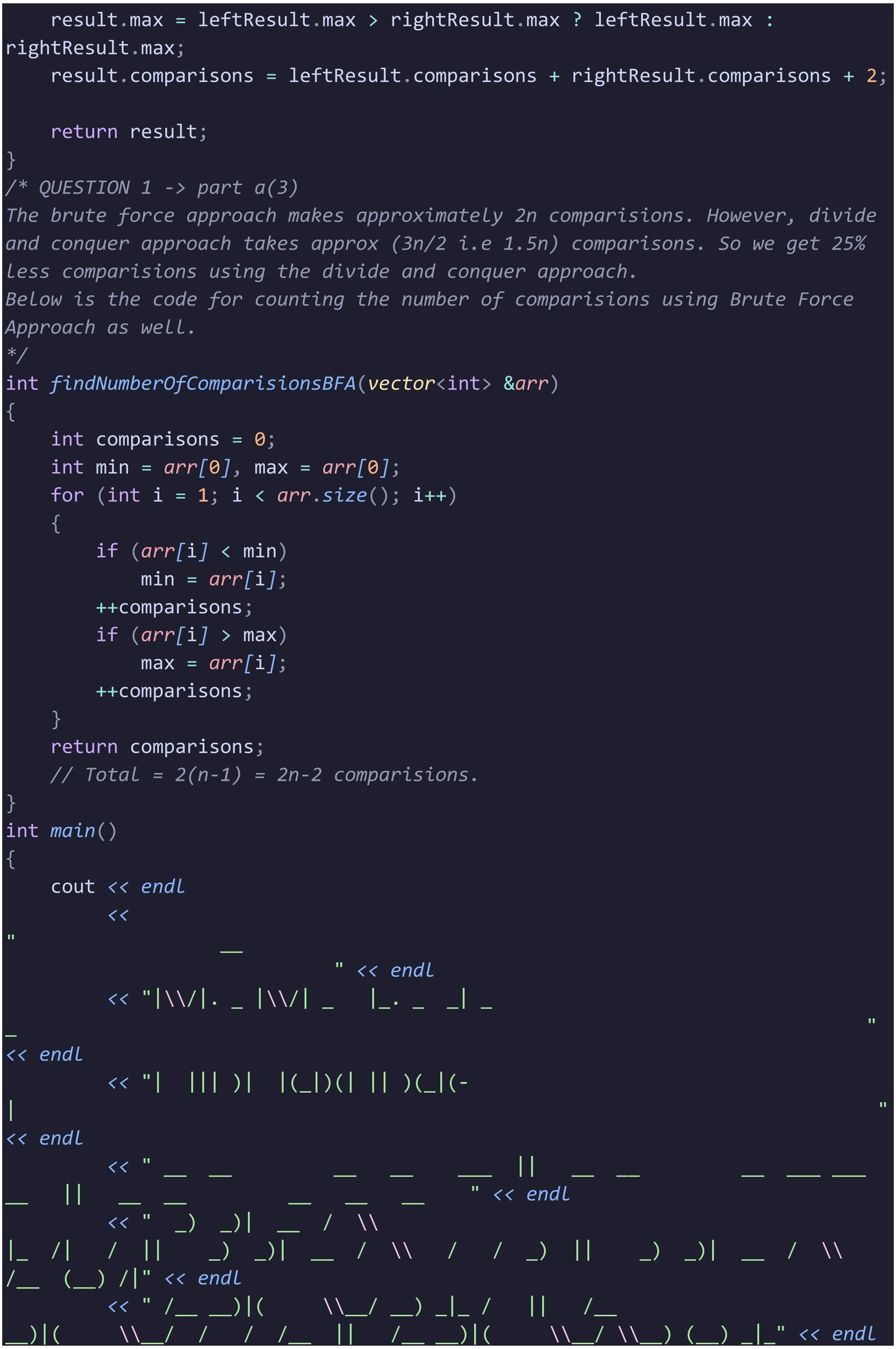
# All Codes and their Outputs

## Part a

```
#include <iostream>
#include <vector>

using namespace std;

struct holdMinMax
{
    int min;
    int max;
    int comparisons;
};

/* QUESTION 1 -> part a(1) */

holdMinMax findMinMax(vector<int> &arr, int left, int right)
{
    holdMinMax result;

    // Base Case 1 : Only one element
    if (left == right)
    {
        result.min = arr[left];
        result.max = arr[right];
        return result;
    }

    // Base Case 2: Two elements
    if (right - left == 1)
    {
        if (arr[left] <= arr[right])
        {
            result.min = arr[left];
            result.max = arr[right];
            return result;
        }
        else
        {
            result.min = arr[right];
            result.max = arr[left];
            return result;
        }
    }
    int mid = left + (right - left) / 2;

    // Divide: Find min and max of left and right halves. Similar to what we
do in merge sort.
    holdMinMax leftResult = findMinMax(arr, left, mid);
    holdMinMax rightResult = findMinMax(arr, mid + 1, right);

    // Conquer: i.e combine results and compare. But because there are only 2
comparisions i.e for min and mix, so unlike merge sort , that takes   O(n)
time  for combining, this will take constant i.e O(1) time.

    result.min = leftResult.min < rightResult.min ? leftResult.min :
rightResult.min;
    result.max = leftResult.max > rightResult.max ? leftResult.max :
rightResult.max;

    return result;
}

/* QUESTION 1 -> part a(2)

C(n) = 2C(n/2) + 2 => 2 comparisions in each stage, each of size of n/2 +
additional 2 comparisions for min and max at the end.

For an array of size n = 2^k, the recurrence relation for comparisons is:
C(n) = 2C(n/2) + 2 for n > 2, with C(2) = 1, C(1) = 0
Solving by backward substitution:
Starting with C(2^k):

= 2C(2^(k-1)) + 2
= 2[2C(2^(k-2)) + 2] + 2 = 2²C(2^(k-2)) + 2² + 2
= 2³[2C(2^(k-3)) + 2] + 2² + 2 = 2³C(2^(k-3)) + 2³ + 2² + 2

Continuing this pattern:

= 2^k C(2⁰) + 2^(k-1) + ... + 2² + 2
= 2^(k-1)·C(2) + (2^k - 2 - 2^1)
= 2^(k-1)·1 + 2^k - 2 - 2
= 2^k - 2 + 2^(k-1) - 2 = (3/2)·2^k - 2 = (3n/2) - 2


For n = 8:
             Level 0: [8 elements]
                      ↓ 2 comparisons
             /                     \
  Level 1: [4 elem]               [4 elem]
            ↓ 2 comp               ↓ 2 comp
         /        \             /        \
Level 2: [2]       [2]         [2]       [2]
```



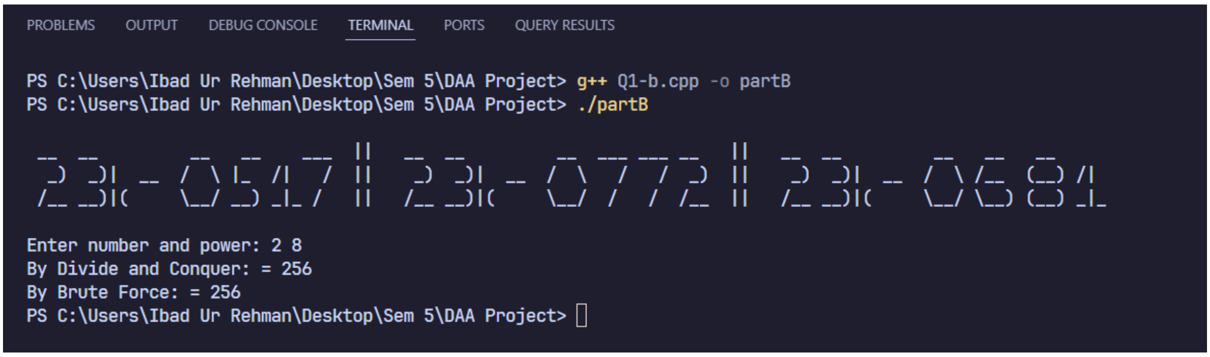
```
↓1 comp   ↓1 comp    ↓1 comp   ↓1 comp
Total comparisons:

Level 2: 4 comparisons (base cases)
Level 1: 4 comparisons (combining)
Level 0: 2 comparisons (final combine)
Total: 10 comparisons

Using formula: (3×8/2) - 2 = 12 - 2 = 10 ✓
Writing code below for counting the number of comparisions...
*/
holdMinMax findNumberOfComparisionsDAC(vector<int> &arr, int left, int right)
{
    holdMinMax result;
    // Base Case 1 : Only one element
    if (left == right)
    {
        result.min = arr[left];
        result.max = arr[right];
        result.comparisons = 0;
        return result;
    }

    // Base Case 2: Two elements
    if (right - left == 1)
    {
        if (arr[left] <= arr[right])
        {
            result.min = arr[left];
            result.max = arr[right];
        }
        else
        {
            result.min = arr[right];
            result.max = arr[left];
        }
        result.comparisons = 1;
        return result;
    }

    int mid = left + (right - left) / 2;

    holdMinMax leftResult = findNumberOfComparisionsDAC(arr, left, mid);
    holdMinMax rightResult = findNumberOfComparisionsDAC(arr, mid + 1, right);

    result.min = leftResult.min < rightResult.min ? leftResult.min :
rightResult.min;
```



```
    result.max = leftResult.max > rightResult.max ? leftResult.max :
rightResult.max;
    result.comparisons = leftResult.comparisons + rightResult.comparisons + 2;

    return result;
}
/* QUESTION 1 -> part a(3)
The brute force approach makes approximately 2n comparisions. However, divide
and conquer approach takes approx (3n/2 i.e 1.5n) comparisons. So we get 25%
less comparisions using the divide and conquer approach.
Below is the code for counting the number of comparisions using Brute Force
Approach as well.
*/
int findNumberOfComparisionsBFA(vector<int> &arr)
{
    int comparisons = 0;
    int min = arr[0], max = arr[0];
    for (int i = 1; i < arr.size(); i++)
    {
        if (arr[i] < min)
            min = arr[i];
        ++comparisons;
        if (arr[i] > max)
            max = arr[i];
        ++comparisons;
    }
    return comparisons;
    // Total = 2(n-1) = 2n-2 comparisions.
}
int main()
{
    cout << endl
         <<
"                   __
                             " << endl
         << "|\\/|. _ |\\/| _   |_. _  _| _
_                                                                         "
<< endl
         << "|  ||| )|  |(_|)(| || )(_|(-
|                                                                          "
<< endl
         << " __  __            __   __     ___  ||   __  __          __  ___ ___
__   ||   __  __        __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(    \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
```

```
         << endl;

    vector<int> inputArr;
    cout << " Enter atleast 10 array elements,press '-1' for end of array: "
<< endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }
    while (true)
    {
        cout << "1. Find Min and Max in Array" << endl
             << "2. Count Number of Comparisions using Divide and Conquer" <<
endl
             << "3. Count Number of Comparions using Brute Force" << endl
             << "4.Exit" << endl;
        int opt;
        cout << "Enter the correct Option Number: ";
        cin >> opt;
        switch (opt)
        {
        case 1:
        {
            holdMinMax res = findMinMax(inputArr, 0, inputArr.size() - 1);
            cout << "Minimum: " << res.min << endl;
            cout << "Maximum: " << res.max << endl;
            cout << endl;
            break;
        }
        case 2:
        {
            holdMinMax res = findNumberOfComparisionsDAC(inputArr, 0,
inputArr.size() - 1);
            cout << "Number of Comparisions: " << res.comparisons << endl;
            break;
        }
        case 3:
        {
            int numberOfComparisionsBFA =
findNumberOfComparisionsBFA(inputArr);
            cout << "Number of Comparisions: " << numberOfComparisionsBFA <<
endl;
            break;
        }
        case 4:
        {
            return 0;
        }
        }
    }
```

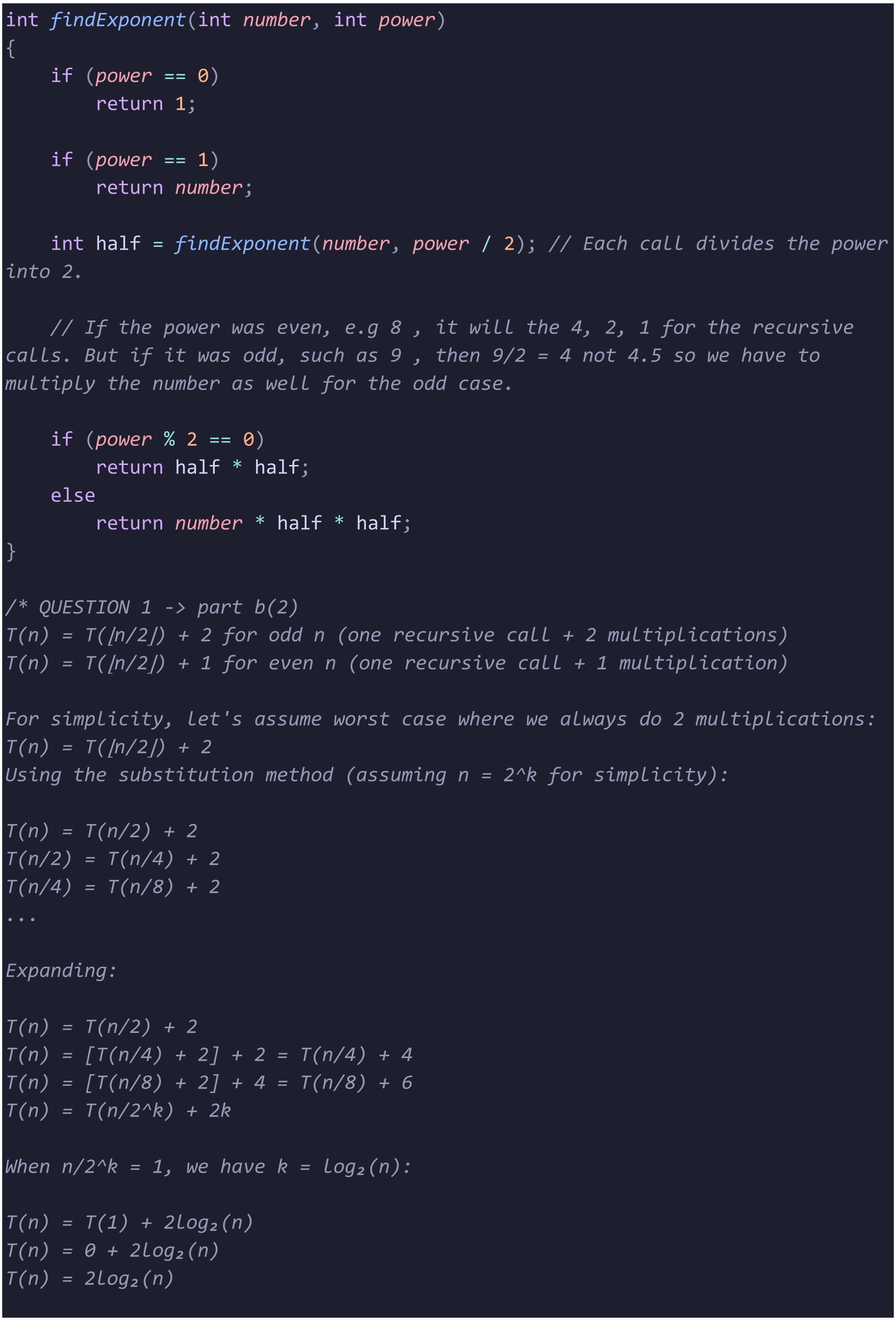
## Output



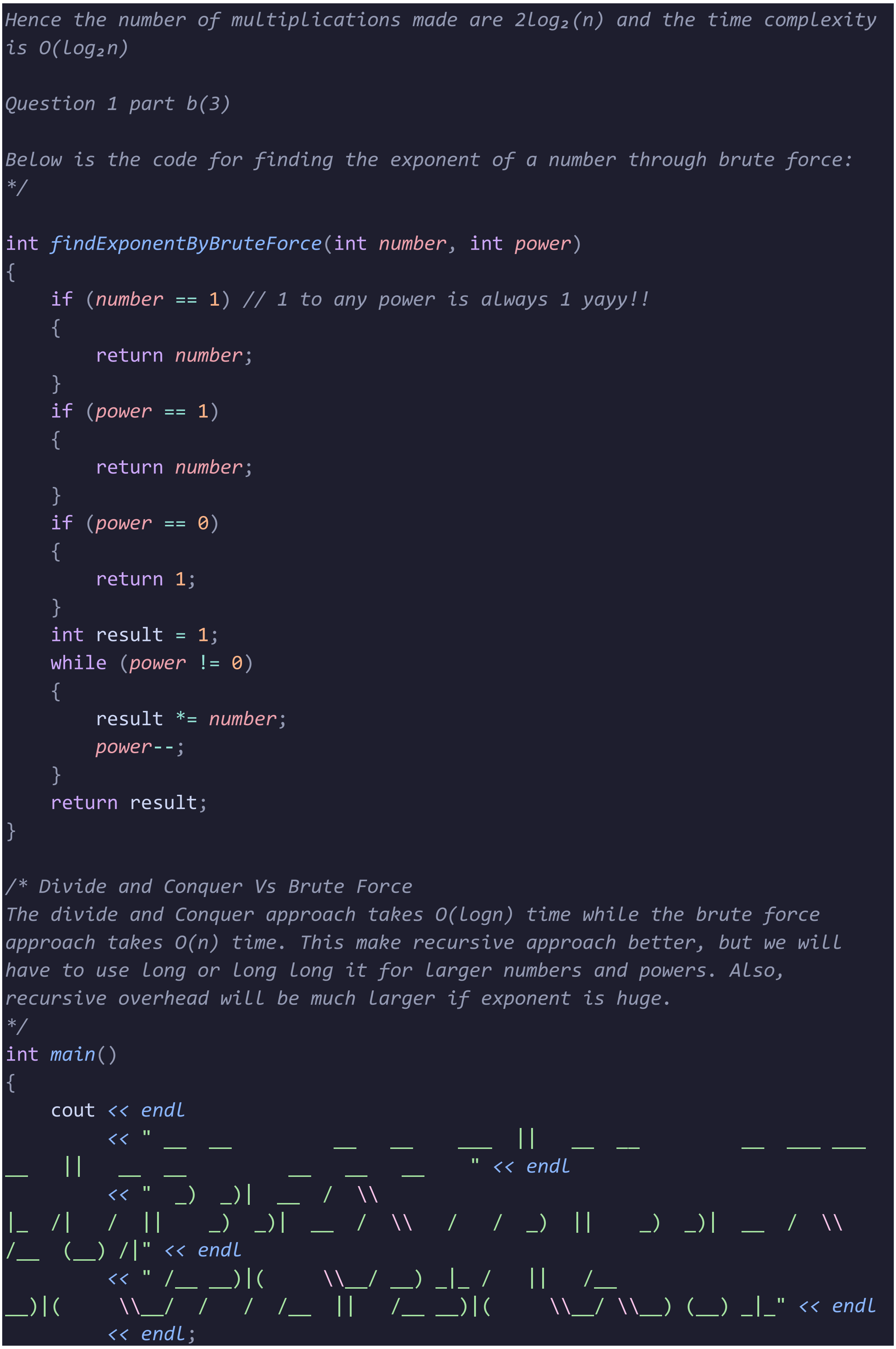
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-b.cpp -o partB
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partB

 __  __          __   __   ___  ||   __  __          __  ___ ___ __   ||   __  __          __   __   __
  _) _)| __  / \ |_  /|   /  ||    _) _)| __  / \   /   /  _)  ||    _) _)| __  / \ /__  (__) /|
 /__ __)|(     \__/ __) _|_ /   ||   /__ __)|(     \__/  /   /  /__  ||   /__ __)|(     \__/ \__) (__) _|_

Enter number and power: 2 8
By Divide and Conquer: = 256
By Brute Force: = 256
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

## Part b

```
#include <iostream>
#include <vector>

using namespace std;

/* QUESTION 1 -> part b(1)

If n is even: a^n = (a^(n/2))^2
If n is odd: a^n = a × (a^(n-1))^2 = a × (a^(n/2))^2

Example: 2^8

2^8 = (2^4)^2
2^4 = (2^2)^2
2^2 = (2^1)^2
2^1 = 2

*/
```

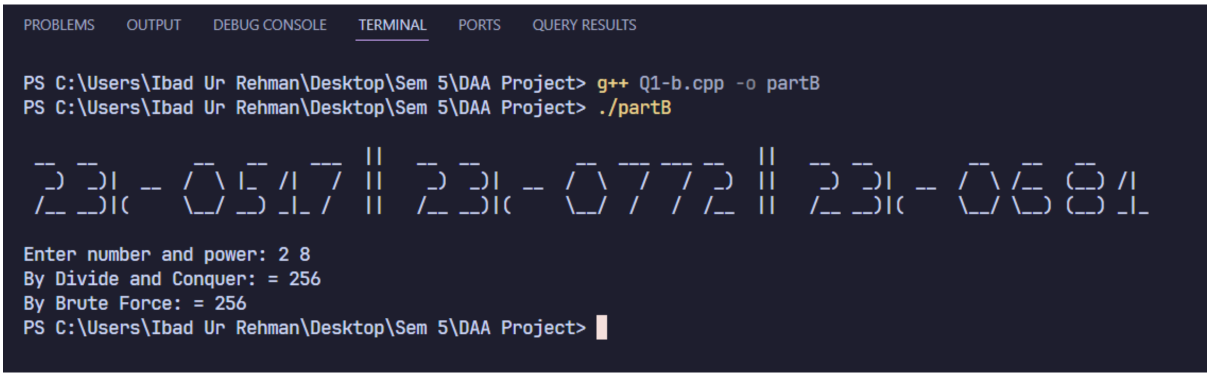


```
int findExponent(int number, int power)
{
    if (power == 0)
        return 1;

    if (power == 1)
        return number;

    int half = findExponent(number, power / 2); // Each call divides the power
into 2.

    // If the power was even, e.g 8 , it will the 4, 2, 1 for the recursive
calls. But if it was odd, such as 9 , then 9/2 = 4 not 4.5 so we have to
multiply the number as well for the odd case.

    if (power % 2 == 0)
        return half * half;
    else
        return number * half * half;
}

/* QUESTION 1 -> part b(2)
T(n) = T(⌊n/2⌋) + 2 for odd n (one recursive call + 2 multiplications)
T(n) = T(⌊n/2⌋) + 1 for even n (one recursive call + 1 multiplication)

For simplicity, let's assume worst case where we always do 2 multiplications:
T(n) = T(⌊n/2⌋) + 2
Using the substitution method (assuming n = 2^k for simplicity):

T(n) = T(n/2) + 2
T(n/2) = T(n/4) + 2
T(n/4) = T(n/8) + 2
...

Expanding:

T(n) = T(n/2) + 2
T(n) = [T(n/4) + 2] + 2 = T(n/4) + 4
T(n) = [T(n/8) + 2] + 4 = T(n/8) + 6
T(n) = T(n/2^k) + 2k

When n/2^k = 1, we have k = log₂(n):

T(n) = T(1) + 2log₂(n)
T(n) = 0 + 2log₂(n)
T(n) = 2log₂(n)
```



```
Hence the number of multiplications made are 2log₂(n) and the time complexity
is O(log₂n)

Question 1 part b(3)

Below is the code for finding the exponent of a number through brute force:
*/

int findExponentByBruteForce(int number, int power)
{
    if (number == 1) // 1 to any power is always 1 yayy!!
    {
        return number;
    }
    if (power == 1)
    {
        return number;
    }
    if (power == 0)
    {
        return 1;
    }
    int result = 1;
    while (power != 0)
    {
        result *= number;
        power--;
    }
    return result;
}

/* Divide and Conquer Vs Brute Force
The divide and Conquer approach takes O(logn) time while the brute force
approach takes O(n) time. This make recursive approach better, but we will
have to use long or long long it for larger numbers and powers. Also,
recursive overhead will be much larger if exponent is huge.
*/
int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    int number, power;
    cout
        << "Enter number and power: ";
    cin >> number >> power;
    cout << "By Divide and Conquer: = " << findExponent(number, power) <<
endl;
    cout << "By Brute Force: = " << findExponentByBruteForce(number, power);
}
```

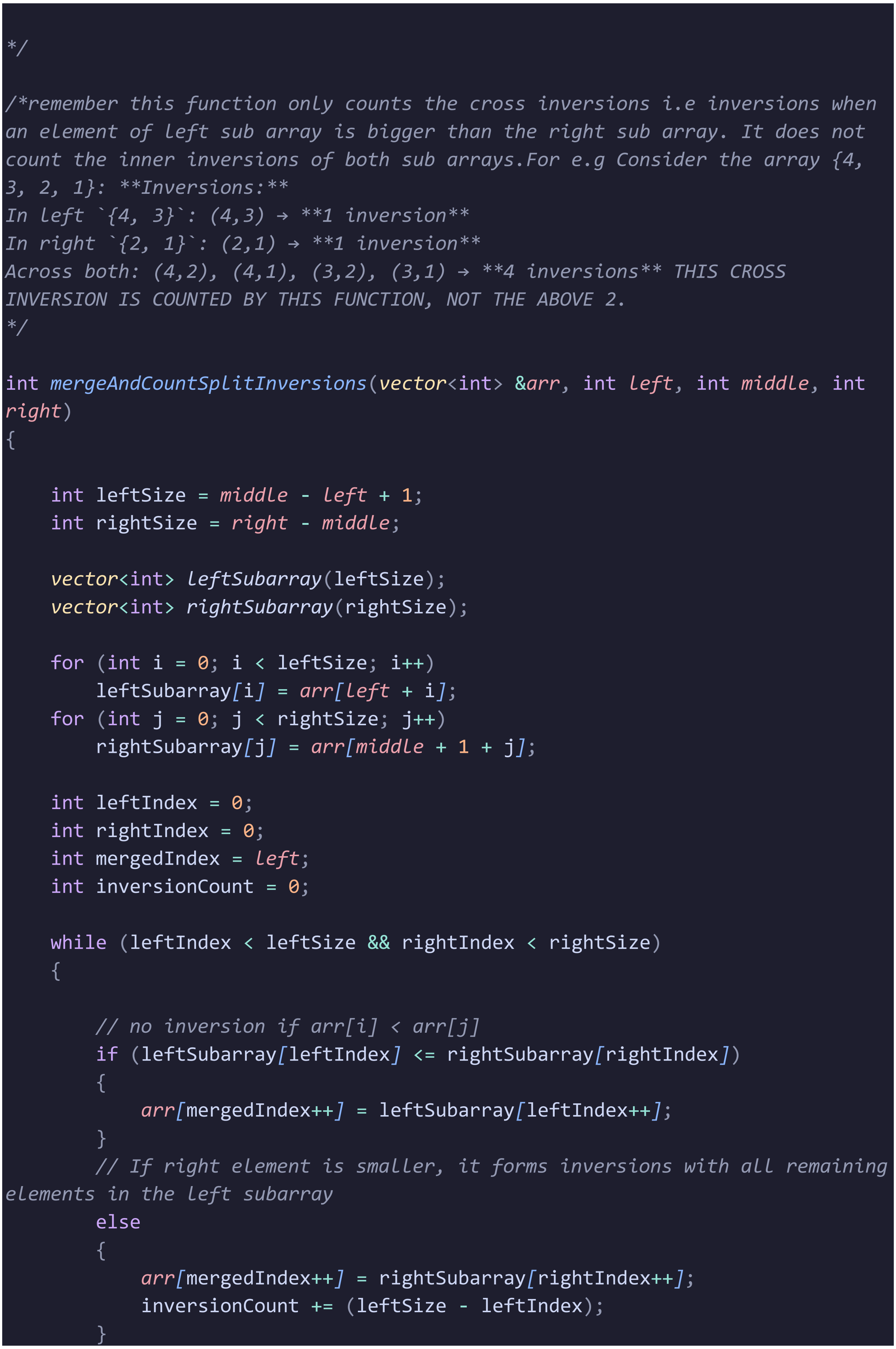
## Output



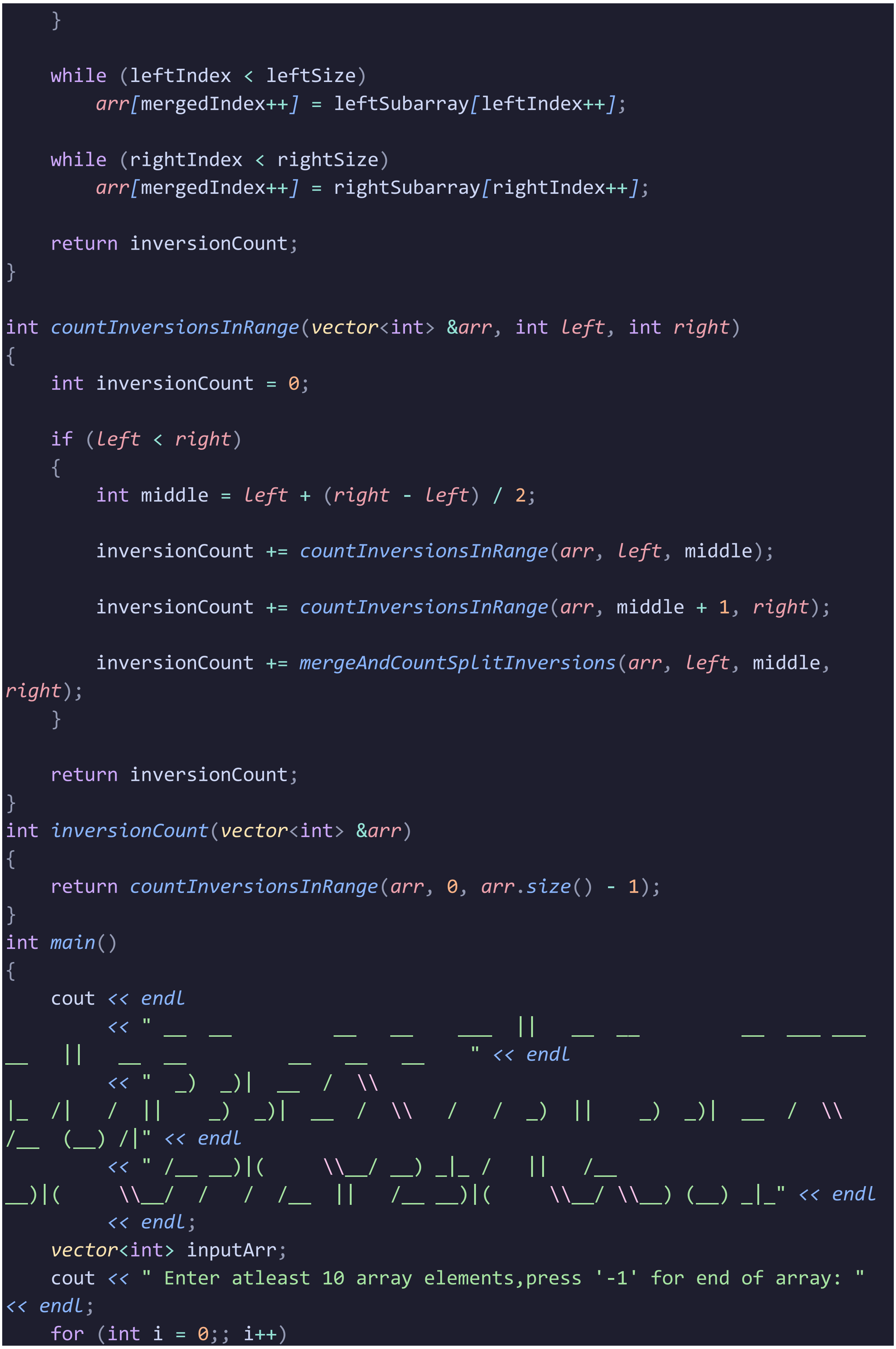
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-b.cpp -o partB
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partB

 __  __      __   __   ___  ||  __  __      __  ___ ___ __  ||  __  __      __   __   __
  _)  _)|  __ /  \ |_  /|   /  ||   _)  _)|  __ /  \   /   /  _)  ||   _)  _)|  __ /  \ /__  (__) /|
 /__ __)|(    \__/ __) _|_ /   ||  /__ __)|(    \__/  /   /  /__  ||  /__ __)|(    \__/ \__) (__) _|_

Enter number and power: 2 8
By Divide and Conquer: = 256
By Brute Force: = 256
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

## Part c

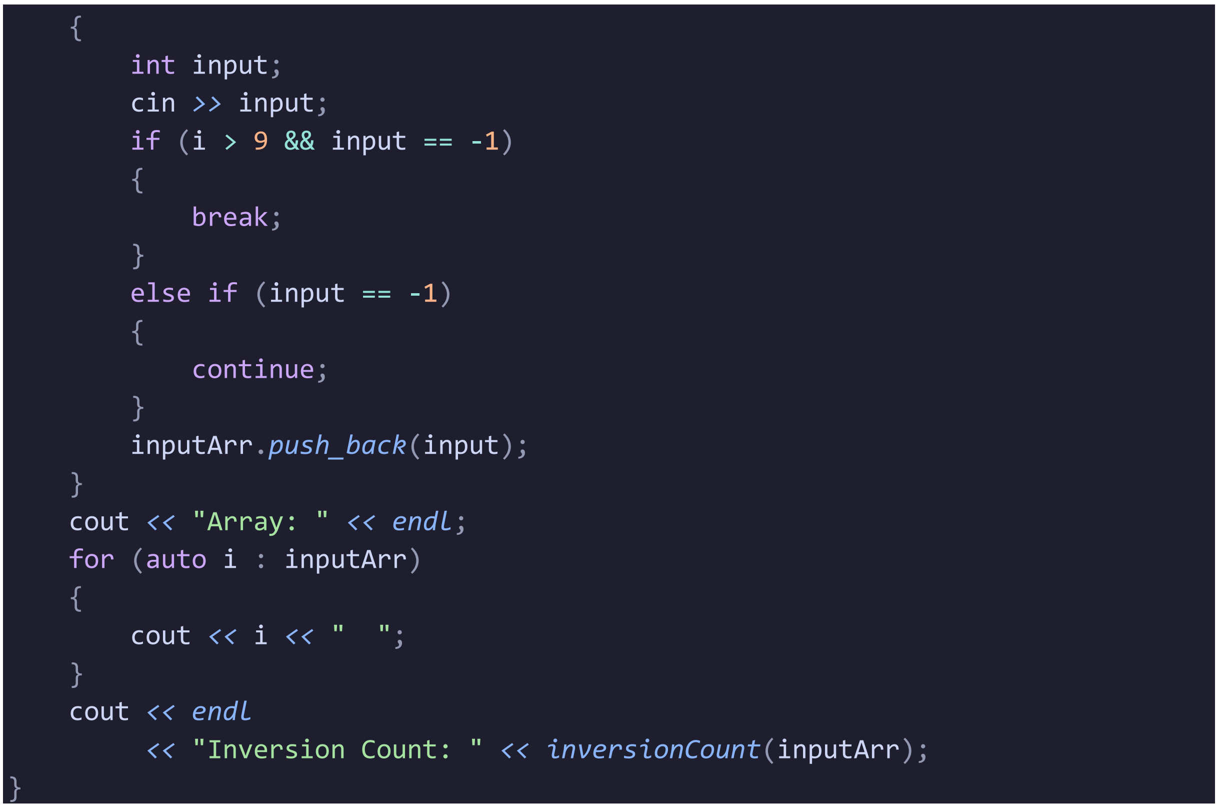
```
#include <iostream>
#include <vector>

using namespace std;

/*
Inversions??
->Those elements in an array where arr[i] > arr[j] but j > i.
->Basically this is showing how much an array is sorted.

How to count inversions??
->During each merging step of the merge sort algorithm, we count cross
inversions by comparing elements from the left half of the array with those
from the right half. If we find an element arr[i] in the left half that is
greater than an element arr[j] in the right half, we can conclude that all
elements after i in the left half will also be greater than arr[j]. This
allows us to count multiple inversions at once. Let's suppose if there are k
elements remaining in the left half after i, then there are k cross inversions
for that particular arr[j]. The rest of the merging process continues as
usual, where we combine the two halves into a sorted array.
```



```
*/

/*remember this function only counts the cross inversions i.e inversions when
an element of left sub array is bigger than the right sub array. It does not
count the inner inversions of both sub arrays.For e.g Consider the array {4,
3, 2, 1}: **Inversions:**
In left `{4, 3}`: (4,3) → **1 inversion**
In right `{2, 1}`: (2,1) → **1 inversion**
Across both: (4,2), (4,1), (3,2), (3,1) → **4 inversions** THIS CROSS
INVERSION IS COUNTED BY THIS FUNCTION, NOT THE ABOVE 2.
*/

int mergeAndCountSplitInversions(vector<int> &arr, int left, int middle, int
right)
{

    int leftSize = middle - left + 1;
    int rightSize = right - middle;

    vector<int> leftSubarray(leftSize);
    vector<int> rightSubarray(rightSize);

    for (int i = 0; i < leftSize; i++)
        leftSubarray[i] = arr[left + i];
    for (int j = 0; j < rightSize; j++)
        rightSubarray[j] = arr[middle + 1 + j];

    int leftIndex = 0;
    int rightIndex = 0;
    int mergedIndex = left;
    int inversionCount = 0;

    while (leftIndex < leftSize && rightIndex < rightSize)
    {

        // no inversion if arr[i] < arr[j]
        if (leftSubarray[leftIndex] <= rightSubarray[rightIndex])
        {
            arr[mergedIndex++] = leftSubarray[leftIndex++];
        }
        // If right element is smaller, it forms inversions with all remaining
elements in the left subarray
        else
        {
            arr[mergedIndex++] = rightSubarray[rightIndex++];
            inversionCount += (leftSize - leftIndex);
        }
```

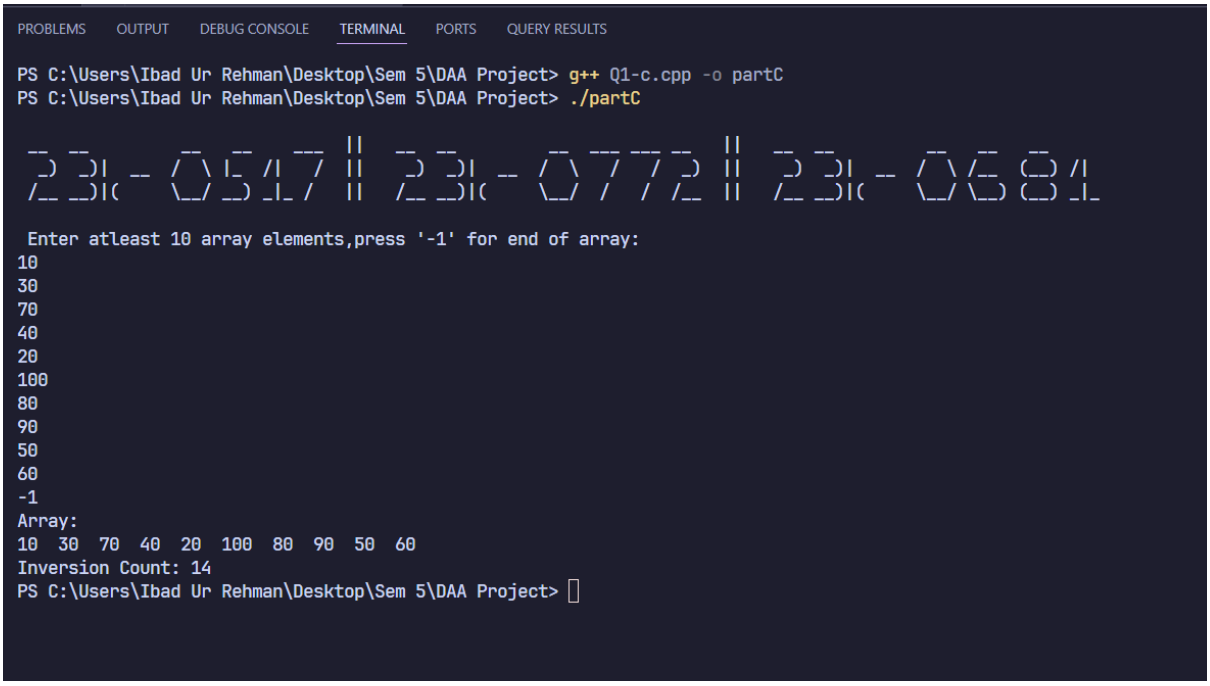


```
    }

    while (leftIndex < leftSize)
        arr[mergedIndex++] = leftSubarray[leftIndex++];

    while (rightIndex < rightSize)
        arr[mergedIndex++] = rightSubarray[rightIndex++];

    return inversionCount;
}

int countInversionsInRange(vector<int> &arr, int left, int right)
{
    int inversionCount = 0;

    if (left < right)
    {
        int middle = left + (right - left) / 2;

        inversionCount += countInversionsInRange(arr, left, middle);

        inversionCount += countInversionsInRange(arr, middle + 1, right);

        inversionCount += mergeAndCountSplitInversions(arr, left, middle,
right);
    }

    return inversionCount;
}
int inversionCount(vector<int> &arr)
{
    return countInversionsInRange(arr, 0, arr.size() - 1);
}
int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    vector<int> inputArr;
    cout << " Enter atleast 10 array elements,press '-1' for end of array: "
<< endl;
    for (int i = 0;; i++)
```



```
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }
    cout << "Array: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }
    cout << endl
         << "Inversion Count: " << inversionCount(inputArr);
}
```

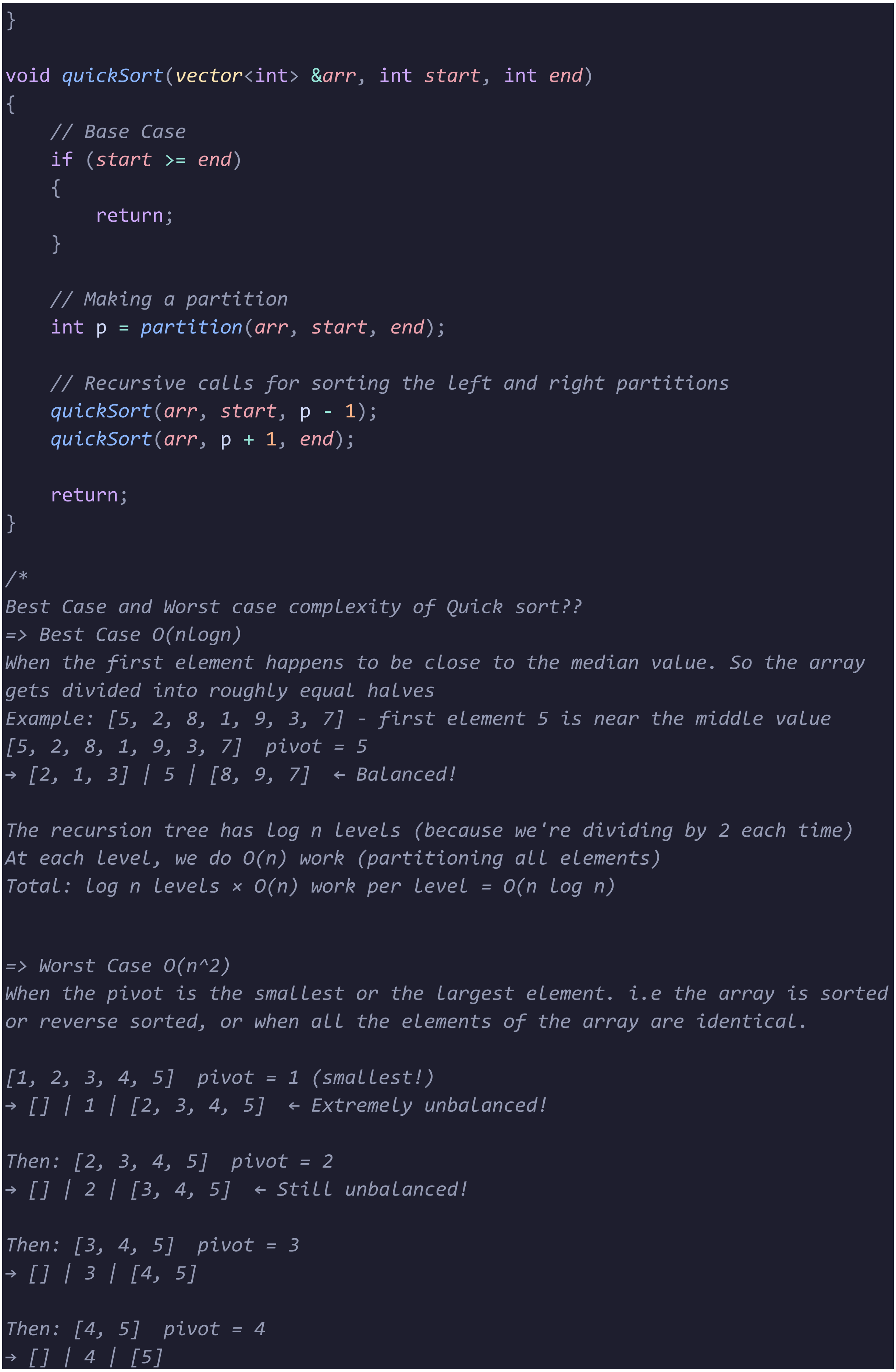
# Output



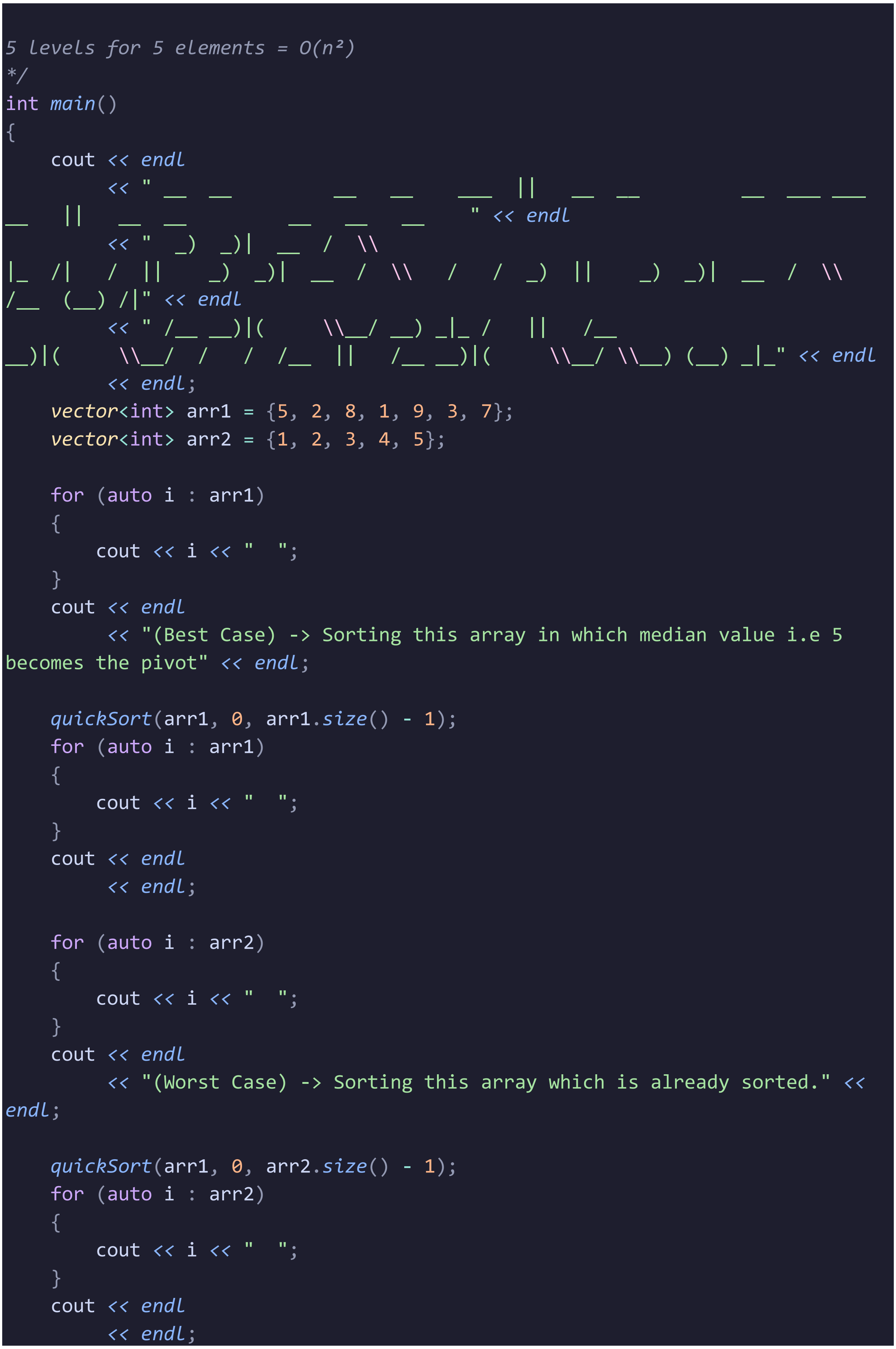
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-c.cpp -o partC
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partC

 __  __         __   __   ___  ||   __  __         __  ___ ___ __   ||   __  __         __   __   __
  _)  _)| __  /  \ |_  /|   /  ||   _)  _)| __  /  \   /   /  _)  ||   _)  _)| __  /  \ /__  (__) /|
 /__ __)|(    \__/ __) _|_ /   ||  /__ __)|(    \__/  /   /  /__  ||  /__ __)|(    \__/ \__) (__) _|_

 Enter atleast 10 array elements,press '-1' for end of array:
10
30
70
40
20
100
80
90
50
60
-1
Array:
10  30  70  40  20  100  80  90  50  60
Inversion Count: 14
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

# Part d

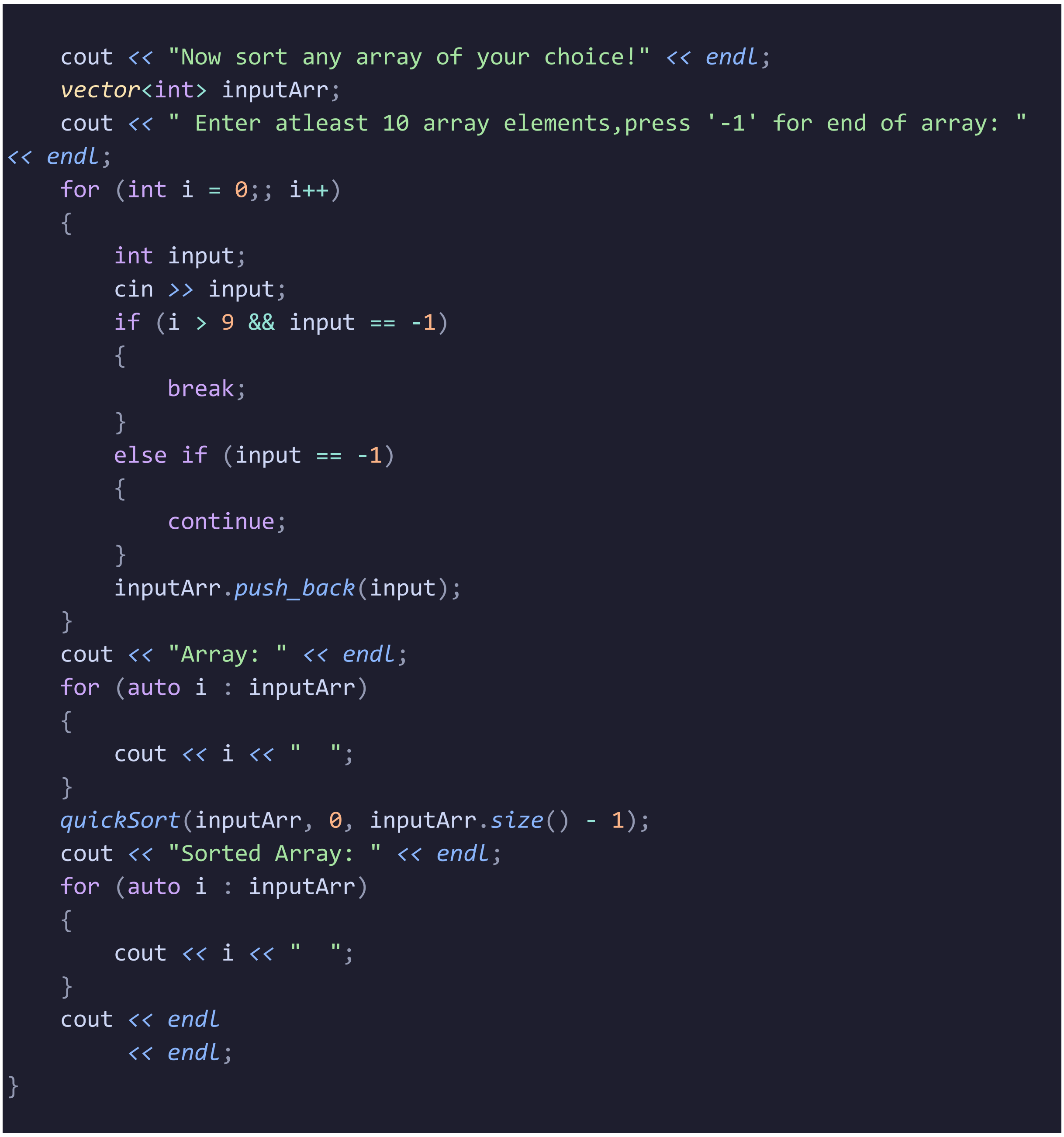
```
// We can have multiple approaches for finding the pivot element. I have used
the approach in which the first element of the partition is the pivot element
always.

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

int partition(vector<int> &arr, int start, int end)
{
    int pivot = arr[start]; // Pivot is our first element of the array

    // First we count how many numbers are less than the pivot in the array
    int count = 0;
    for (int i = start; i <= end; i++)
    {
        if (arr[i] < pivot)
        {
            count++;
        }
    }

    int pivotIndex = start + count;

    swap(arr[start], arr[pivotIndex]);
    int i = start;
    int j = end;
    while (i < pivotIndex && j > pivotIndex)
    {
        while (arr[i] < pivot) // No need to swap case1
        {
            i++;
        }

        while (arr[j] > pivot) // No need to swap case2
        {
            j--;
        }

        if (i < pivotIndex && j > pivotIndex)
        {
            swap(arr[i], arr[j]);
        }
    }

    return pivotIndex;
```



```
}

void quickSort(vector<int> &arr, int start, int end)
{
    // Base Case
    if (start >= end)
    {
        return;
    }

    // Making a partition
    int p = partition(arr, start, end);

    // Recursive calls for sorting the left and right partitions
    quickSort(arr, start, p - 1);
    quickSort(arr, p + 1, end);

    return;
}

/*
Best Case and Worst case complexity of Quick sort??
=> Best Case O(nlogn)
When the first element happens to be close to the median value. So the array
gets divided into roughly equal halves
Example: [5, 2, 8, 1, 9, 3, 7] - first element 5 is near the middle value
[5, 2, 8, 1, 9, 3, 7]  pivot = 5
→ [2, 1, 3] | 5 | [8, 9, 7]  ← Balanced!

The recursion tree has log n levels (because we're dividing by 2 each time)
At each level, we do O(n) work (partitioning all elements)
Total: log n levels × O(n) work per level = O(n log n)


=> Worst Case O(n^2)
When the pivot is the smallest or the largest element. i.e the array is sorted
or reverse sorted, or when all the elements of the array are identical.

[1, 2, 3, 4, 5]  pivot = 1 (smallest!)
→ [] | 1 | [2, 3, 4, 5]  ← Extremely unbalanced!

Then: [2, 3, 4, 5]  pivot = 2
→ [] | 2 | [3, 4, 5]  ← Still unbalanced!

Then: [3, 4, 5]  pivot = 3
→ [] | 3 | [4, 5]

Then: [4, 5]  pivot = 4
→ [] | 4 | [5]
```

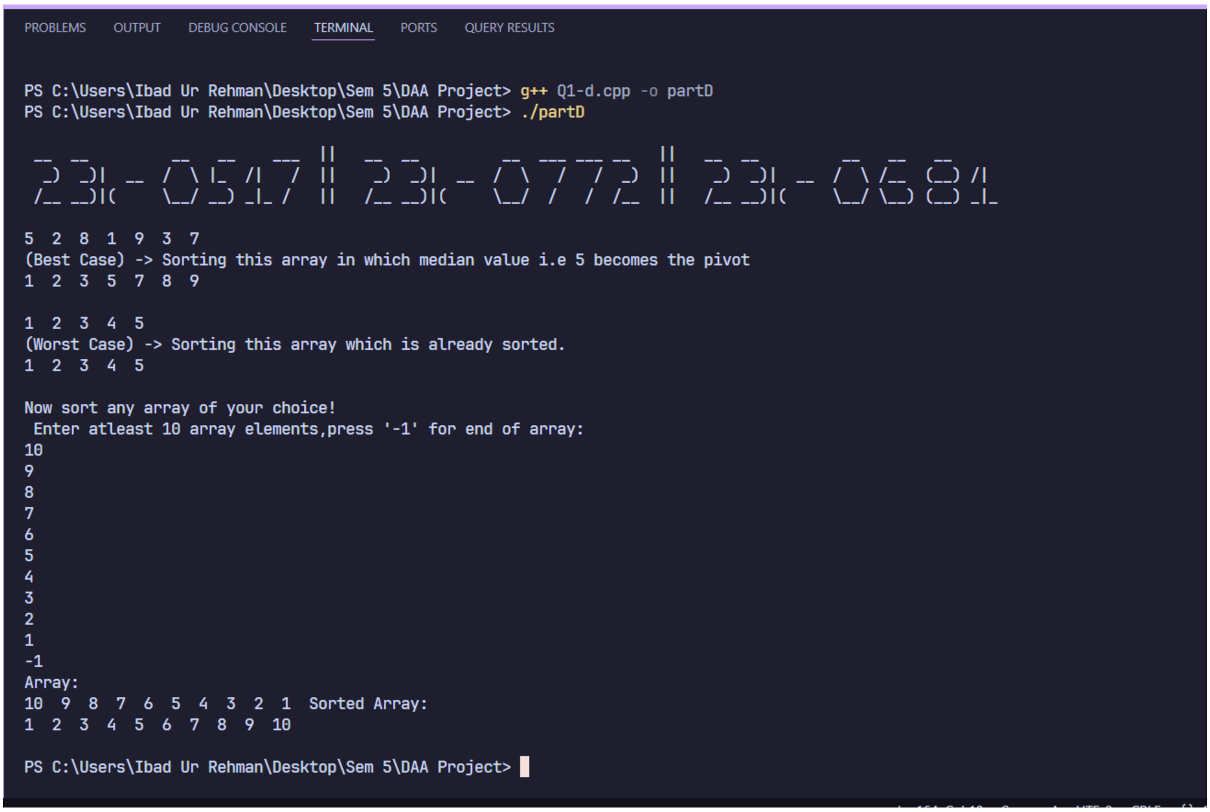


```
5 levels for 5 elements = O(n²)
*/
int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    vector<int> arr1 = {5, 2, 8, 1, 9, 3, 7};
    vector<int> arr2 = {1, 2, 3, 4, 5};

    for (auto i : arr1)
    {
        cout << i << "  ";
    }
    cout << endl
         << "(Best Case) -> Sorting this array in which median value i.e 5
becomes the pivot" << endl;

    quickSort(arr1, 0, arr1.size() - 1);
    for (auto i : arr1)
    {
        cout << i << "  ";
    }
    cout << endl
         << endl;

    for (auto i : arr2)
    {
        cout << i << "  ";
    }
    cout << endl
         << "(Worst Case) -> Sorting this array which is already sorted." <<
endl;

    quickSort(arr1, 0, arr2.size() - 1);
    for (auto i : arr2)
    {
        cout << i << "  ";
    }
    cout << endl
         << endl;
```



```
    cout << "Now sort any array of your choice!" << endl;
    vector<int> inputArr;
    cout << " Enter atleast 10 array elements,press '-1' for end of array: "
<< endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }
    cout << "Array: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }
    quickSort(inputArr, 0, inputArr.size() - 1);
    cout << "Sorted Array: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }
    cout << endl
         << endl;
}
```

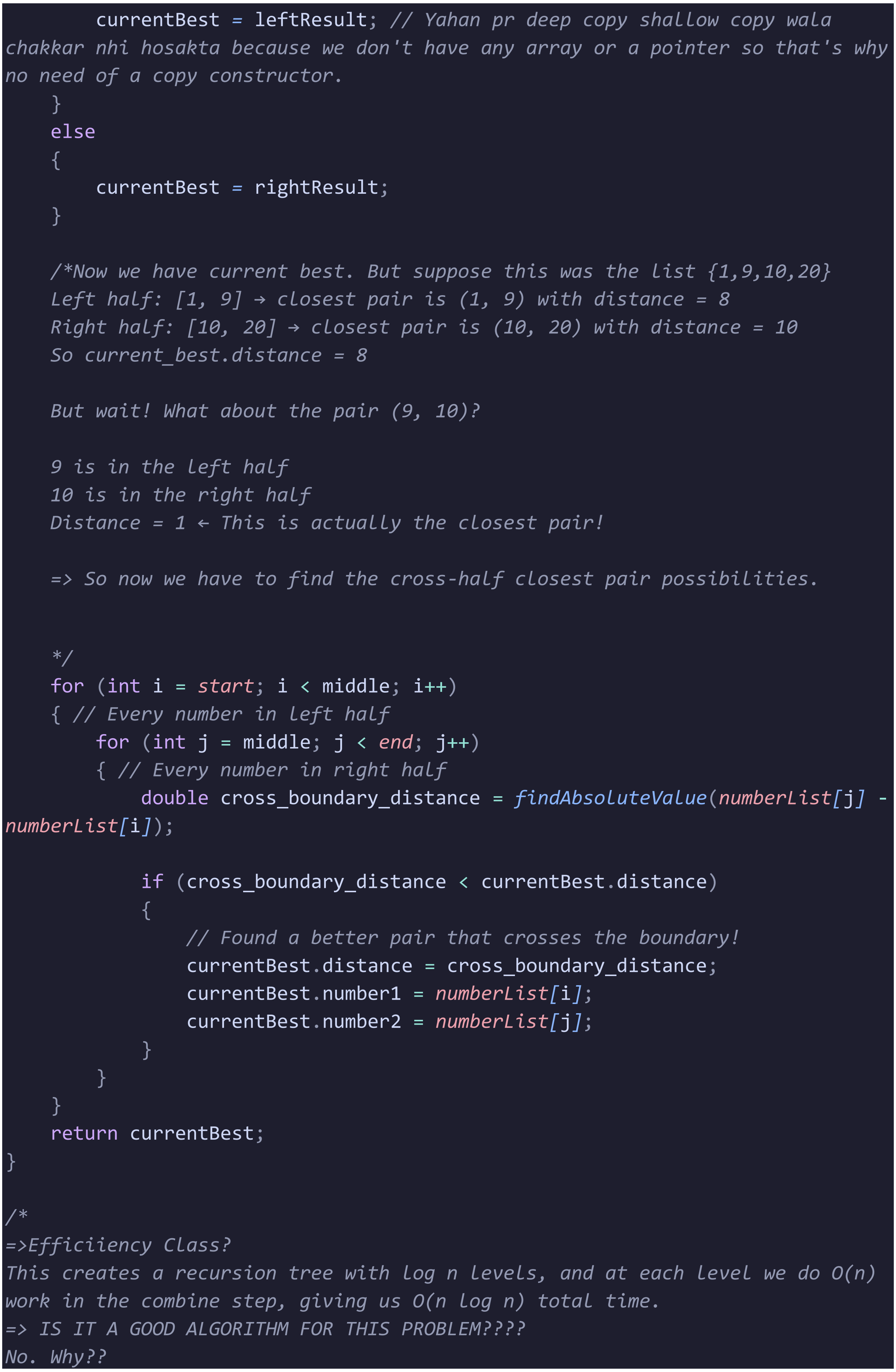
# Output



```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-d.cpp -o partD
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partD

 __ __          __   __    ___  ||   __ __        __  ___ ___ __   ||   __ __       __   __    __
  _) _)| __  /  \ |_  /|   /   ||    _)  _)| __  /  \   /   /  _)  ||    _)  _)| __  /  \ /__  (__) /|
 /__ __)|(    \__/ __) _|_ /    ||   /__ __)|(    \__/  /   /  /__  ||   /__ __)|(    \__/ \__) (__) _|_

5  2  8  1  9  3  7
(Best Case) -> Sorting this array in which median value i.e 5 becomes the pivot
1  2  3  5  7  8  9

1  2  3  4  5
(Worst Case) -> Sorting this array which is already sorted.
1  2  3  4  5

Now sort any array of your choice!
 Enter atleast 10 array elements,press '-1' for end of array:
10
9
8
7
6
5
4
3
2
1
-1
Array:
10  9  8  7  6  5  4  3  2  1  Sorted Array:
1  2  3  4  5  6  7  8  9  10

PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

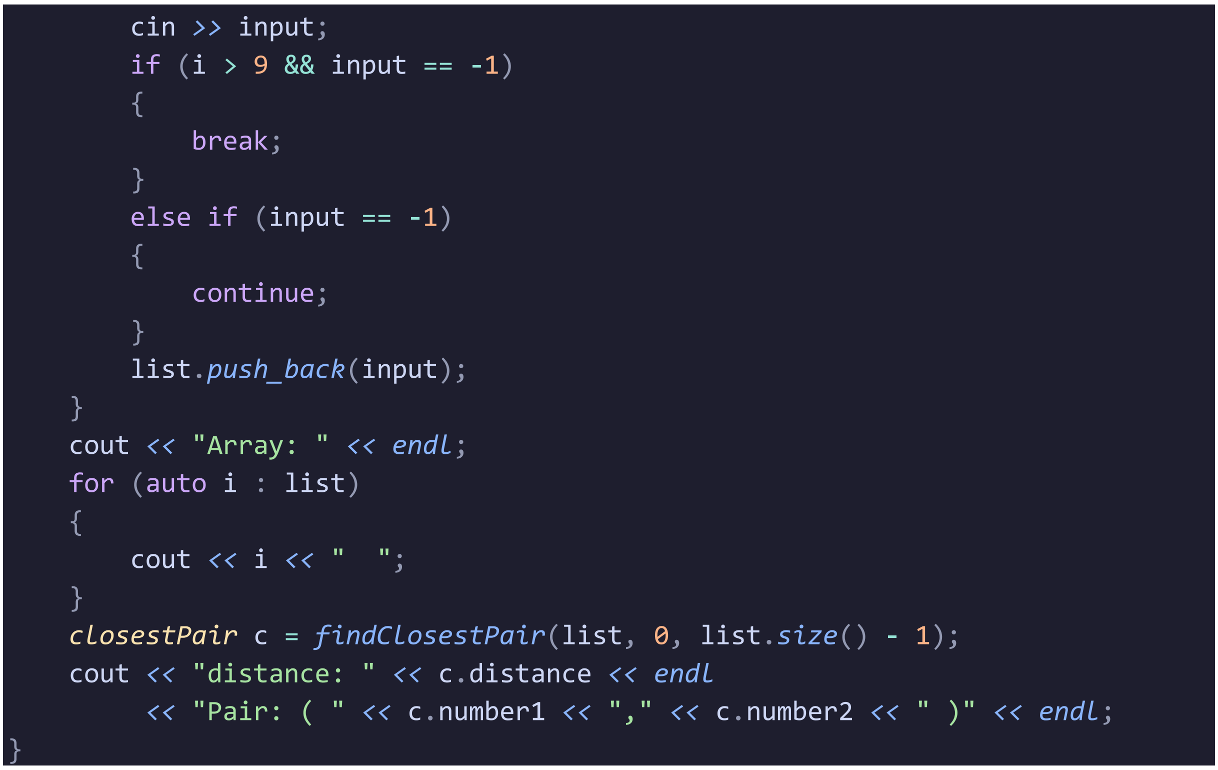
# Part e

```
/*
One dimensional???
-> This means we are working on a single number line , therefore we do not
have points here in 2D space, in which we have x and y coordinates both.

*/

#include <iostream>
#include <vector>

using namespace std;

// Creating a class here as we have to return the cloesest distance + the pair
of points. So we will return the whole object instead.
class closestPair
{
public:
    double number1;
    double number2;
    double distance;
    closestPair() {};
    closestPair(double n1, double n2, double d) : number1(n1), number2(n2),
distance(d) {}
};

// This fnction makes every negative distance positive , as distance is always
positive.
double findAbsoluteValue(double x)
{
    if (x < 0)
    {
        return -x; // E.g if x is -5, it does -(-5) so it becomes 5.
    }
    return x;
}

// Asal function ye hai

closestPair findClosestPair(vector<double> &numberList, int start, int end)
{
    int size = end - start;

    if (size <= 1)
    {
        return closestPair(0, 0, 1e308); // Base case 1
    }

    if (size == 2)
    {
        double dist = findAbsoluteValue(numberList[start + 1] -
numberList[start]);
        return closestPair(numberList[start], numberList[start + 1], dist); //
Base Case 2
    }

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

    // Recursive Calls for left and right halves
    closestPair leftResult = findClosestPair(numberList, start, middle);
    closestPair rightResult = findClosestPair(numberList, middle, end);

    // Find now which half has the smaller distance.

    closestPair currentBest;
    if (leftResult.distance <= rightResult.distance)
    {
```



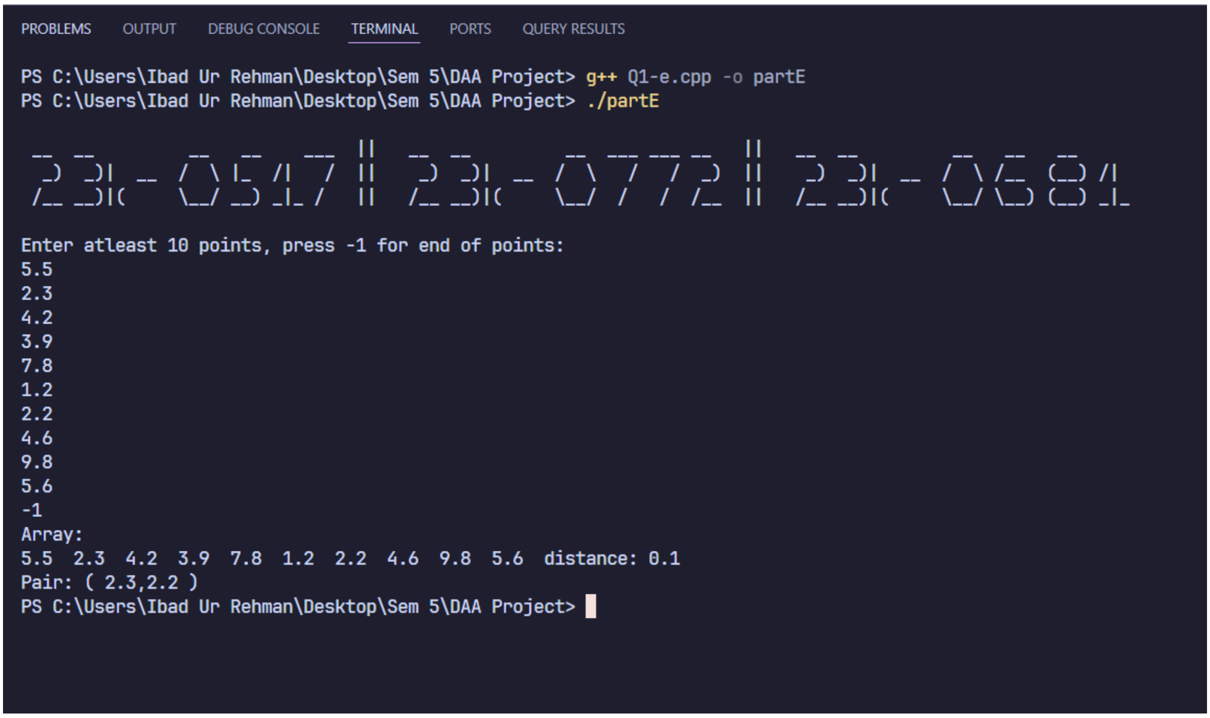
```
        currentBest = leftResult; // Yahan pr deep copy shallow copy wala
chakkar nhi hosakta because we don't have any array or a pointer so that's why
no need of a copy constructor.
    }
    else
    {
        currentBest = rightResult;
    }

    /*Now we have current best. But suppose this was the list {1,9,10,20}
    Left half: [1, 9] → closest pair is (1, 9) with distance = 8
    Right half: [10, 20] → closest pair is (10, 20) with distance = 10
    So current_best.distance = 8

    But wait! What about the pair (9, 10)?

    9 is in the left half
    10 is in the right half
    Distance = 1 ← This is actually the closest pair!

    => So now we have to find the cross-half closest pair possibilities.


    */
    for (int i = start; i < middle; i++)
    { // Every number in left half
        for (int j = middle; j < end; j++)
        { // Every number in right half
            double cross_boundary_distance = findAbsoluteValue(numberList[j] -
numberList[i]);

            if (cross_boundary_distance < currentBest.distance)
            {
                // Found a better pair that crosses the boundary!
                currentBest.distance = cross_boundary_distance;
                currentBest.number1 = numberList[i];
                currentBest.number2 = numberList[j];
            }
        }
    }
    return currentBest;
}

/*
=>Efficiiency Class?
This creates a recursion tree with log n levels, and at each level we do O(n)
work in the combine step, giving us O(n log n) total time.
=> IS IT A GOOD ALGORITHM FOR THIS PROBLEM????
No. Why??
```

```
  for (int i = start; i < middle; i++)
    { // Every number in left half
        for (int j = middle; j < end; j++)
        { // Every number in right half
            double cross_boundary_distance = findAbsoluteValue(numberList[j] -
numberList[i]);

            if (cross_boundary_distance < currentBest.distance)
            {
                // Found a better pair that crosses the boundary!
                currentBest.distance = cross_boundary_distance;
                currentBest.number1 = numberList[i];
                currentBest.number2 = numberList[j];
            }
        }
    }

This nested loop checks every possible pair where:

One number comes from the left half
One number comes from the right half

If any of these cross-boundary pairs are closer than what we found before, we
update our answer.
Why Is This O(n) Work?
In the worst case, if the left half has n/2 numbers and right half has n/2
numbers, we're doing (n/2) × (n/2) = O(n²) comparisons at each level! This is
actually inefficient, which is why this approach isn't optimal for the 1D
problem.

*/

int main()
{
    cout << endl
         << " __  __         __   __   ___  ||   __  __        __  ___ ___
__   ||   __ __         __   __  __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||   _)  _)|  __  /  \\   /  /  _)  ||   _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||  /__
__)|(     \\__/  /   /  /__  ||  /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    vector<double> list;
    cout << "Enter atleast 10 points, press -1 for end of points: " << endl;
    for (int i = 0;; i++)
    {
        double input;
```



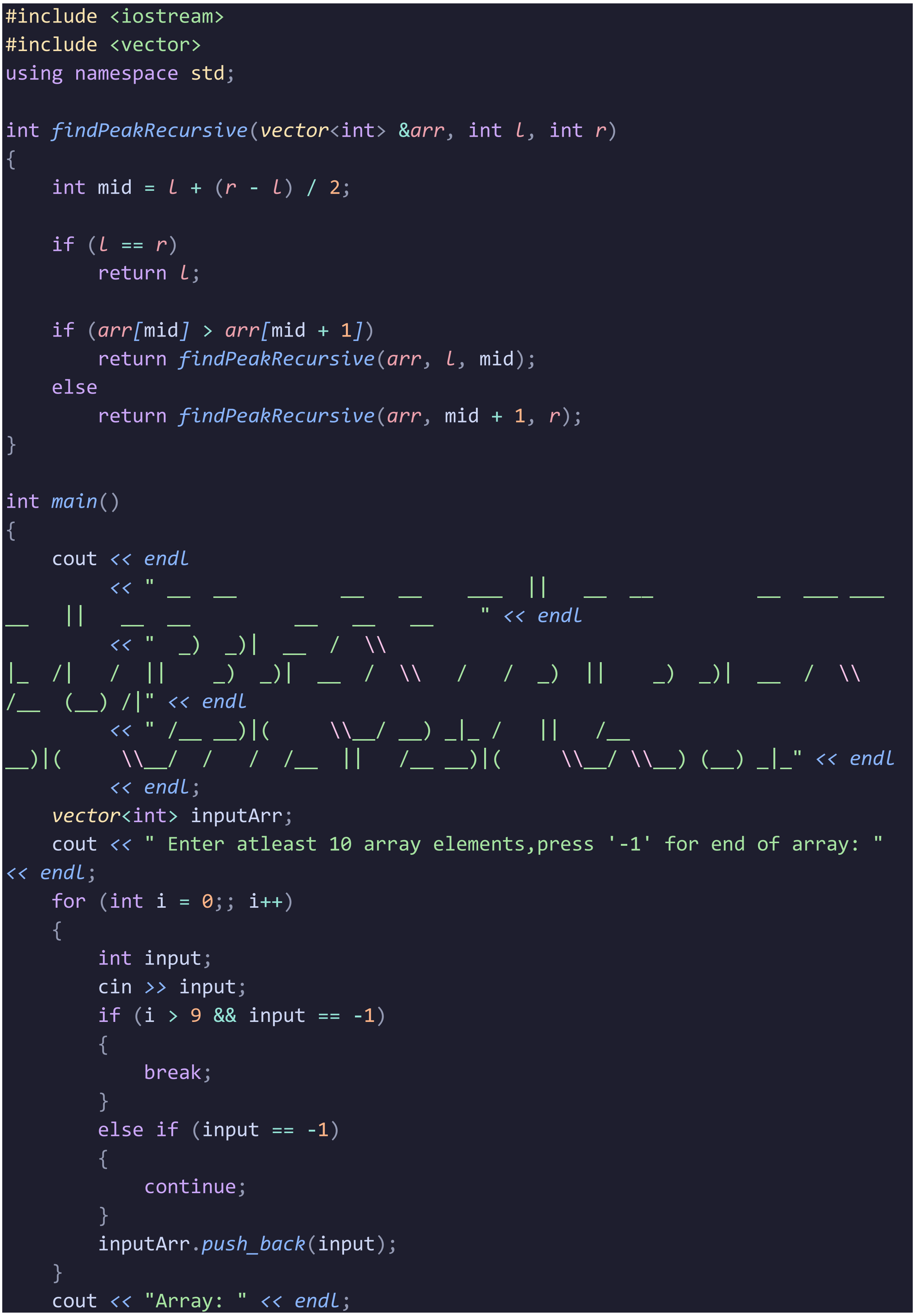
```
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        list.push_back(input);
    }
    cout << "Array: " << endl;
    for (auto i : list)
    {
        cout << i << "  ";
    }
    closestPair c = findClosestPair(list, 0, list.size() - 1);
    cout << "distance: " << c.distance << endl
         << "Pair: ( " << c.number1 << "," << c.number2 << " )" << endl;
}
```

## Output



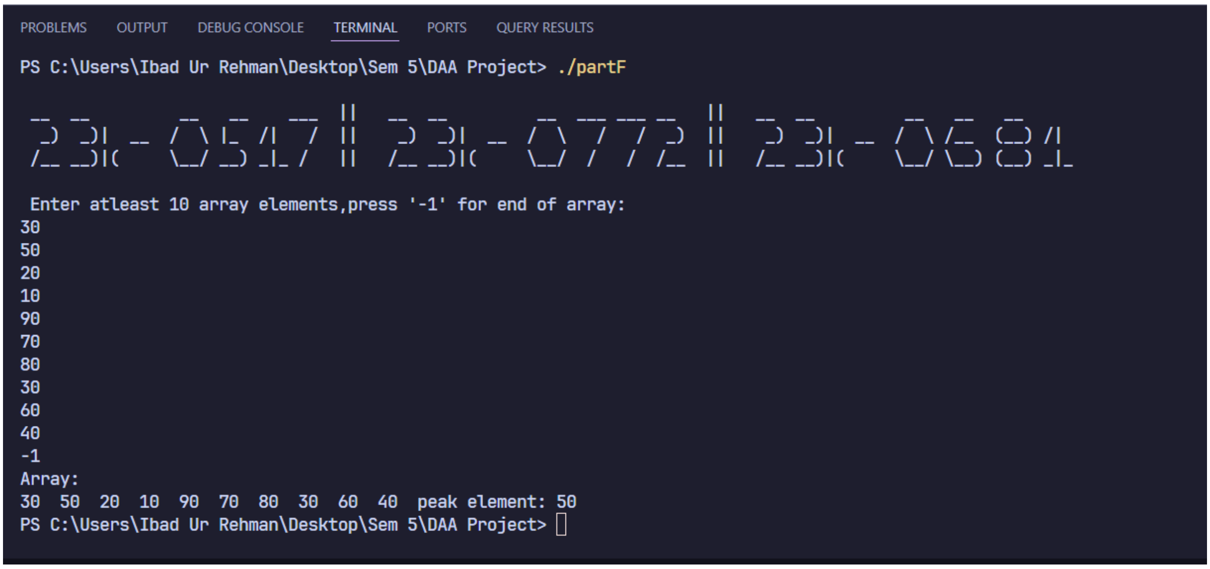
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-e.cpp -o partE
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partE

                                    ||                                  ||
 __  __        __   __   ___  ||   __  __        __  ___ ___ __   ||   __  __        __   __   __
  _)  _)|  __  /  \ |_  /|   /  ||   _)  _)|  __  /  \   /   /  _)  ||   _)  _)|  __  /  \ /__  (__) /|
 /__ __)|(     \__/ __) _|_ /   ||  /__ __)|(     \__/  /   /  /__  ||  /__ __)|(     \__/ \__) (__) _|_

Enter atleast 10 points, press -1 for end of points:
5.5
2.3
4.2
3.9
7.8
1.2
2.2
4.6
9.8
5.6
-1
Array:
5.5  2.3  4.2  3.9  7.8  1.2  2.2  4.6  9.8  5.6  distance: 0.1
Pair: ( 2.3,2.2 )
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> █
```

## Part f



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

int findPeakRecursive(vector<int> &arr, int l, int r)
{
    int mid = l + (r - l) / 2;

    if (l == r)
        return l;

    if (arr[mid] > arr[mid + 1])
        return findPeakRecursive(arr, l, mid);
    else
        return findPeakRecursive(arr, mid + 1, r);
}

int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    vector<int> inputArr;
    cout << " Enter atleast 10 array elements,press '-1' for end of array: "
<< endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }
    cout << "Array: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }
    int n = inputArr.size();
    int p = findPeakRecursive(inputArr, 0, n - 1);
    cout << "peak element: " << inputArr[p] << endl;
}
```

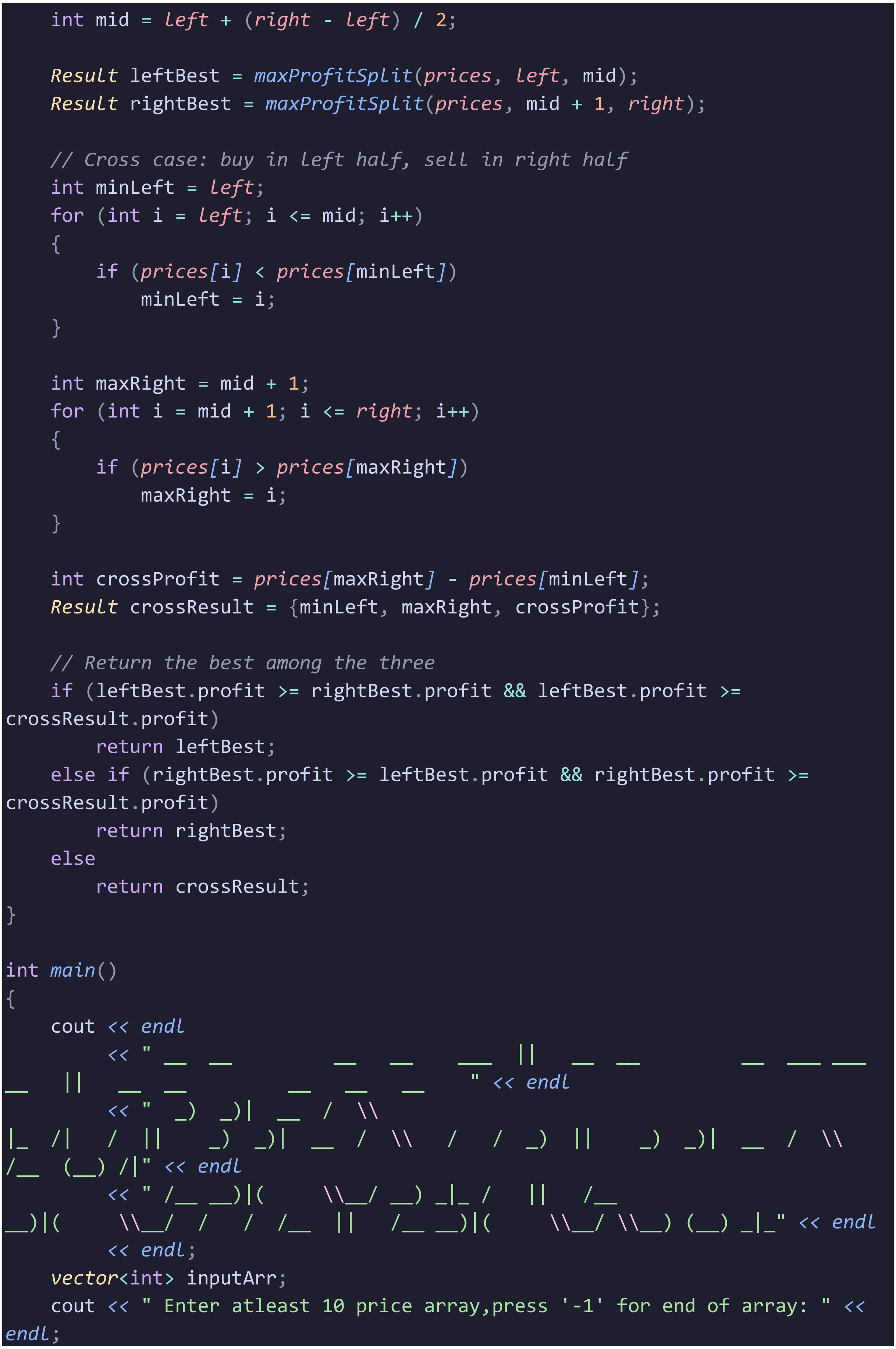
## Output



```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partF

 __  __       __   __   ___  ||  __  __       __  ___ ___ __   ||  __  __       __   __   __
  _)  _)|  __  /  \ |_  /|   /  ||   _)  _)|  __  /  \   /   /  _)  ||   _)  _)|  __  /  \ /__  (__) /|
 /__ __)|(     \__/ __) _|_ /   ||  /__ __)|(     \__/  /   /  /__  ||  /__ __)|(     \__/ \__) (__) _|_

 Enter atleast 10 array elements,press '-1' for end of array:
30
50
20
10
90
70
80
30
60
40
-1
Array:
30  50  20  10  90  70  80  30  60  40  peak element: 50
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

## Part g

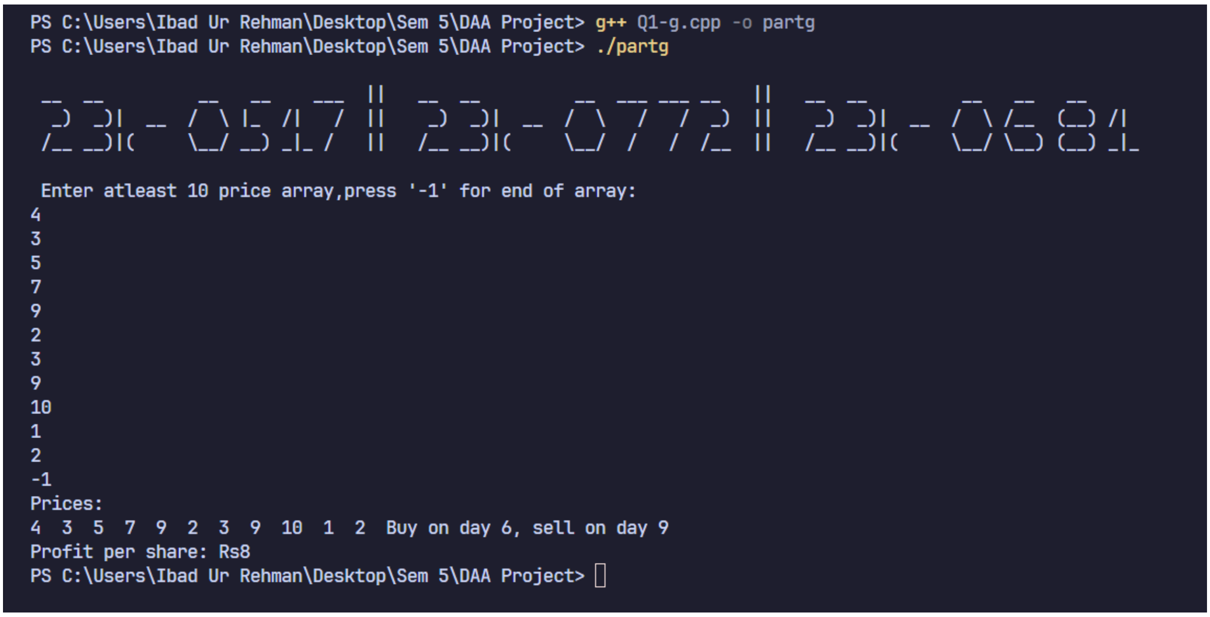
```
#include <iostream>
#include <vector>
using namespace std;

struct Result
{
    int buyDay;
    int sellDay;
    int profit;
};

Result maxProfitSplit(vector<int> &prices, int left, int right)
{
    if (left >= right)
        return {left, right, 0};
```



```
    int mid = left + (right - left) / 2;

    Result leftBest = maxProfitSplit(prices, left, mid);
    Result rightBest = maxProfitSplit(prices, mid + 1, right);

    // Cross case: buy in left half, sell in right half
    int minLeft = left;
    for (int i = left; i <= mid; i++)
    {
        if (prices[i] < prices[minLeft])
            minLeft = i;
    }

    int maxRight = mid + 1;
    for (int i = mid + 1; i <= right; i++)
    {
        if (prices[i] > prices[maxRight])
            maxRight = i;
    }

    int crossProfit = prices[maxRight] - prices[minLeft];
    Result crossResult = {minLeft, maxRight, crossProfit};

    // Return the best among the three
    if (leftBest.profit >= rightBest.profit && leftBest.profit >=
crossResult.profit)
        return leftBest;
    else if (rightBest.profit >= leftBest.profit && rightBest.profit >=
crossResult.profit)
        return rightBest;
    else
        return crossResult;
}

int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
    vector<int> inputArr;
    cout << " Enter atleast 10 price array,press '-1' for end of array: " <<
endl;
```

```
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }
    cout << "Prices: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }

    Result result = maxProfitSplit(inputArr, 0, inputArr.size() - 1);

    if (result.profit > 0)
    {
        cout << "Buy on day " << result.buyDay + 1 << ", sell on day " << 
result.sellDay + 1 << endl;
        cout << "Profit per share: Rs" << result.profit << endl;
    }
    else
    {
        cout << "No profit possible during this period." << endl;
    }
}
```

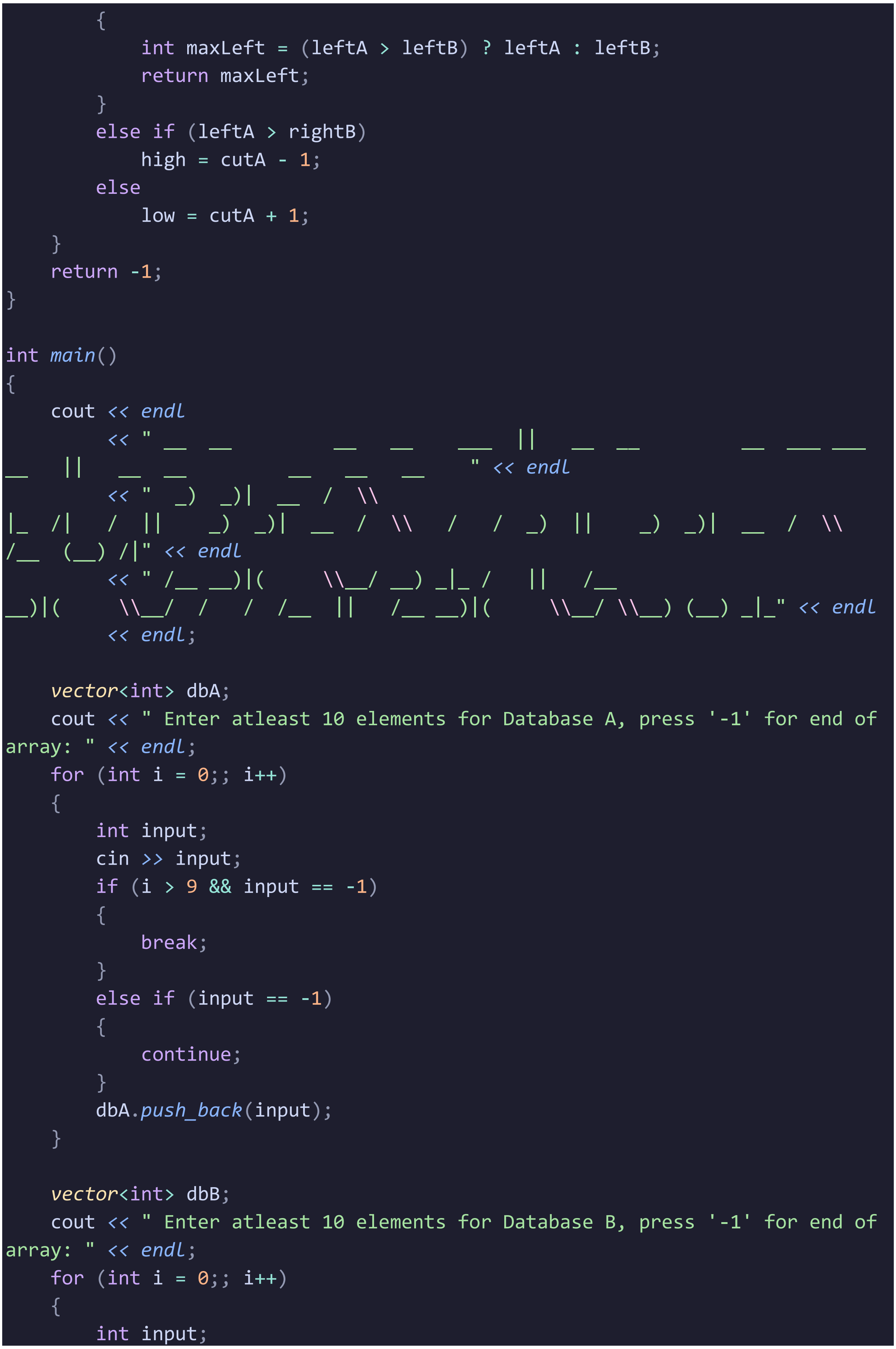
## Output



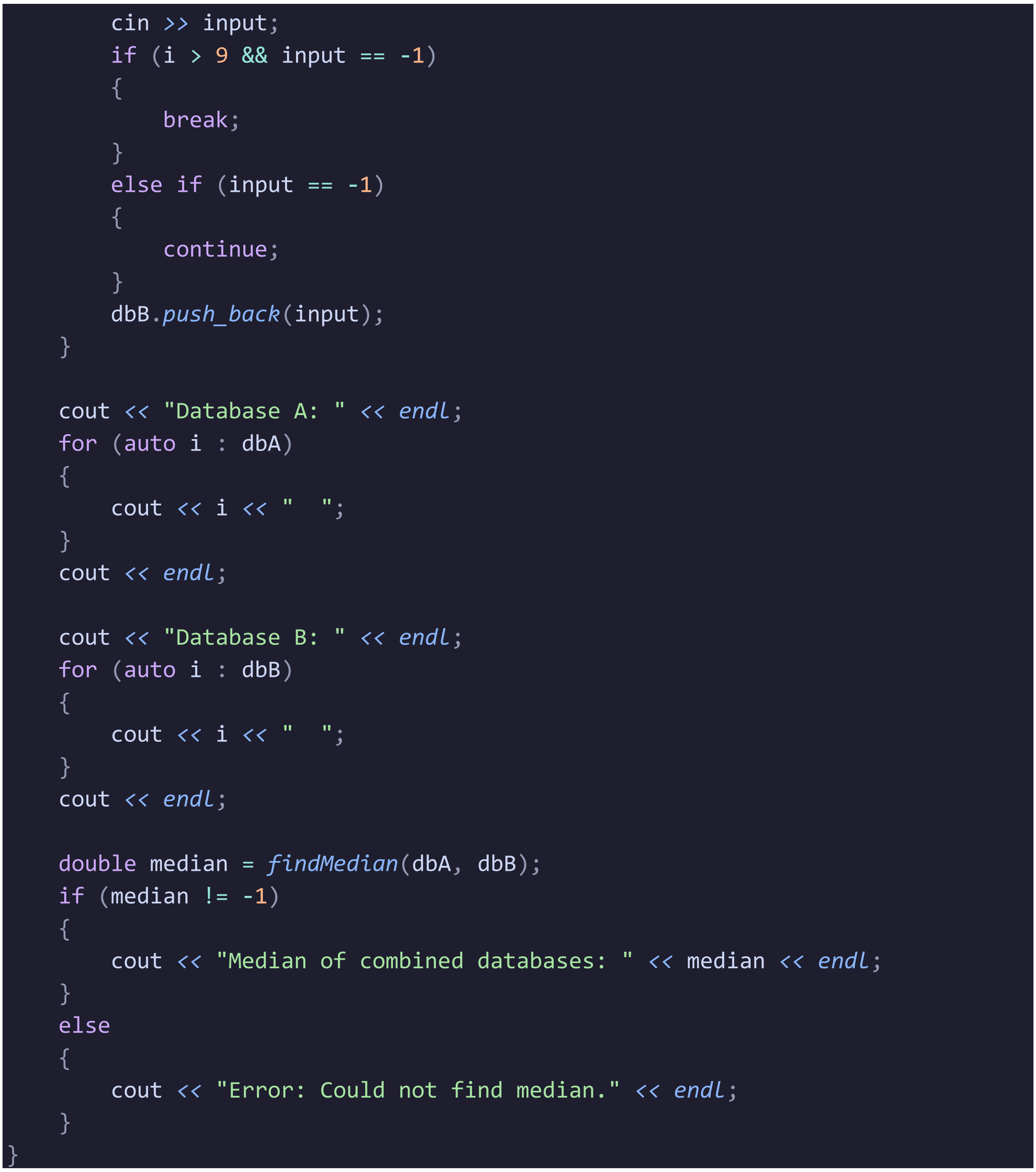
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-g.cpp -o partg
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./partg

 __  __         __   __    ___  ||   __  __         __  ___ ___ __   ||   __  __         __   __   __
  _)  _)|  __  / \  |_  /|   /  ||    _)  _)|  __  / \    /   /  _)  ||    _)  _)|  __  / \  /__  (__) /|
 /__ __)|(     \__/ __) _|_ /   ||   /__ __)|(     \__/  /   /  /__  ||   /__ __)|(     \__/ \__) (__) _|_

 Enter atleast 10 price array,press '-1' for end of array:
4
3
5
7
9
2
3
9
10
1
2
-1
Prices:
4  3  5  7  9  2  3  9  10  1  2  Buy on day 6, sell on day 9
Profit per share: Rs8
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```

## Part h – Ex1

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

double findMedian(vector<int> &a, vector<int> &b)
{
    int n = a.size();
    if (n == 0)
        return -1;
    if (a[0] > b[0])
    {
        vector<int> temp = a;
        a = b;
        b = temp;
    }
    int low = 0, high = n;
    while (low <= high)
    {
        int cutA = (low + high) / 2;
        int cutB = n - cutA;
        int leftA = (cutA == 0) ? INT_MIN : a[cutA - 1];
        int leftB = (cutB == 0) ? INT_MIN : b[cutB - 1];
        int rightA = (cutA == n) ? INT_MAX : a[cutA];
        int rightB = (cutB == n) ? INT_MAX : b[cutB];
        if (leftA <= rightB && leftB <= rightA)
```

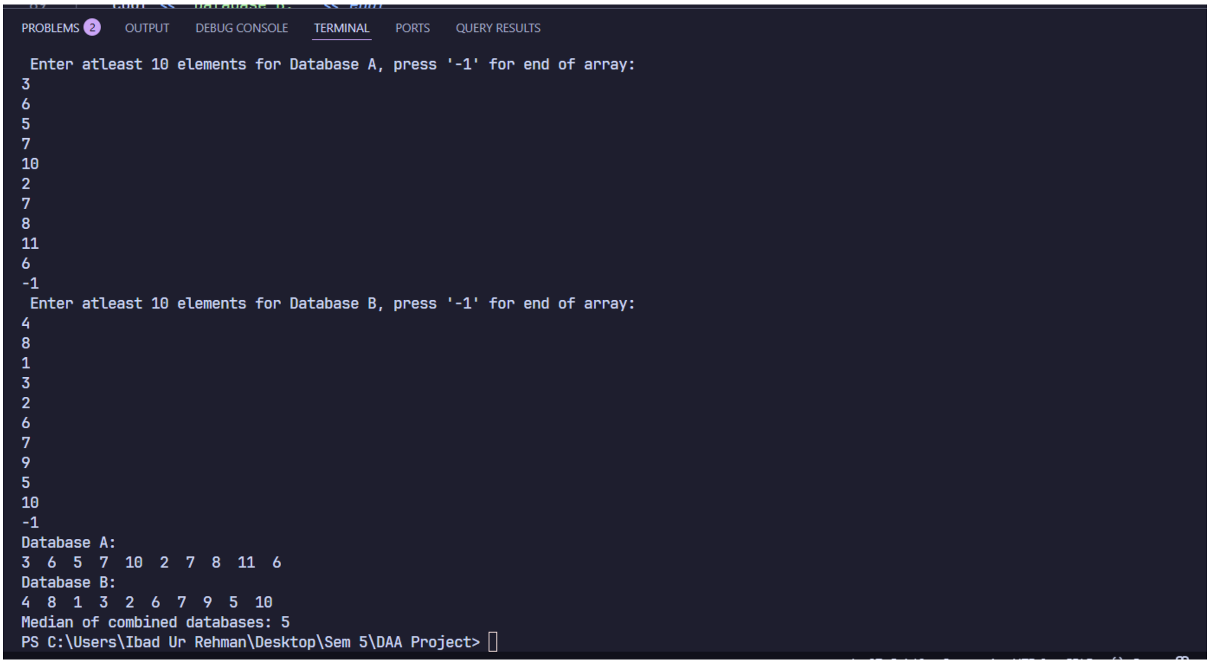


```
        {
            int maxLeft = (leftA > leftB) ? leftA : leftB;
            return maxLeft;
        }
        else if (leftA > rightB)
            high = cutA - 1;
        else
            low = cutA + 1;
    }
    return -1;
}

int main()
{
    cout << endl
         << " __  __         __   __     ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|    /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;

    vector<int> dbA;
    cout << " Enter atleast 10 elements for Database A, press '-1' for end of
array: " << endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        dbA.push_back(input);
    }

    vector<int> dbB;
    cout << " Enter atleast 10 elements for Database B, press '-1' for end of
array: " << endl;
    for (int i = 0;; i++)
    {
        int input;
```



```
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        dbB.push_back(input);
    }

    cout << "Database A: " << endl;
    for (auto i : dbA)
    {
        cout << i << "  ";
    }
    cout << endl;

    cout << "Database B: " << endl;
    for (auto i : dbB)
    {
        cout << i << "  ";
    }
    cout << endl;

    double median = findMedian(dbA, dbB);
    if (median != -1)
    {
        cout << "Median of combined databases: " << median << endl;
    }
    else
    {
        cout << "Error: Could not find median." << endl;
    }
}
```

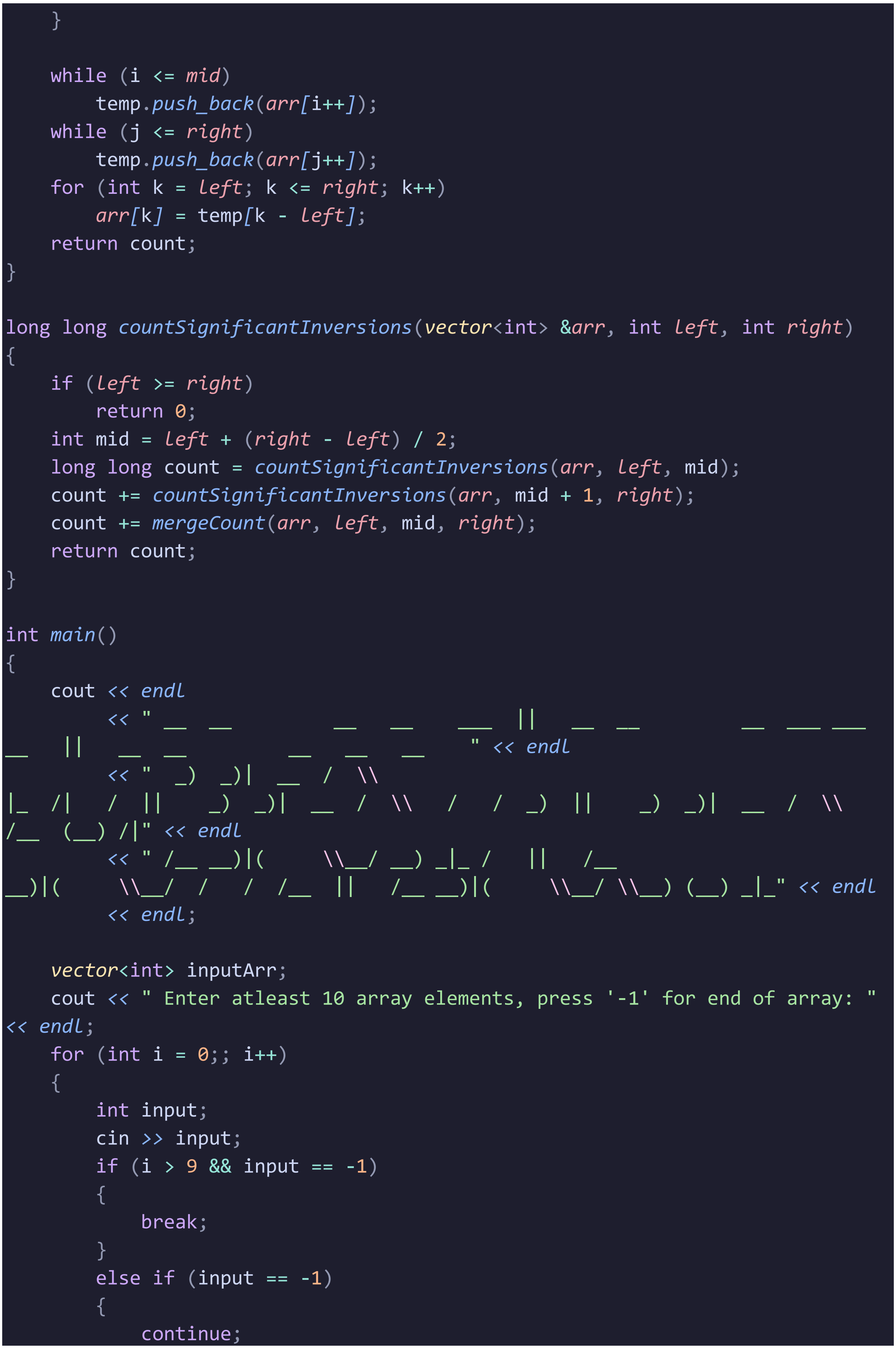
## Output



```
 Enter atleast 10 elements for Database A, press '-1' for end of array:
3
6
5
7
10
2
7
8
11
6
-1
 Enter atleast 10 elements for Database B, press '-1' for end of array:
4
8
1
3
2
6
7
9
5
10
-1
Database A:
3  6  5  7  10  2  7  8  11  6
Database B:
4  8  1  3  2  6  7  9  5  10
Median of combined databases: 5
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project>
```

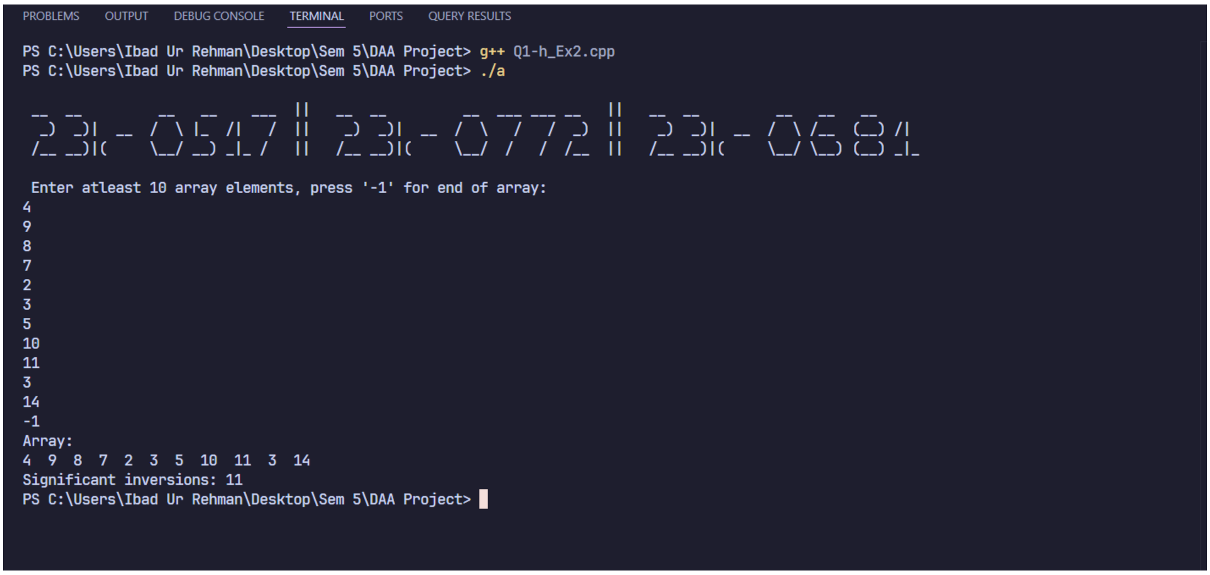
## Part h – Ex2

```cpp
#include <iostream>
#include <vector>

using namespace std;

long long mergeCount(vector<int> &arr, int left, int mid, int right)
{
    long long count = 0;
    int j = mid + 1;
    for (int i = left; i <= mid; i++)
    {
        while (j <= right && (long long)arr[i] > 2LL * arr[j])
            j++;
        count += j - (mid + 1);
    }

    vector<int> temp;
    int i = left;
    j = mid + 1;
    while (i <= mid && j <= right)
    {
        if (arr[i] <= arr[j])
            temp.push_back(arr[i++]);
        else
            temp.push_back(arr[j++]);
```



```
    }

    while (i <= mid)
        temp.push_back(arr[i++]);
    while (j <= right)
        temp.push_back(arr[j++]);
    for (int k = left; k <= right; k++)
        arr[k] = temp[k - left];
    return count;
}

long long countSignificantInversions(vector<int> &arr, int left, int right)
{
    if (left >= right)
        return 0;
    int mid = left + (right - left) / 2;
    long long count = countSignificantInversions(arr, left, mid);
    count += countSignificantInversions(arr, mid + 1, right);
    count += mergeCount(arr, left, mid, right);
    return count;
}

int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __    " << endl
         << "  _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;

    vector<int> inputArr;
    cout << " Enter atleast 10 array elements, press '-1' for end of array: "
<< endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        inputArr.push_back(input);
    }

    cout << "Array: " << endl;
    for (auto i : inputArr)
    {
        cout << i << "  ";
    }
    cout << endl;

    long long result = countSignificantInversions(inputArr, 0, inputArr.size()
- 1);
    cout << "Significant inversions: " << result << endl;
}
```

## Output

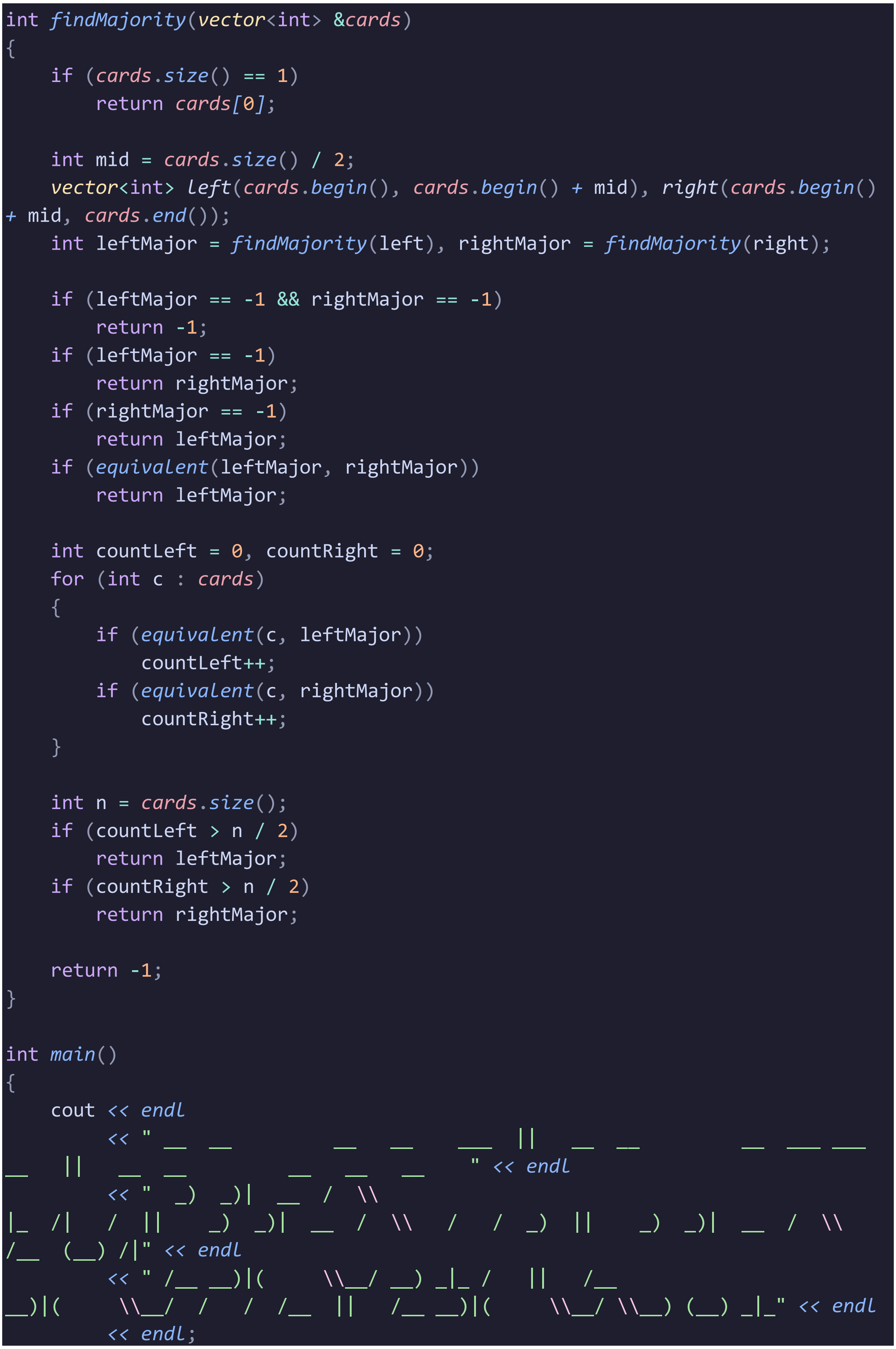


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

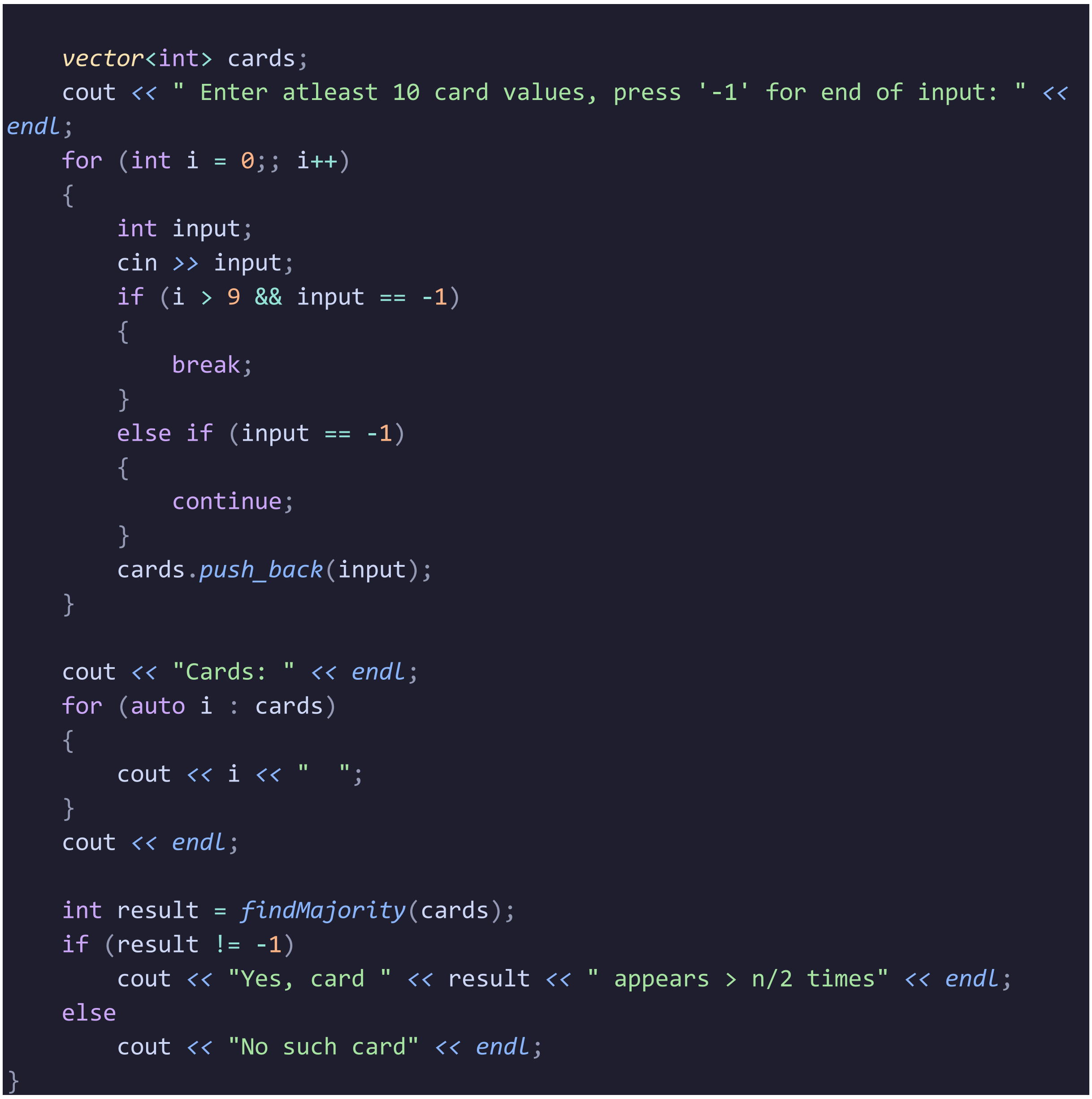
```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-h_Ex2.cpp
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./a

 __  __        __   __   ___  ||   __  __        __  ___ ___ __   ||   __  __        __   __   __
  _)  _)|  __  /  \ |_  /|   /  ||    _)  _)|  __  /  \   /   /  _)  ||    _)  _)|  __  /  \ /__  (__) /|
 /__ __)|(     \__/ __) _|_ /   ||   /__ __)|(     \__/  /   /  /__  ||   /__ __)|(     \__/ \__) (__) _|_

 Enter atleast 10 array elements, press '-1' for end of array:
4
9
8
7
2
3
5
10
11
3
14
-1
Array:
4  9  8  7  2  3  5  10  11  3  14
Significant inversions: 11
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> █
```

## Part h – Ex 3

```
#include <iostream>
#include <vector>

using namespace std;

bool equivalent(int a, int b)
{
    return a == b;
}
```

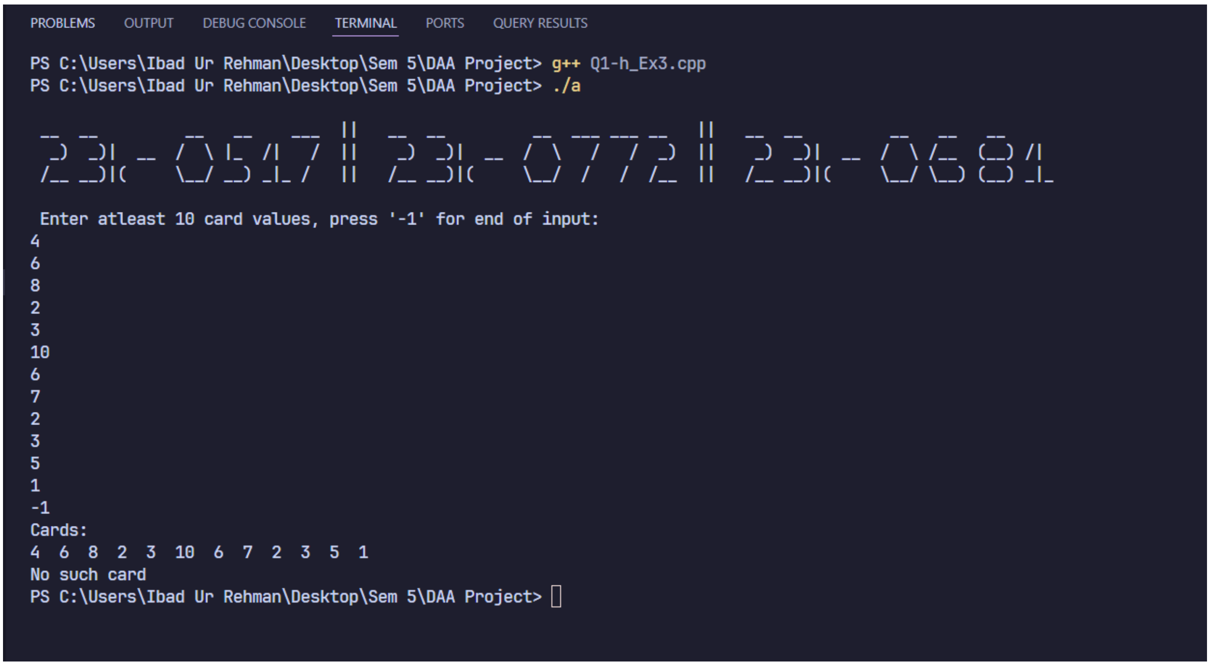


```cpp
int findMajority(vector<int> &cards)
{
    if (cards.size() == 1)
        return cards[0];

    int mid = cards.size() / 2;
    vector<int> left(cards.begin(), cards.begin() + mid), right(cards.begin()
+ mid, cards.end());
    int leftMajor = findMajority(left), rightMajor = findMajority(right);

    if (leftMajor == -1 && rightMajor == -1)
        return -1;
    if (leftMajor == -1)
        return rightMajor;
    if (rightMajor == -1)
        return leftMajor;
    if (equivalent(leftMajor, rightMajor))
        return leftMajor;

    int countLeft = 0, countRight = 0;
    for (int c : cards)
    {
        if (equivalent(c, leftMajor))
            countLeft++;
        if (equivalent(c, rightMajor))
            countRight++;
    }

    int n = cards.size();
    if (countLeft > n / 2)
        return leftMajor;
    if (countRight > n / 2)
        return rightMajor;

    return -1;
}

int main()
{
    cout << endl
         << " __  __         __   __    ___  ||   __  __         __  ___ ___
__   ||   __  __         __   __   __     " << endl
         << "   _)  _)|  __  /  \\
|_  /|   /  ||    _)  _)|  __  /  \\   /   /  _)  ||    _)  _)|  __  /  \\
/__  (__) /|" << endl
         << " /__ __)|(     \\__/ __) _|_ /   ||   /__
__)|(     \\__/  /   /  /__  ||   /__ __)|(     \\__/ \\__) (__) _|_" << endl
         << endl;
```



```
    vector<int> cards;
    cout << " Enter atleast 10 card values, press '-1' for end of input: " <<
endl;
    for (int i = 0;; i++)
    {
        int input;
        cin >> input;
        if (i > 9 && input == -1)
        {
            break;
        }
        else if (input == -1)
        {
            continue;
        }
        cards.push_back(input);
    }

    cout << "Cards: " << endl;
    for (auto i : cards)
    {
        cout << i << "  ";
    }
    cout << endl;

    int result = findMajority(cards);
    if (result != -1)
        cout << "Yes, card " << result << " appears > n/2 times" << endl;
    else
        cout << "No such card" << endl;
}
```

# Output



```
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> g++ Q1-h_Ex3.cpp
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> ./a

 __ __       __   __   ___  ||  __ __      __  ___ ___ __  ||  __ __      __   __   __
  _) _)|  _  /  \ |_  /|   /  ||   _) _)|  _  /  \   /   /  _) ||   _) _)|  _  /  \ /__  (__) /|
 /__ __)|(    \__/ __) _|_ /   ||  /__ __)|(    \__/  /   /  /__ ||  /__ __)|(    \__/ \__) (__) _|_

 Enter atleast 10 card values, press '-1' for end of input:
4
6
8
2
3
10
6
7
2
3
5
1
-1
Cards:
4  6  8  2  3  10  6  7  2  3  5  1
No such card
PS C:\Users\Ibad Ur Rehman\Desktop\Sem 5\DAA Project> 
```