**Problem Statement:** Design and implement the Min-Max algorithm using the Divide and Conquer method and compare this with the basic method to find the minimum and maximum elements from the given list. Determine the number of comparisons required to find the minimum and maximum for a large value of *n*.

**Brief About the Problem:** Finding the minimum and maximum values from an array is a fundamental problem in computer science. The two primary approaches used are:

1. **Brute Force Method:** This method sequentially iterates through the array, checking each element to find the minimum and maximum values.
2. **Divide and Conquer Method:** This method recursively splits the array into smaller subarrays, finds the min and max in those subarrays, and then combines the results efficiently.

The goal is to analyze and compare the efficiency of these methods in terms of time complexity and the number of comparisons.

**Algorithm:**

1. **Brute Force Approach:**

* Initialize min to maximum possible value and max to minimum possible value.
* Iterate through the array.
* Update min if the current element is smaller.
* Update max if the current element is larger.
* Return min and max.
* Time Complexity: **O(n)**
* Number of comparisons: **2(n - 1)**

1. **Divide and Conquer Approach:**

* If the array contains a single element, return it as both min and max.
* If the array contains two elements, compare them and assign min and max.
* Otherwise, divide the array into two halves.
* Recursively find the min and max in both halves.
* Compare the results of both halves to determine the final min and max.
* Time Complexity: **O(n)** (better in terms of comparisons than brute force)
* Number of comparisons: **Approximately 3n/2**

**Analysis of Algorithm:**

* The brute force method makes **2(n - 1)** comparisons in the worst case.
* The divide and conquer method reduces the number of comparisons to **1.5n** approximately.
* Thus, the divide and conquer method is more efficient in terms of the number of comparisons, especially for large values of n.

**Program Implementation:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <climits>

#include <chrono>

using namespace std;

using namespace std::chrono;

int bruteForceComparisons = 0;

int divideAndConquerComparisons = 0;

void MinMax(int arr[], int size, int result[]) {

    int min = INT\_MAX;

    int max = INT\_MIN;

    for (int i = 0; i < size; i++) {

        if (arr[i] > max) {

            max = arr[i];

        }

        if (arr[i] < min) {

            min = arr[i];

        }

        bruteForceComparisons += 2; // Two comparisons per iteration

    }

    result[0] = min;

    result[1] = max;

}

void DandQMinMax(int arr[], int low, int high, int &min, int &max) {

    if (low == high) {  // Single element

        min = max = arr[low];

        return;

    }

    if (high == low + 1) {  // Two elements

        divideAndConquerComparisons++;

        if (arr[low] < arr[high]) {

            min = arr[low];

            max = arr[high];

        } else {

            min = arr[high];

            max = arr[low];

        }

        return;

    }

    int mid = (low + high) / 2;

    int min1, max1, min2, max2;

    DandQMinMax(arr, low, mid, min1, max1);

    DandQMinMax(arr, mid + 1, high, min2, max2);

    divideAndConquerComparisons++;

    min = (min1 < min2) ? min1 : min2;

    divideAndConquerComparisons++;

    max = (max1 > max2) ? max1 : max2;

}

int main() {

    srand(time(0));

    int size;

    cout << "Enter the size of the array: ";

    cin >> size;

    if (size <= 0) {

        cout << "Invalid array size!" << endl;

        return 1;

    }

    int \*arr = new int[size];  // Dynamic memory allocation

    int result[2];

    // Generate random numbers in a reasonable range

    for (int i = 0; i < size; i++) {

        arr[i] = rand() % 10000;  // Random numbers between 0 and 9999

    }

    // Brute Force Approach

    auto start1 = high\_resolution\_clock::now();

    MinMax(arr, size, result);

    auto end1 = high\_resolution\_clock::now();

    auto timeOfMinMax = duration\_cast<microseconds>(end1 - start1);

    cout << "\nBrute Force Approach\n";

    cout << "Minimum Element: " << result[0] << endl;

    cout << "Maximum Element: " << result[1] << endl;

    cout << "Time taken: " << timeOfMinMax.count() << " microseconds" << endl;

    cout << "Comparisons: " << bruteForceComparisons << endl;

    // Divide and Conquer Approach

    int min, max;

    auto start2 = high\_resolution\_clock::now();

    DandQMinMax(arr, 0, size - 1, min, max);

    auto end2 = high\_resolution\_clock::now();

    auto timeOfDandQ = duration\_cast<microseconds>(end2 - start2);

    cout << "\nDivide & Conquer Approach\n";

    cout << "Minimum Element: " << min << endl;

    cout << "Maximum Element: " << max << endl;

    cout << "Time taken: " << timeOfDandQ.count() << " microseconds" << endl;

    cout << "Comparisons: " << divideAndConquerComparisons << endl;

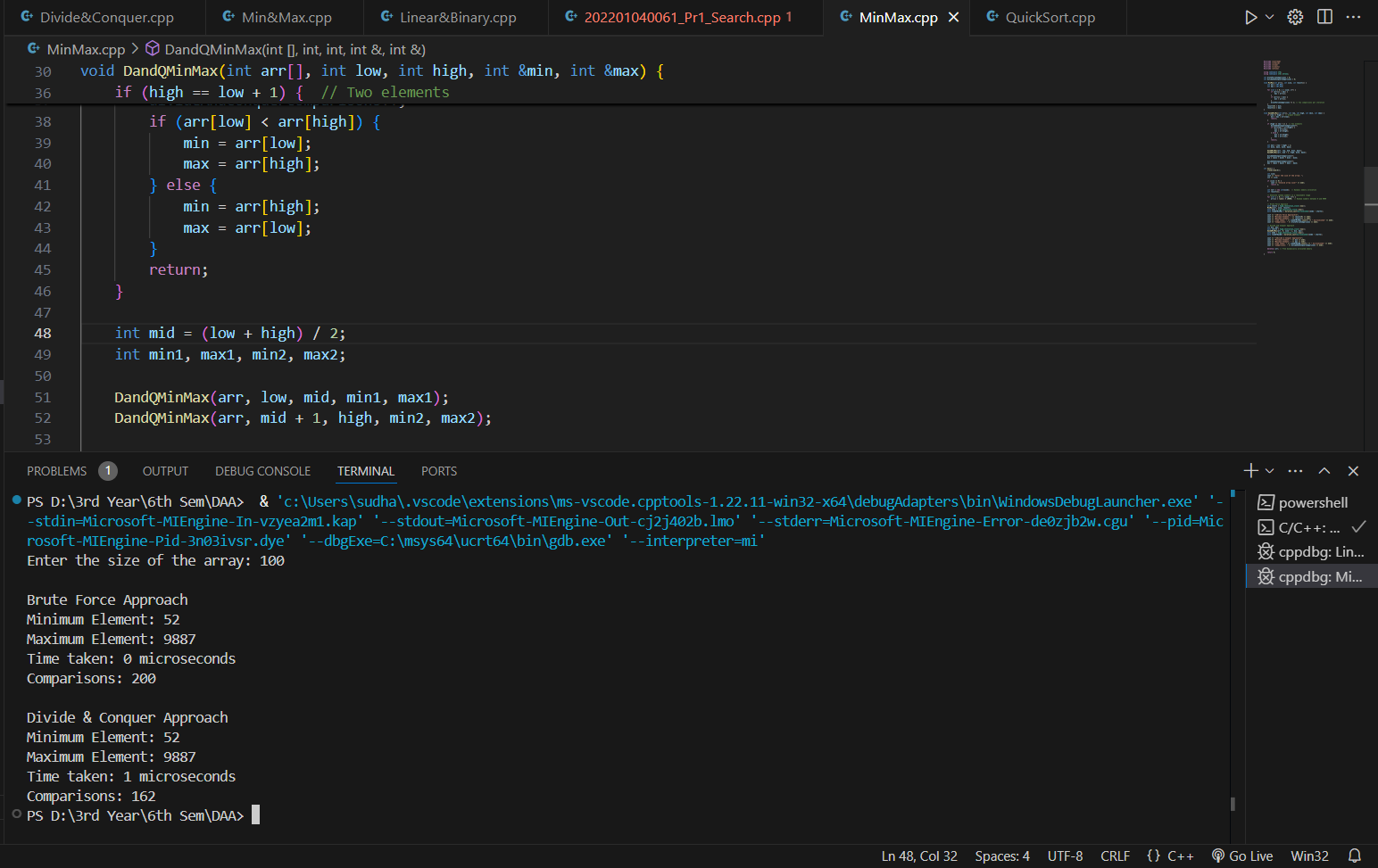
    delete[] arr; // Free dynamically allocated memory

    return 0;

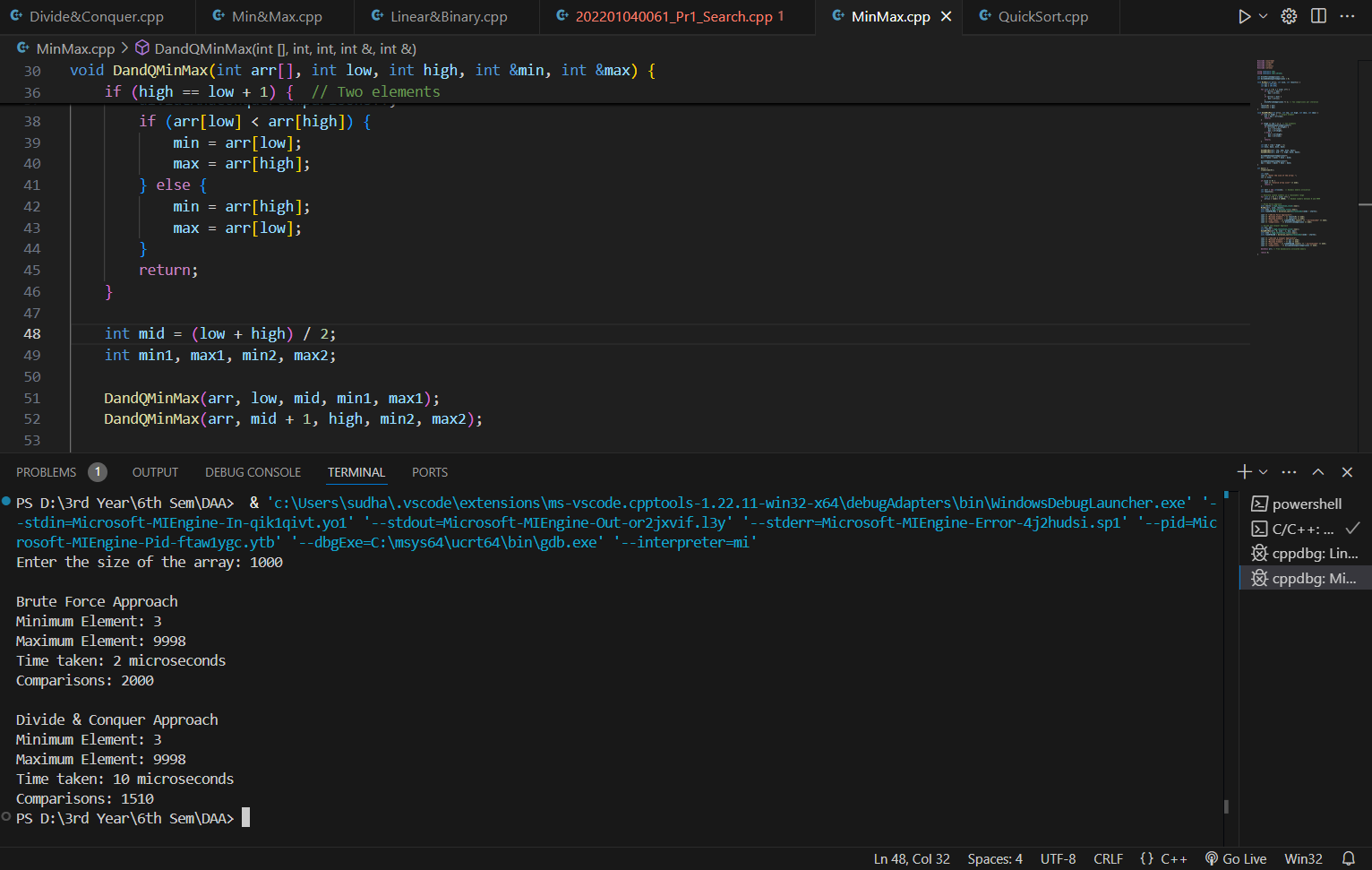
}

**Output Screenshot:**

**For 100 Elements**

****

**For 1000 Elements**

****

**For 5000 Elements**

**A screenshot of a computer

Description automatically generated**

**For 10000 Elements**

**A screenshot of a computer program

Description automatically generated**

**For 50000 Elements**

**A screenshot of a computer

Description automatically generated**

**Tables:**

|  |  |  |
| --- | --- | --- |
| **Number of Elements** | **Brute Force Comparisons** | **Divide & Conquer Comparisons** |
| 100 | 200 | 162 |
| 1000 | 2000 | 1510 |
| 5000 | 10000 | 7950 |
| 10000 | 20000 | 15902 |
| 50000 | 100000 | 82766 |

**Graph:**

