KATHMANDU UNIVERSIY

REPORT : LAB 5

Submitted By:
Name: Bikraj Shrestha
Roll:50

Name: Janak Sitaula
Roll:51

Group : CE
2nd Year/1st Sem

Submitted To:
Dr. Rajani Chulyadyo
Undergraduate Coordinator CE Program
Department of Computer Science and Engineering

sorting.h M   sorting.cpp   main.cpp M

main.cpp > main()

```cpp
{
    __int64 size = 100000;
    vector<__int64> timerLogInsertion;
    vector<__int64> timerLogQuick;
    fstream fout;
    fout.open("random2.csv", ios::out | ios::app);
    fout << "Size of Array"
         << "Insertion Sort"
         << "Quick Sort" << endl;
    for (__int64 i = 1; i < size; i += 1000)
    {
        vector<__int64> arrayForInsertion;
        randomArray(arrayForInsertion, i);      You, 7 seconds ago • Uncommitted changes
        vector<__int64> arrayForQuick;
        arrayForQuick = arrayForInsertion;
        cout << i << endl;
        __int64 nowInsertion = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count();
        insertionsort(arrayForInsertion);
        timerLogInsertion.push_back(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count() - nowInsertion);
        cout << i << "  " << std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count() - nowInsertion << endl;
        cout << i << endl;
        __int64 nowQuick = std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count();
        quickSort(arrayForQuick,0,arrayForQuick.size()-1);
        timerLogQuick.push_back(std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count() - nowQuick);
        cout << i << "  " << std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count() - nowQuick << endl;
    }
    for (__int64 i = 0; i < timerLogInsertion.size(); i++)
    {
        static int j = 1;
        fout << j << "," << timerLogInsertion[i] << "," << timerLogQuick[i] << endl;
        j += 1000;
    }
```

Help sorting.cpp - CE2020_Lab5_50_51 - Visual Studio Code

sorting.h M | sorting.cpp × | main.cpp M

rc > sorting.cpp > quickSort(vector<__int64>&, __int64, __int64)

```cpp
// For example : Array = {1,15,3,4,7,8,9}
// if 3 were to be the pivot element its position in the array would be second place
// Repeat this until you find the position for all the elements in the array
void quickSort(vector<__int64> &array, __int64 low, __int64 high)        unknown, 7 hours ago • added __int64 instead of int
{
    if (low < high)
    {
        __int64 pivot = partition(array, low, high);
        quickSort(array, low, pivot - 1);
        quickSort(array, pivot + 1, high);
    }
}

//  __int64 partition(vec                t64 low,  __int64 high)···
__int64 partition(vector<__int64> &array, __int64 low, __int64 high)
{
    __int64 pivot = array[high];
    __int64 i = (low - 1);
    for (__int64 j = low; j <= high - 1; j++)
    {

        if (array[j] < pivot)
        {
            i++;
            swapElement(array, i, j);
        }
    }
    swapElement(array, i + 1, high);
    return (i + 1);
}
// swapElement the element in the given positions
void swapElement(vector<__int64> &array, __int64 firstPosi, __int64 secondPosi)
{
    __int64 temp = array[firstPosi];
    array[firstPosi] = array[secondPosi];
    array[secondPosi] = temp;
}
```

#define __int64 long long

Expands to:

long long

al Help sorting.cpp - CE2020_Lab5_50_51 - Visual Studio Code

sorting.h M | sorting.cpp × | main.cpp M

src > sorting.cpp > insertionsort(vector<__int64>&)

```cpp
}

void randomArrary(vector<__int64> &array, __int64 size)
{
    for (__int64 i = 0; i < size; i++)
    {
        array.push_back(i);
    }
    std::random_device rd;
    std::mt19937 g(rd());

    shuffle(array.begin(), array.end(), g);
}

void insertionsort(vector<__int64> &array){

    for(int j=1; j<array.size()-1; j++){
        int key = array [j];
        int i= j-1;
        while (i>=0 && array[i]>key){
            array[i+1]= array[i];
            i=i-1;
        }
        array[i+1] = key;        sitjan, 3 hours ago • added …
    }
}
```

githd: Express Off Git Graph Reconnect to Discord Ln 91, Col 26 Spaces: 4 UTF-8 CRLF C++ Go Live Win32 ✓ Prettier

Quick Sort (Sorting Algorithm)

A quick sort algorithm is a based on divide and conquer strategy. The main problem is divided into sub problem and these sub problems are solved seperately. In this case the array is divided by a pivot element where the left of the pivot element contains all the elements that are greater than the pivot and the elements lesser than the pivot element are kept on the right side of the array.

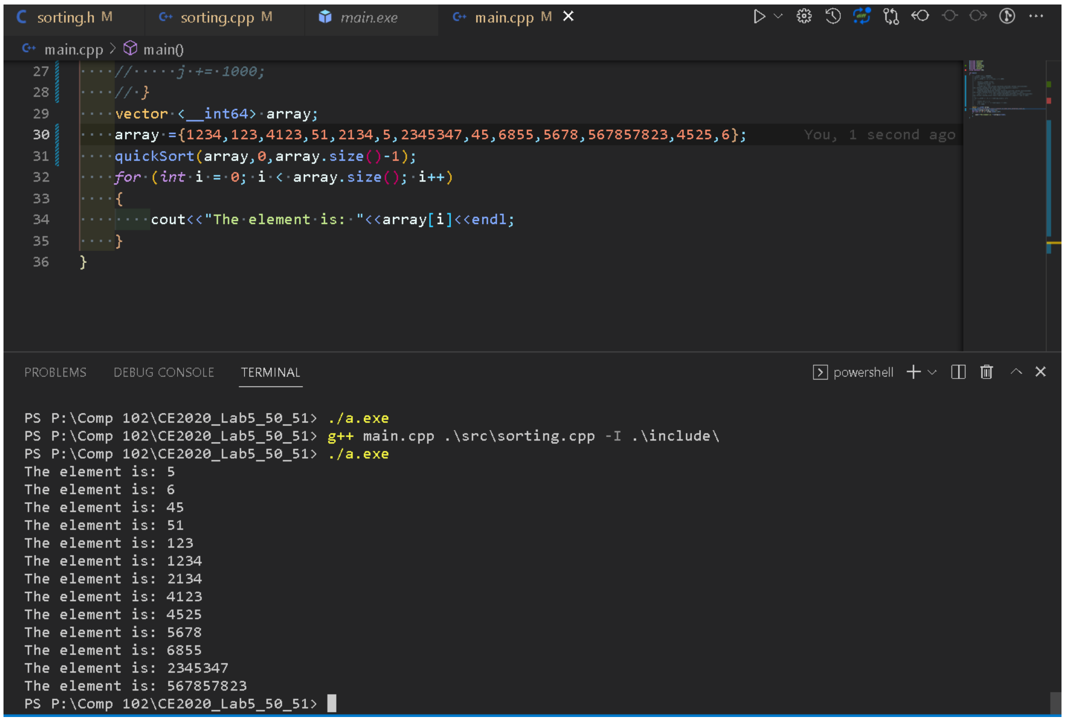


Figure: Output Screen for contents of sorted Array

The quickSort()algorithm uses a main recursive function and a partition function. The partition function does most of the sorting in the array. It takes the first element as a pivot and divided the array in two halves with halves having element greater than the pivot and the other having more than the pivot. The swap function simply swaps the element in the given index.

InsertionSort (Sorting Algorithm)

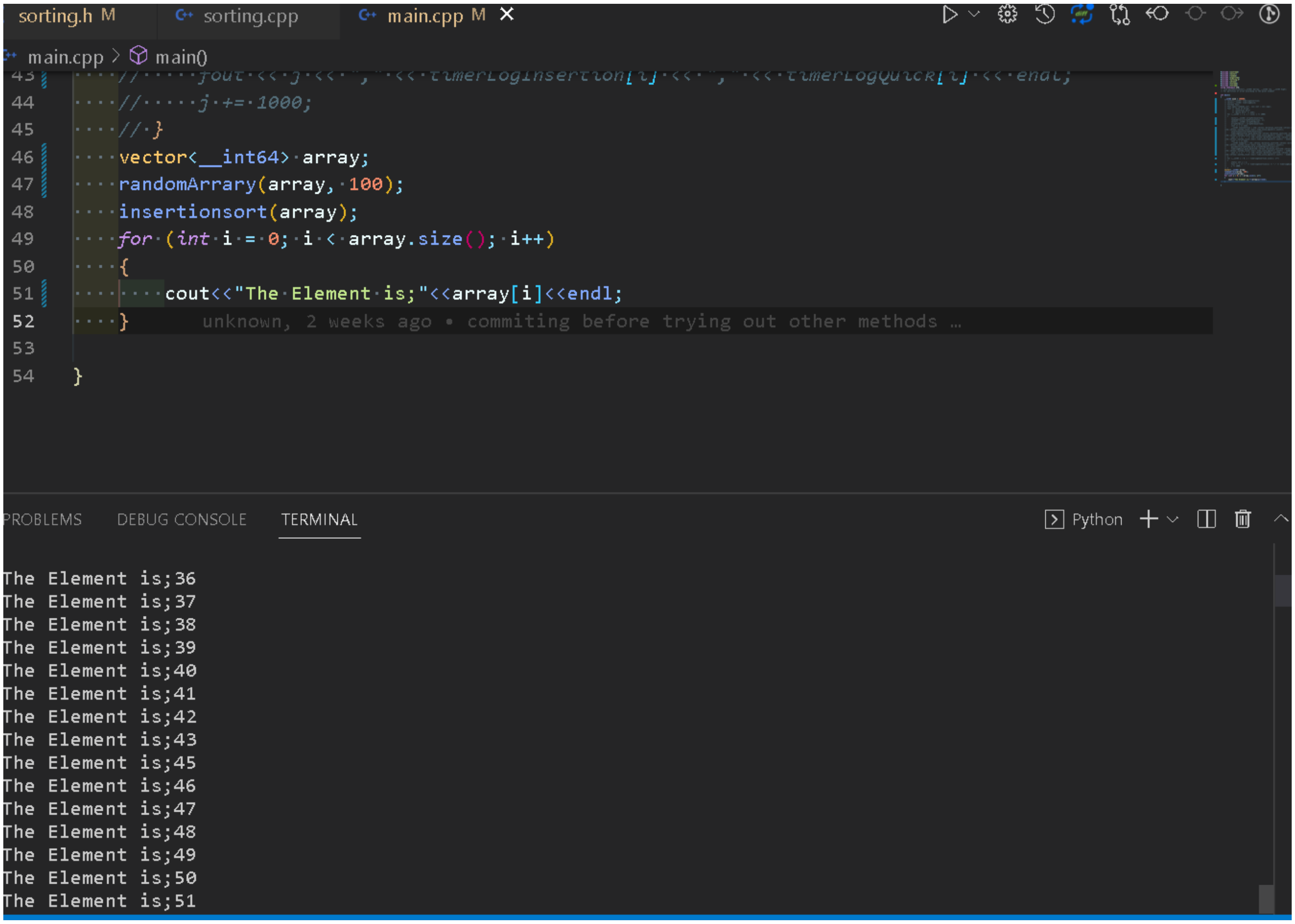
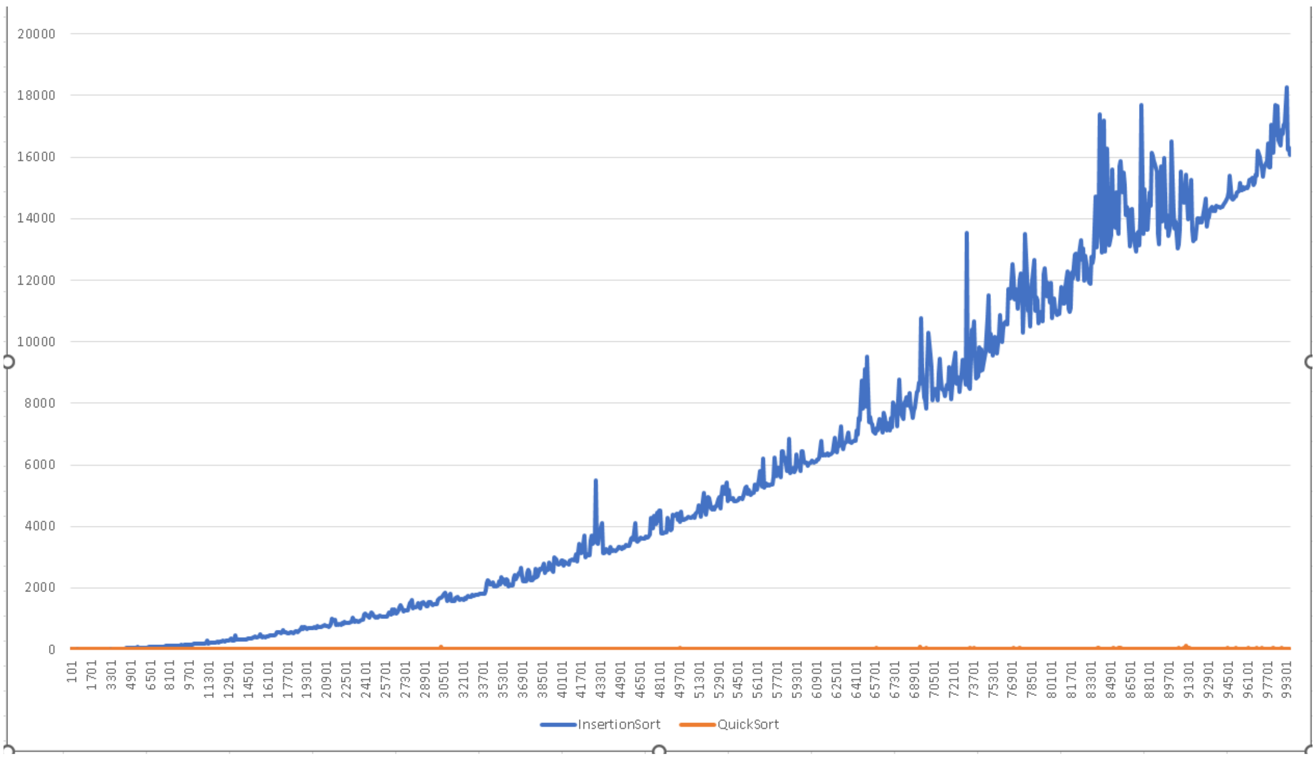


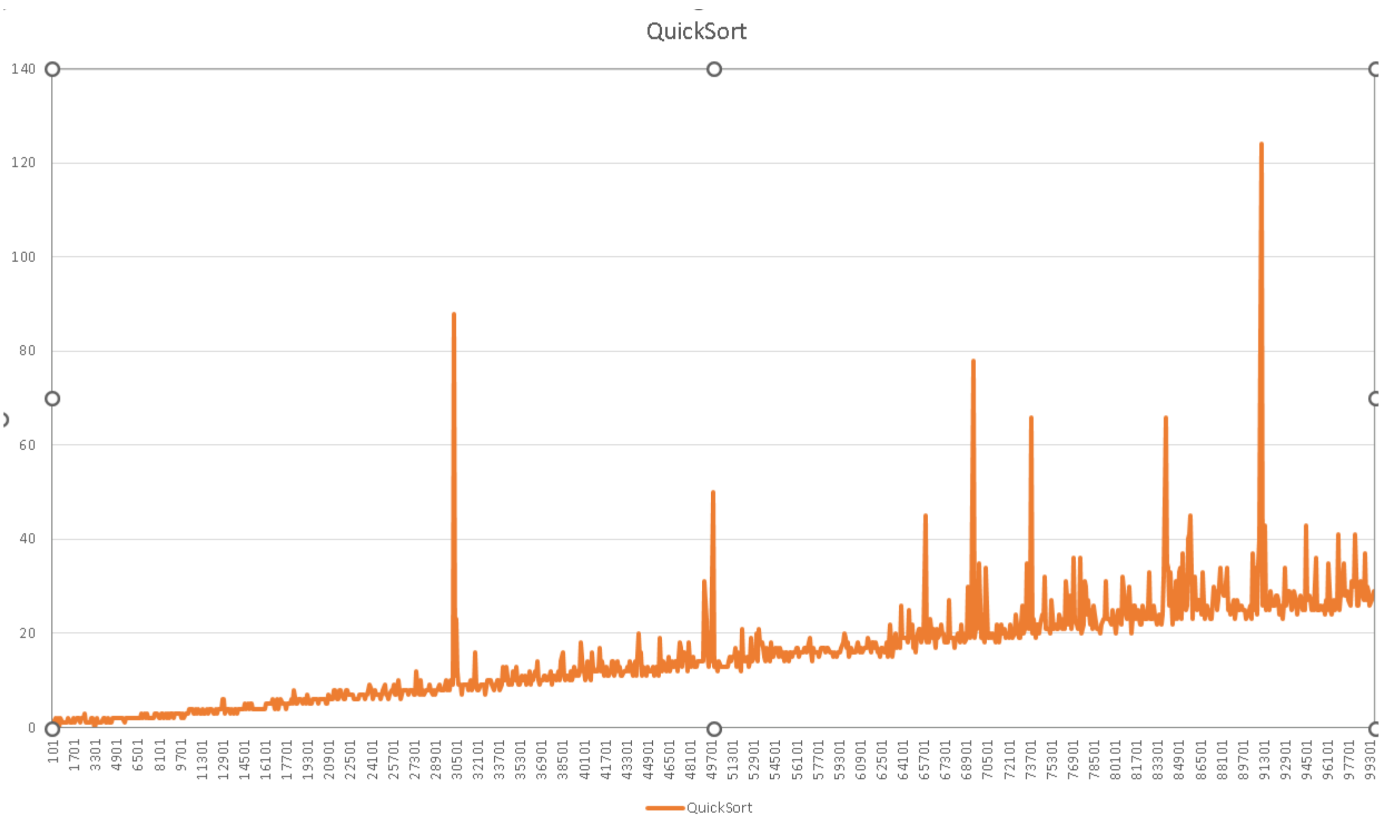
Figure: Output Screen for contents of sorted Array

Time Complexity

The graph of size of array vs the time taken to sort the array by both the algorithms: QuickSort and InsertionSort clearly shows that the insertion takes more time to sort the same array as compared to the quickSort. This difference starts getting clearer when the size of array increased from 5000.This is because the average time Complexity for Insertion Sort is n squared while that for quickSort is nLogn. The high abnormalities are seen due to the background application running while executing this program.

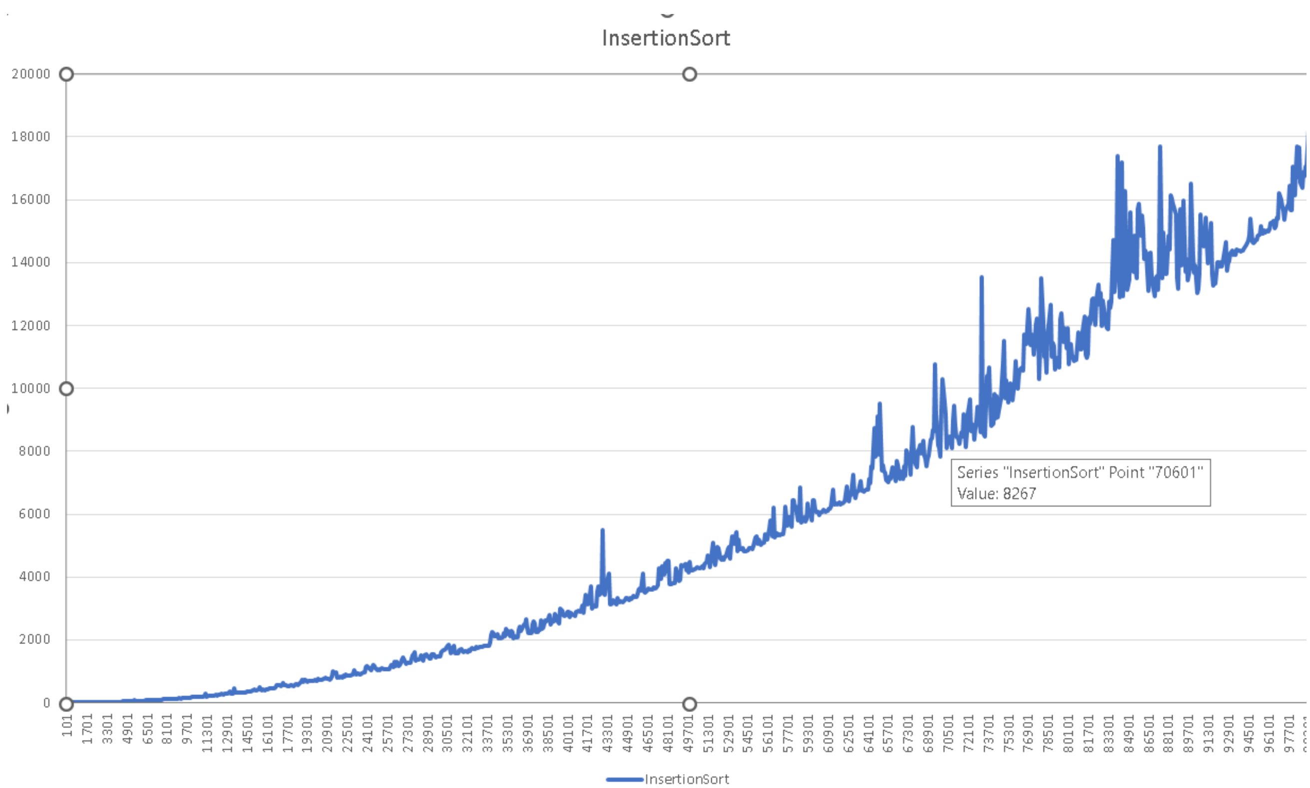


QuickSort



Time Complexity:
Average Case:O(nlogn)
Best Case:O(nlogn)
Worst Case: O(n^2)

InsertionSort



Time Complexity:

Average Case: O(n^2)

Best Case:O(n)

Worst Case: O(n^2)