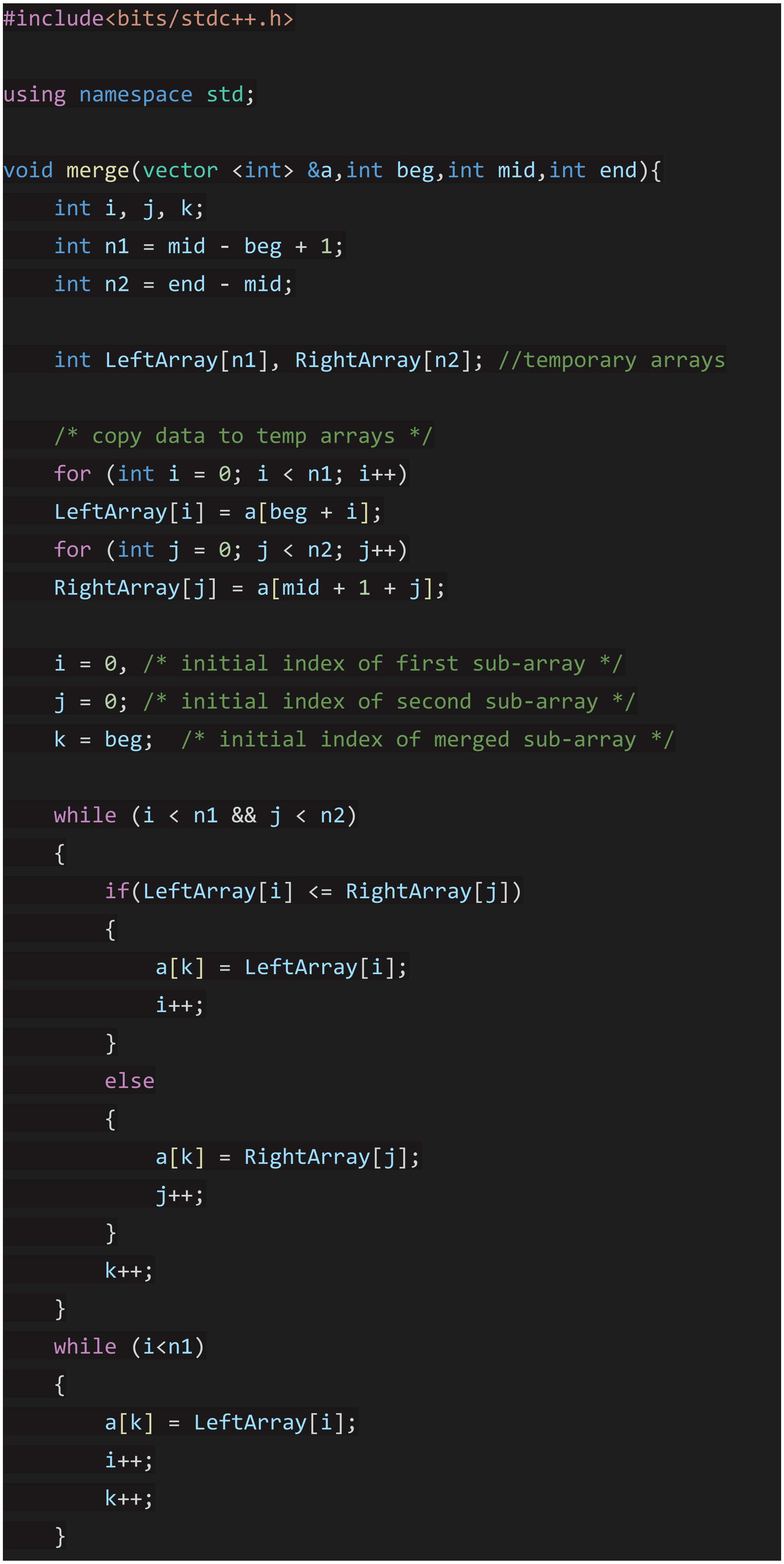
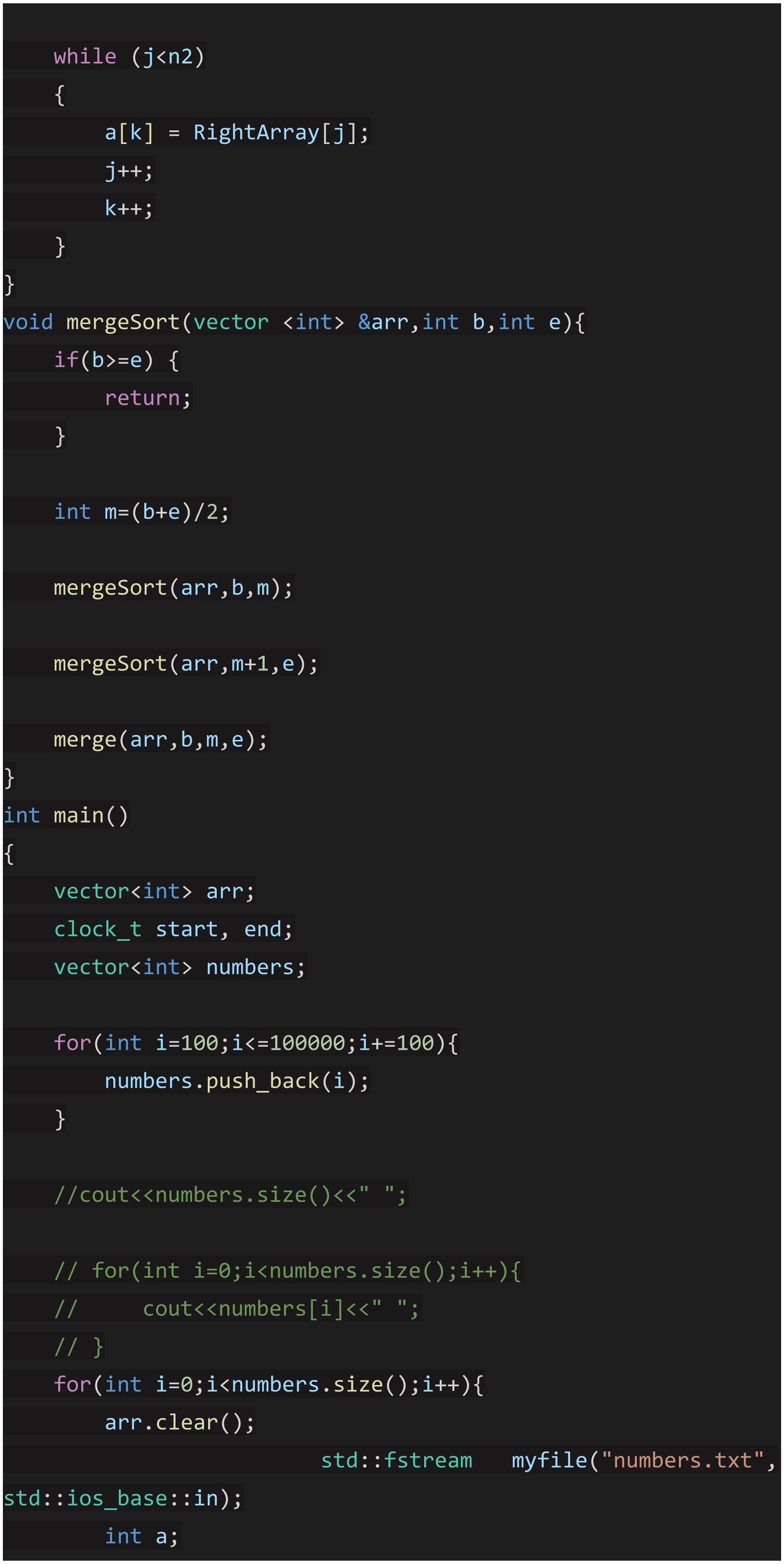
NAME:	Harshith Kunder
UID:	2021300068
SUBJECT	Design and Analysis of Algorithms.
EXPERIMENT NO :	2. Extras
PROBLEM STATEMENT 1:	Experiment on finding the running time of Merge sort and count of swaps using random pivot and last pivot.

Program:	**Merge Sort:**



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

using namespace std;

void merge(vector <int> &a,int beg,int mid,int end){
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;

    int LeftArray[n1], RightArray[n2]; //temporary arrays

    /* copy data to temp arrays */
    for (int i = 0; i < n1; i++)
    LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
    RightArray[j] = a[mid + 1 + j];

    i = 0, /* initial index of first sub-array */
    j = 0; /* initial index of second sub-array */
    k = beg;  /* initial index of merged sub-array */

    while (i < n1 && j < n2)
    {
        if(LeftArray[i] <= RightArray[j])
        {
            a[k] = LeftArray[i];
            i++;
        }
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
        a[k] = LeftArray[i];
        i++;
        k++;
    }
```



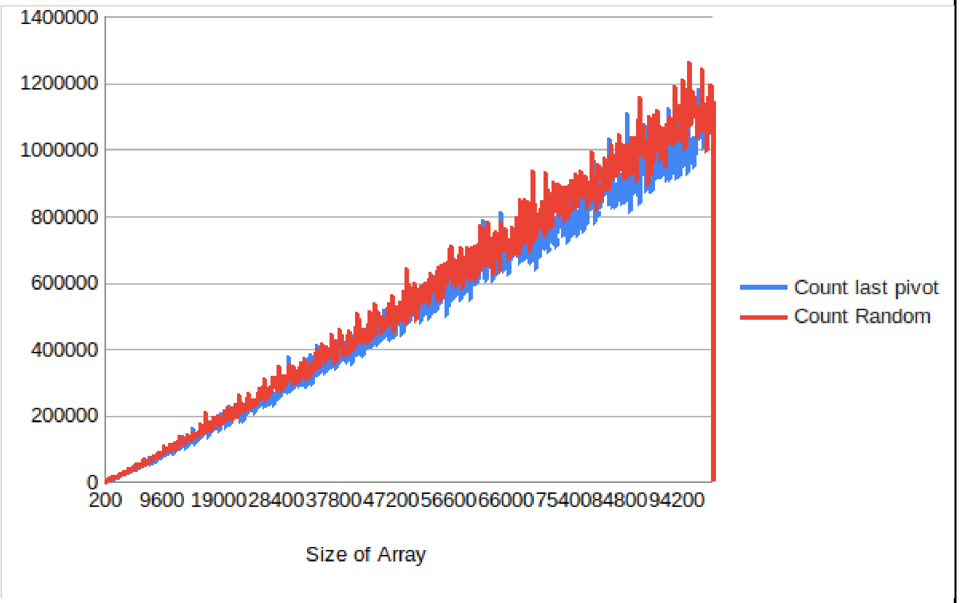
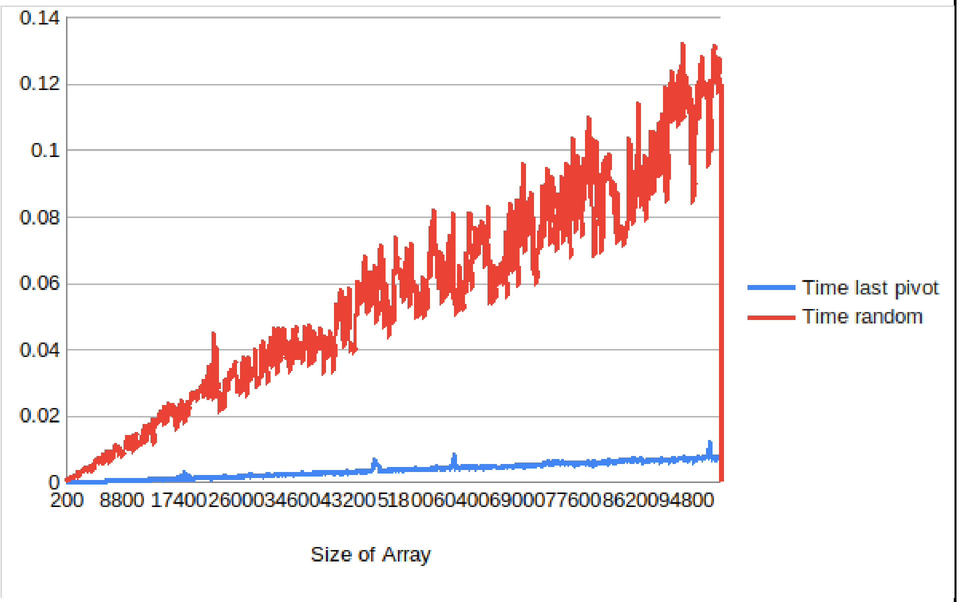
```
    while (j<n2)
    {
        a[k] = RightArray[j];
        j++;
        k++;
    }
}
void mergeSort(vector <int> &arr,int b,int e){
    if(b>=e) {
        return;
    }

    int m=(b+e)/2;

    mergeSort(arr,b,m);

    mergeSort(arr,m+1,e);

    merge(arr,b,m,e);
}
int main()
{
    vector<int> arr;
    clock_t start, end;
    vector<int> numbers;

    for(int i=100;i<=100000;i+=100){
        numbers.push_back(i);
    }

    //cout<<numbers.size()<<" ";

    // for(int i=0;i<numbers.size();i++){
    //     cout<<numbers[i]<<" ";
    // }
    for(int i=0;i<numbers.size();i++){
        arr.clear();
                         std::fstream   myfile("numbers.txt",
std::ios_base::in);
        int a;
        for(int j=0 ; j<numbers[i] ; j++){
            myfile>>a;
            arr.push_back(a);
        }
        /* Recording the time.*/
        start = clock();

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

        end = clock();

        // Calculating total time taken by the program.
               double  time_taken  =  double(end  -  start)  /
double(CLOCKS_PER_SEC);
        cout<< fixed
            << time_taken << setprecision(5);
        cout << "\n" ;
    }
```

Algorithm:

Merge sort:

To sort an array of size N in ascending order:

- Step 1 − if it is only one element in the list it is already sorted, return.
- Step 2 − divide the list recursively into two halves until it can no more be divided.
- Step 3 − merge the smaller lists into new list in sorted order.

Data:	Graphs: Count of swaps For last pivot and random pivot:  Time graph for last pivot and random pivot: 
Observation:	1. Fluctuations can be seen in the sorting algorithm performance due to Laptop overheating, Background apps and low battery.

	2. In general we can see that the performance of both algorithms is almost identical up to array size of 10000. 3. In quick sort array is not divided into equal parts while in merge sort,array is divided in half. 4. Merge sort is more efficient and works faster than quick sort in case of larger array size or datasets. whereas Quick sort is more efficient and works faster than merge sort in case of smaller array size or datasets.
Conclusion:	I understood in depth the Merge and Quick sorting algorithms and their relative performance.