Emilie Gunti
CSE 310 – Balasooriya

Recitation 2 Submission Template
CSE 310 – Summer 2023

a) Copy and paste your C++ program here and upload your C++ program to google drive and provide the link for the google drive so that we can download and run the program. Missing the Google link will result in (-10) points for the recitation.

```
/* name: Emilie Gunti

teacher: Janaka Balasooriya

recitation 2: run times of the sorting algorithms according to the textbook, the following

code sorts a random array according to the given size

and sorts each one, returns the running time of each sorted algorithm*/



#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;


void insertionSort(double array[], int n);

void selectionSort(double array[], int n);

void quickSort(double array[], int n);

void mergeSort(double array[], int n);

void randomGen(double array[], int n);


void insertionSort(double array[], int n){

    for (int i = 1; i < n; i++){

        double k = array[i];

        int j = i-1;

        while (array[j] > k && j>= 0){
            array[j+1] = array[j];
            j--;
        }

    array[j+1] = k ;
    }
}
//change
void selectionSort( double array[], int n){
    for ( int i = 0; i < n-1; i++){
        int indexMin = i;
        for (int j = i+1; j<n; j++){
            if(array[j] < array[indexMin]){
                indexMin = j;
            }
        }
        if (indexMin != i) {
            double val = array[i];
            array[i] = array[indexMin];
            array[indexMin] = val;
        }
    }

}
int partition( double array[], int low, int high){
    double s = array[high];
    int i = (low -1);
```

Emilie Gunti
CSE 310 – Balasooriya

```
    for(int j = low; j <= high - 1; j++){
        if ( array[j] < s) {
            i++;
            double var = array[i];
            array[i] = array[j];
            array[j] = var;
        }
    }

    double var = array[i +1];
    array[i +1] = array[high];
    array[high] = var;
    return i +1;

}



void quickSort( double array[], int low, int high){
    if (low < high){
        int p = partition (array, low, high);

        quickSort(array, low, p-1);
        quickSort(array, p+1, high);
    }

}
```

Emilie Gunti
CSE 310 – Balasooriya

```
void merge(double array[], int left, int mid, int right){
    int num1 = mid - left +1;
    int num2 = right - mid;
    double leftArray[num1], rightArray[num2];

    for (int i = 0; i < num1; i++){
        leftArray[i] = array[left + i];
    }
    for (int j = 0; j < num2; j++){
        leftArray[j] = array[mid + 1 + j];
    }

    int i = 0, j = 0, k = left;
    while (i < num1 && j < num2){
        if( leftArray[i] <= rightArray[j]){
            array[k] = leftArray[i];
            i++;
        }
        else {
            array[k] = rightArray[j];
            j++;
        }
        k++;
    }
    while (i < num1) {
        array[k] = leftArray[i];
        i++;
        k++;
    }

    while (j < num2) {
        array[k] = rightArray[j];
        j++;
        k++;
    }
}

void mergeSort( double array[], int left, int right){
    if (left < right){
        int mid = left + (right - left)/2;
        mergeSort(array, left, mid);
        mergeSort(array, mid +1, right);
        merge(array, left, mid, right);

    }

}

void randomGen( double array[], int n){
    for (int i = 0; i < n; i++){
        array[i] = 100.00 +(rand() % (1000-100 +1));
    }
}



int main() {
    const int arraySize = 10000;
    double array[arraySize];

    //generate random array
    randomGen(array, arraySize);

    //Insertion sort
    double insertionArray[arraySize];
    copy (array, array +arraySize, insertionArray);

    clock_t startInsertion = clock();
    insertionSort(insertionArray, arraySize);
    clock_t endInsertion = clock();

    double insertionTime = (double) (endInsertion - startInsertion)/ CLOCKS_PER_SEC;

    //Selection sort
    double selectionArray[arraySize];
    copy(array, array +arraySize, selectionArray);

    clock_t startSelection = clock();
    selectionSort(selectionArray, arraySize);
    clock_t endSelection = clock();

    double selectionTime = (double)(endSelection - startSelection) /
CLOCKS_PER_SEC;

    //Quick sort
    double quickArray[arraySize];
    copy(array, array + arraySize, quickArray);

    clock_t startQuick = clock();
    quickSort(quickArray, 0, arraySize -1);
    clock_t endQuick = clock();

    double quickTime = (double)(endQuick - startQuick) / CLOCKS_PER_SEC;

    //merge sort
    double mergeArray[arraySize];
     copy(array, array + arraySize, mergeArray);

    clock_t startMerge = clock();
    mergeSort(mergeArray, 0, arraySize -1);
    clock_t endMerge = clock();

    double mergeTime = (double)(endMerge - startMerge) / CLOCKS_PER_SEC;

    //output running times
    cout << "Insertion sort time: " << insertionTime * 1000 << " milliseconds" << endl;
```

Emilie Gunti
CSE 310 – Balasooriya

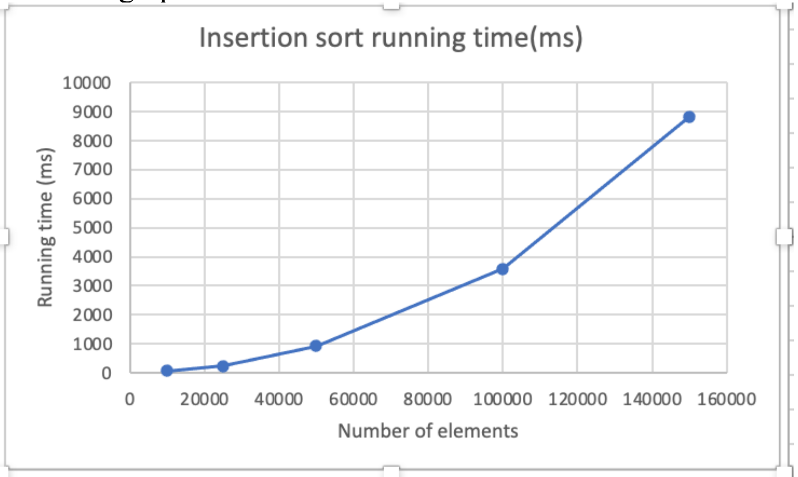
```
    cout << "Selection sort time: " << selectionTime * 1000 << " milliseconds" << endl;

    cout << "Quick sort time: " << quickTime * 1000 << " milliseconds" << endl;

    cout << "Merge sort time: " << mergeTime * 1000 << " milliseconds" << endl;


return 0;


}
```

Google drive link: https://drive.google.com/file/d/1NT7yIqa_ZcooMx-eDczyZsCt-gMHphVr/view?usp=sharing

b) Fill the following table with run times in milliseconds

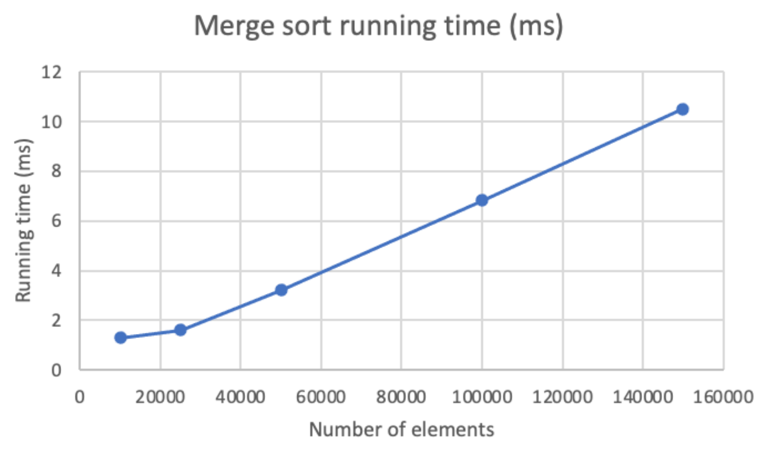
Array Size (n)	Insertion Sort	Merge Sort	Quick Sort	Selection Sort
10000	63.787 ms	1.301 ms	1.405 ms	109.81 ms
25000	242.244 ms	1.605 ms	2.216 ms	422.444 ms
50000	921.575 ms	3.215 ms	5.707 ms	1637.41 ms
100000	3578.39 ms	6.827 ms	14.962 ms	6557.59 ms
150000	8835.14 ms	10.509 ms	28.035 ms	14732.2 ms
200000	Out of bounds	Out of bounds	Out of bounds	Out of bounds

c) Copy and paste your running time graph here
Insertion graph:

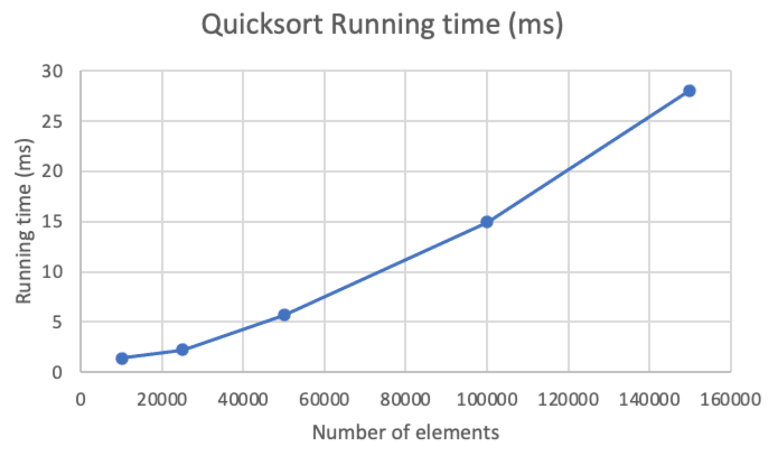


Emilie Gunti
CSE 310 – Balasooriya

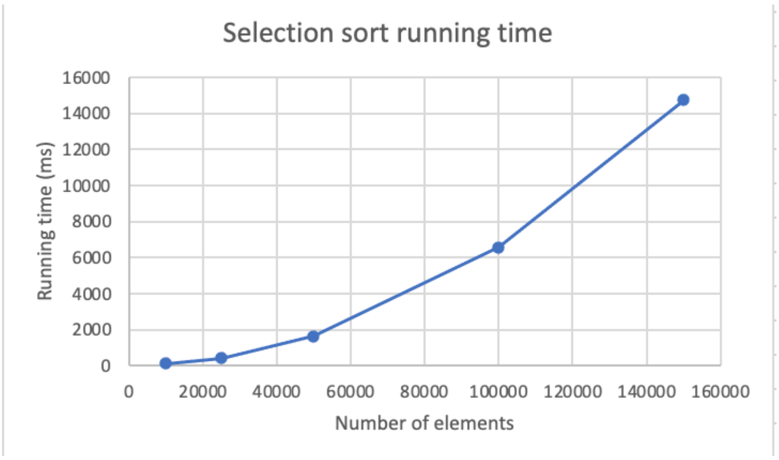
Merge graph:



Quicksort graph:



Selection sort graph:



d) Explain your comparison with the asymptotic analysis based running time of the above sorting algorithms.
   - The asymptotic analysis based on running time of insertion sort, merge sort, quick sort, and selection sort is listed respectively: $O(n^2)$, O(nlogn), O(nlogn), $O(n^2)$. This matches with the graphs of the sorting algorithms above. Firstly, insertion sort. The graph shows quadratic behavior of running time against input size. As the number of elements increase, the running time of insertion sort will also increase significantly, showing the steep growth from (c). Secondly, merge sort follows a DC approach, recursively dividing the input until each subarray has one element. The merging process takes O(n) time while the divide step takes O(log n) time, and putting them together it would be O(nlogn), and the given graph provides evidence that supports the asymptotic notation and shows the growth rate close to o(nlogn) as the input size increases (through its growth become more linear as the size grows. Thirdly, quicksort is also O(nlogn). In average case it exhibits o(nlogn) while in worst case it shows $O(n^2)$, meaning that as the input size grows, the running time is logarithmically increasing with respect to the number of elements. When it is $O(n^2)$, the chosen pivot index is either the largest or smallest element in the input array, in the end it takes more time dividing, causing a quadratic increase in running time. Although the graph could be representative of either, in these set of values, I believe it is O(nlogn), with its values also showing it to be more of the average case than the worst-case scenario. Lastly, selection sort is $O(n^2)$, because it repeatedly chooses the smallest element from the unsorted array and swaps it at the beginning, the increase of nested loops results in a quadratic time complexity. We see this quadratic behavior in the running time of the selection sort graph, proving it to be $O(n^2)$