CSCI-311 **Project 1**

1. Implement the following algorithms that will run on a vector of integers:

   Implement bubble sort.

   Implement insertion sort.

   Implement selection sort.

   Implement quicksort.

2. Run the following experiments:

We use five different sizes for our vectors of integers (10,000, 100,000, 500,000, 1mln and 2mln).
For each size, we create a single randomly generated vector of integers and
Run bubble sort, insertion, selection and quicksort on the same vector (we make a copy of the original vector for each algorithm).
We measure the run time in seconds and print the results for each algorithm.
Save those results, you'll need them to build graphs in Step 3.

```
/*
Declare your functions here and put code for functions' definitions below main()
function
*/
//function that creates a random vector called rVec of size "asize"
void randomVector(vector<int> &rVec, int asize){
    rVec.resize(asize, 0);
    for(int i = 0; i < size; i++){
           rVec[i] = rand() % (1000000);
    }
}
int main(){
    srand(time(0));
    int TOTAL = 5;
    int sizesOfVectors[TOTAL] = {10000, 100000, 500000, 1000000, 2000000};
    for(int i = 0; i < TOTAL; i++){
           curSize = sizeOfVectors[i];
           vector<int> origVector;
           randomVector(origVector, curSize);
           //origVector contains random integers
           //for each algorithm, make a separate copy of the original vector

           //bubble sort
           vector<int> bubbleVec = origVector;
           //run bubble sort and measure time of bubble sort:
           clock_t t1 = clock();
           //call bubble sort in the line below:
                              //here where you need to call bubble sort on bubbleVec
           clock_t t2 = clock();
           double elapsed = double(t2 - t1)/CLOCKS_PER_SEC;
           cout << "Bubble sort (size and run time in sec): " << curSize << " " <<
elapsed << endl;


           //insertion sort
           vector<int> insertionVec = origVector;
           //run insertion sort and measure the run time:
```

```
            t1 = clock();
            //call insertion sort in the line below:
                                  //here where you need to call insertion sort on
                                   //insertionVec vector
            t2 = clock();
            elapsed = double(t2 - t1)/CLOCKS_PER_SEC;
            cout << "Insertion sort (size and run time in sec): " << curSize << " " <<
elapsed << endl;


            //selection sort
            vector<int> selectionVec = origVector;
            //run selection sort and measure the run time:
            t1 = clock();
            //call selection sort in the line below:
                                       //here where you need to call selection sort on
                                        // selectionVec vector
            t2 = clock();
            elapsed = double(t2 - t1)/CLOCKS_PER_SEC;
            cout << "Selection sort (size and run time in sec): " << curSize << " " <<
elapsed << endl;


            //quickSort sort
            vector<int> quickSortVec = origVector;
            //run quickSort sort and measure the run time:
            t1 = clock();
            //call quickSort sort in the line below:
                                  //here where you need to call quickSort sort on
                                   //quickSortVec vector
            t2 = clock();
            elapsed = double(t2 - t1)/CLOCKS_PER_SEC;
            cout << "QuickSort sort (size and run time in sec): " << curSize << " " <<
elapsed << endl;
    }//for loop
     return 0;
}
```

3. Collect the data from the experiments in Step 2.
4. Write the following report (below).
   You need to fill in the given table(s) and
   Build the graphs of your experiments and
   Answer (type in) the questions in allocated space.
   Submit the Report only on Blackboard to Project 1.

Project 1 Report **Name: Jhet Cabigas**

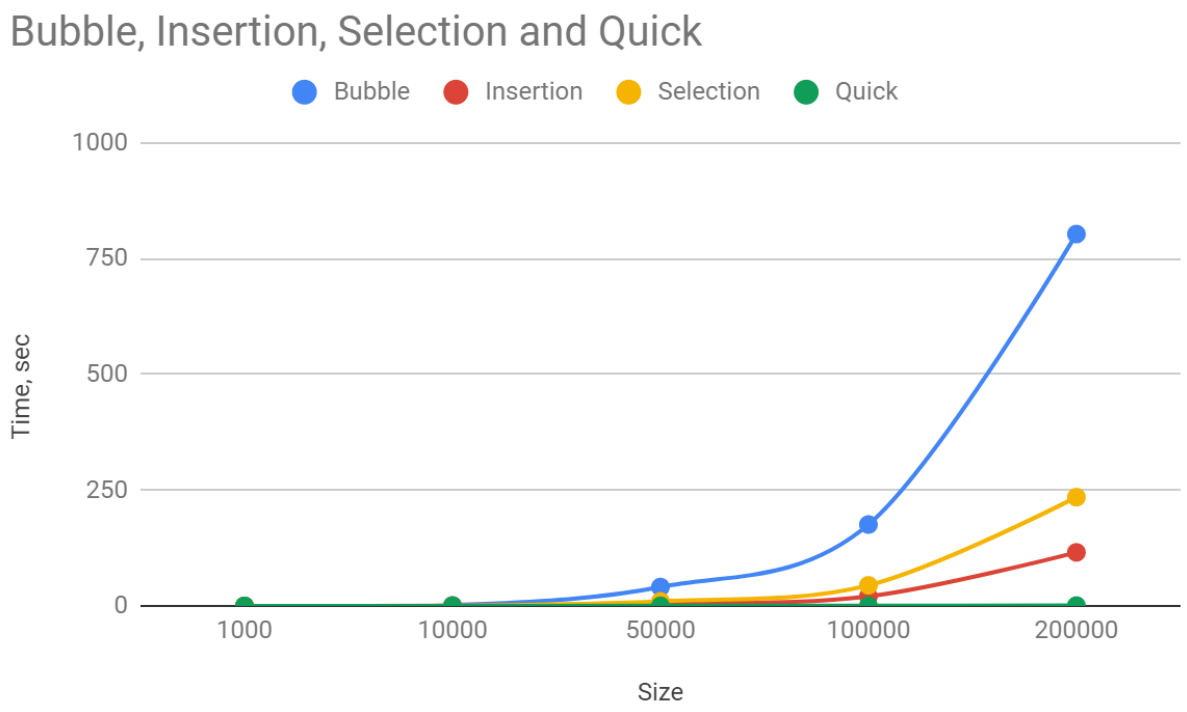
Data from the experiment of running the four algorithms on different data sets: you need to type in the results.

Algorithm:	Time, sec For size 1,000	Time, sec For size 10,000	Time, sec For size 50,000	Time, sec For size 100,000	Time, sec For size 200,000
Bubble	0.022955	1.34001	40.611	175.609	802.708
Insertion	0.002959	0.185649	5.30828	20.4594	115.524
Selection	0.006105	0.381827	9.87802	44.2801	234.709
QuickSort	0.00248	0.017082	0.094219	0.206347	0.641084

Build a graph with x-axis containing the sizes of the vectors and y-axis containing time in seconds:

- Add x-axis title "Size"
- Add y-axis title "Time, sec"
- Add legends identifying which algorithm corresponds to which line on the graph

Insert the graph (picture of the graph) below:



Conclusion: Fill in the following table with your conclusion about the experiment (type your answer):

Which algorithm was the fastest (with the smallest run time)?	Quicksort.
Which algorithm was the slowest (with the highest run time)?	Bubble Sort.
Which algorithms performed similarly?	Selection sort and Insertion sort had similar runtimes.
Were the run-time results of the algorithms expected for average case scenario? Briefly explain why.	The results were as expected. Bubble sort pulled away from the group substantially, while quicksort maintained its logarithmic complexity, being dwarfed by the other algorithms.