Gabriel Brehm

CS260

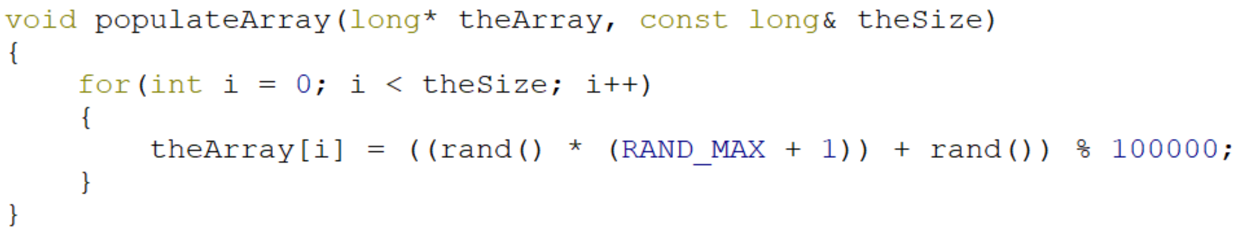
Dr. Breeann Flesch

22 April 2018

Lab 1 Report

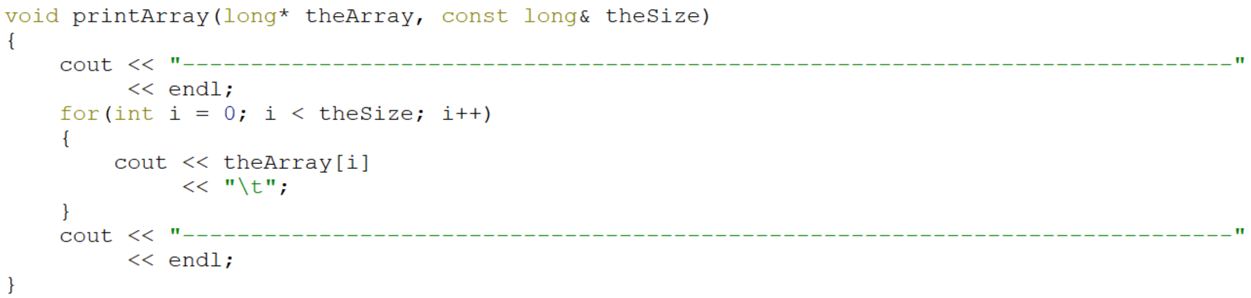
Part One:

For the first part of the lab, I wrote two separate functions; one that populates the array, and one that prints out the array. All the functions written in this project take in the array and the size of the array as parameters.



```
void populateArray(long* theArray, const long& theSize)
{
    for(int i = 0; i < theSize; i++)
    {
        theArray[i] = ((rand() * (RAND_MAX + 1)) + rand()) % 100000;
    }
}
```

The first function goes through a for loop setting the element at each index to a random number between 0 and 99999.



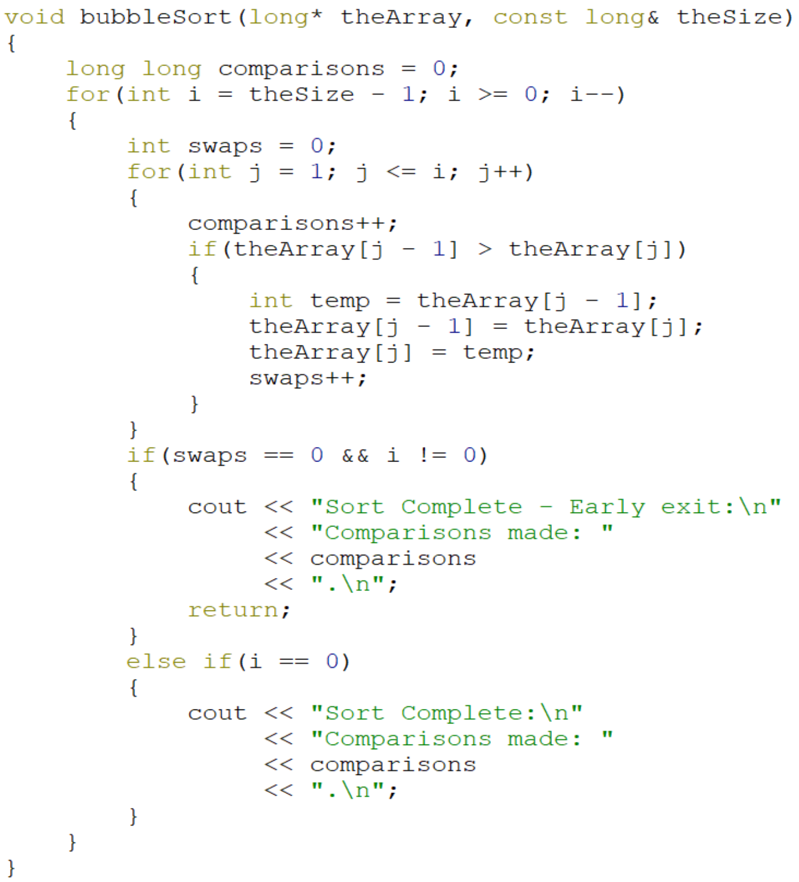
```
void printArray(long* theArray, const long& theSize)
{
    cout << "-----------------------------------------------------------------------------"
         << endl;
    for(int i = 0; i < theSize; i++)
    {
        cout << theArray[i]
             << "\t";
    }
    cout << "-----------------------------------------------------------------------------"
         << endl;
}
```

The second function goes through a for loop printing out each element in the array, along with some formatting for easier readability.

Parts Two & Three:

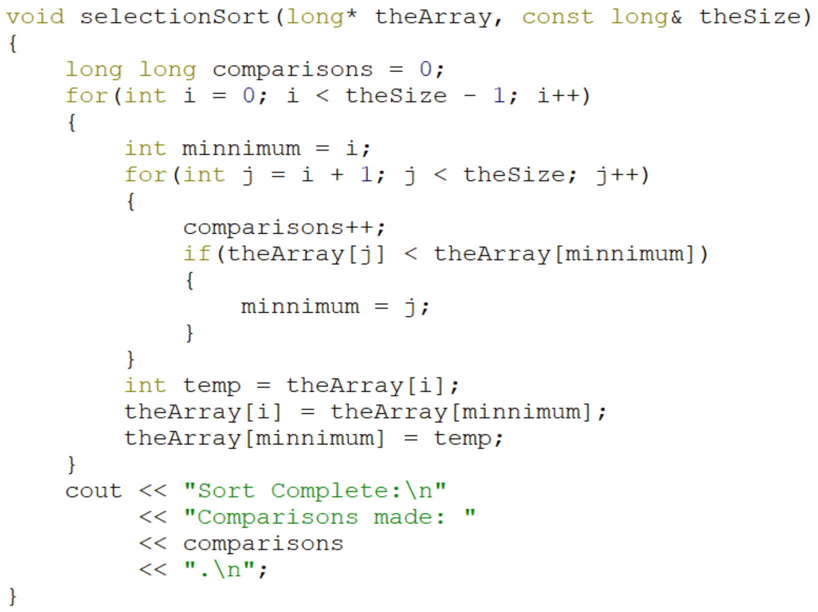
For this part of the lab, I wrote three more functions, one for each of the sorting methods. All three of these functions also keep track of how many comparisons are made throughout the process and print them out with some formatting at the end.

The bubble sort function sorts the elements by swapping smaller elements with larger ones, so the larger elements is closer to the end of the array. There is a built-in test to see if the array is sorted before it reaches the end, resulting in an early exit.



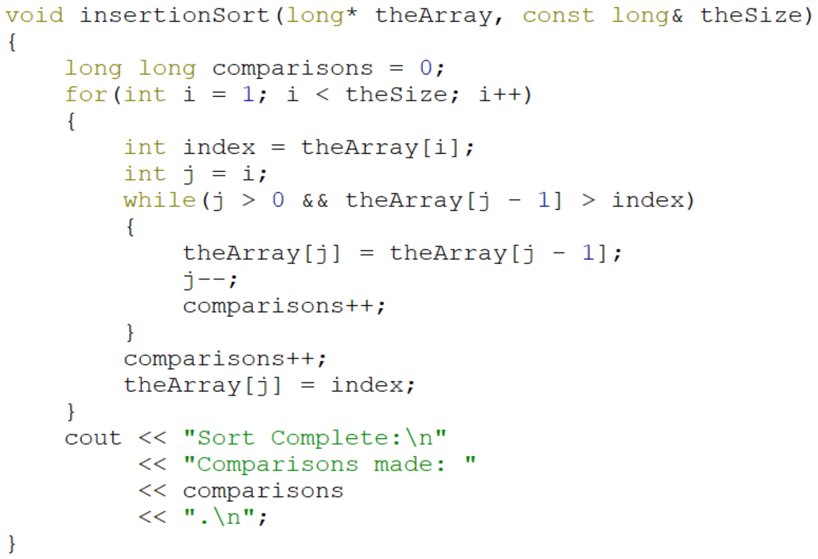
```
void bubbleSort(long* theArray, const long& theSize)
{
    long long comparisons = 0;
    for(int i = theSize - 1; i >= 0; i--)
    {
        int swaps = 0;
        for(int j = 1; j <= i; j++)
        {
            comparisons++;
            if(theArray[j - 1] > theArray[j])
            {
                int temp = theArray[j - 1];
                theArray[j - 1] = theArray[j];
                theArray[j] = temp;
                swaps++;
            }
        }
        if(swaps == 0 && i != 0)
        {
            cout << "Sort Complete - Early exit:\n"
                 << "Comparisons made: "
                 << comparisons
                 << ".\n";
            return;
        }
        else if(i == 0)
        {
            cout << "Sort Complete:\n"
                 << "Comparisons made: "
                 << comparisons
                 << ".\n";
        }
    }
}
```

The selection sort function assumes the first element is the smallest, and then looks through the array until it finds the next smallest, and it swaps them. There is no early exit option for this function.



```
void selectionSort(long* theArray, const long& theSize)
{
    long long comparisons = 0;
    for(int i = 0; i < theSize - 1; i++)
    {
        int minnimum = i;
        for(int j = i + 1; j < theSize; j++)
        {
            comparisons++;
            if(theArray[j] < theArray[minnimum])
            {
                minnimum = j;
            }
        }
        int temp = theArray[i];
        theArray[i] = theArray[minnimum];
        theArray[minnimum] = temp;
    }
    cout << "Sort Complete:\n"
         << "Comparisons made: "
         << comparisons
         << ".\n";
}
```

The insertion sort function assumes the first element of the array to be sorted, and then proceeds to insert the unsorted section into the sorted section. Like the selection sort function, this one has no early exit.

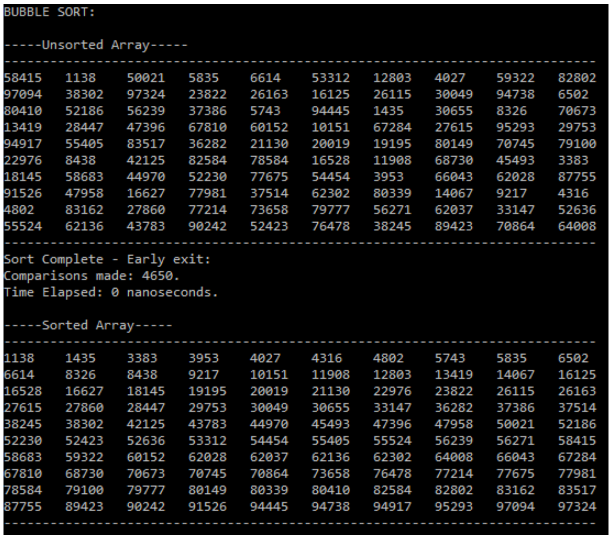


```
void insertionSort(long* theArray, const long& theSize)
{
    long long comparisons = 0;
    for(int i = 1; i < theSize; i++)
    {
        int index = theArray[i];
        int j = i;
        while(j > 0 && theArray[j - 1] > index)
        {
            theArray[j] = theArray[j - 1];
            j--;
            comparisons++;
        }
        comparisons++;
        theArray[j] = index;
    }
    cout << "Sort Complete:\n"
         << "Comparisons made: "
         << comparisons
         << ".\n";
}
```

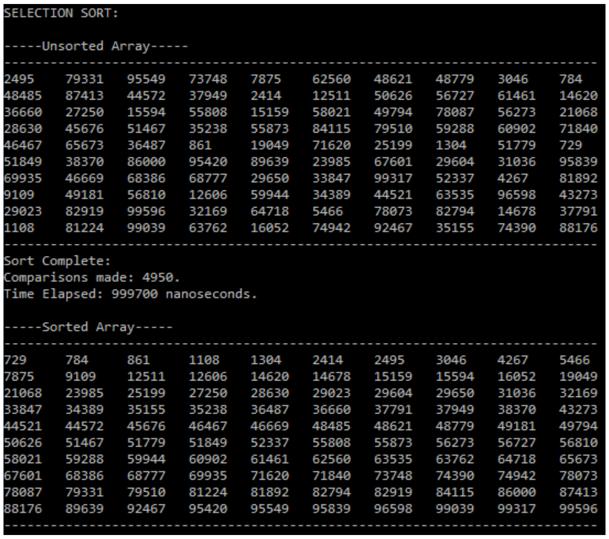
I then wrote three more function that run and time the above functions. They each populate the array, or repopulate it if it was already populated, start a timer, run their respective sort functions (bubble, selection, insertion), stop the timer, and then print out the elapsed time. Since each sort itself already pints out its own comparisons, the only thing these functions need to print out is their time.

```
void printBubbleSort(long *theArray, const long &theSize)
{
    cout << "BUBBLE SORT:\n";
    populateArray(theArray, theSize);
    auto startTime = chrono::high_resolution_clock::now();
    bubbleSort(theArray, theSize);
    auto endTime = chrono::high_resolution_clock::now();
    cout << "Time Elapsed: "
         << chrono::duration_cast<std::chrono::nanoseconds>(endTime-startTime).count()
         << " nanoseconds.\n";
}

void printSelectionSort(long *theArray, const long &theSize)
{
    cout << "\nSELECTION SORT:\n";
    populateArray(theArray, theSize);
    auto startTime = chrono::high_resolution_clock::now();
    selectionSort(theArray, theSize);
    auto endTime = chrono::high_resolution_clock::now();

    cout << "Time Elapsed: "
         << chrono::duration_cast<std::chrono::nanoseconds>(endTime-startTime).count()
         << " nanoseconds.\n";
}

void printInsertionSort(long* theArray, const long& theSize)
{
    cout << "\nINSERTION SORT:\n";
    populateArray(theArray, theSize);
    auto startTime = chrono::high_resolution_clock::now();
    insertionSort(theArray, theSize);
    auto endTime = chrono::high_resolution_clock::now();
    cout << "Time Elapsed: "
         << chrono::duration_cast<std::chrono::nanoseconds>(endTime-startTime).count()
         << " nanoseconds.\n";
}
```

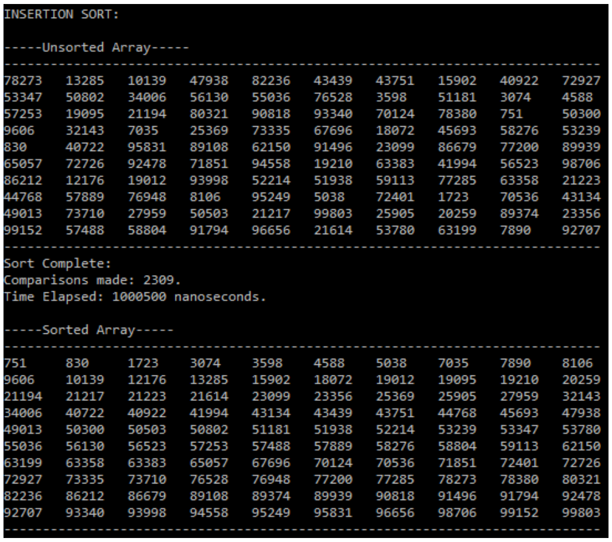
The following images label the type of sort, show an unsorted array, show how many comparisons are made, how long it takes, and then finally prints out the sorted array. There is one image for each type of sort.



```
BUBBLE SORT:

-----Unsorted Array-----
-------------------------------------------------------------------------------
58415   1138    50021   5835    6614    53312   12803   4027    59322   82802
97094   38302   97324   23822   26163   16125   26115   30049   94738   6502
80410   52186   56239   37386   5743    94445   1435    30655   8326    70673
13419   28447   47396   67810   60152   10151   67284   27615   95293   29753
94917   55405   83517   36282   21130   20019   19195   80149   70745   79100
22976   8438    42125   82584   78584   16528   11908   68730   45493   3383
18145   58683   44970   52230   77675   54454   3953    66043   62028   87755
91526   47958   16627   77981   37514   62302   80339   14067   9217    4316
4802    83162   27860   77214   73658   79777   56271   62037   33147   52636
55524   62136   43783   90242   52423   76478   38245   89423   70864   64008
-------------------------------------------------------------------------------
Sort Complete - Early exit:
Comparisons made: 4650.
Time Elapsed: 0 nanoseconds.

-----Sorted Array-----
-------------------------------------------------------------------------------
1138    1435    3383    3953    4027    4316    4802    5743    5835    6502
6614    8326    8438    9217    10151   11908   12803   13419   14067   16125
16528   16627   18145   19195   20019   21130   22976   23822   26115   26163
27615   27860   28447   29753   30049   30655   33147   36282   37386   37514
38245   38302   42125   43783   44970   45493   47396   47958   50021   52186
52230   52423   52636   53312   54454   55405   55524   56239   56271   58415
58683   59322   60152   62028   62037   62136   62302   64008   66043   67284
67810   68730   70673   70745   70864   73658   76478   77214   77675   77981
78584   79100   79777   80149   80339   80410   82584   82802   83162   83517
87755   89423   90242   91526   94445   94738   94917   95293   97094   97324
-------------------------------------------------------------------------------
```



```
SELECTION SORT:

-----Unsorted Array-----
-------------------------------------------------------------------------------
2495    79331   95549   73748   7875    62560   48621   48779   3046    784
48485   87413   44572   37949   2414    12511   50626   56727   61461   14620
36660   27250   15594   55808   15159   58021   49794   78087   56273   21068
28630   45676   51467   35238   55873   84115   79510   59288   60902   71840
46467   65673   36487   861     19049   71620   25199   1304    51779   729
51849   38370   86000   95420   89639   23985   67601   29604   31036   95839
69935   46669   68386   68777   29650   33847   99317   52337   4267    81892
9109    49181   56810   12606   59944   34389   44521   63535   96598   43273
29023   82919   99596   32169   64718   5466    78073   82794   14678   37791
1108    81224   99039   63762   16052   74942   92467   35155   74390   88176
-------------------------------------------------------------------------------
Sort Complete:
Comparisons made: 4950.
Time Elapsed: 999700 nanoseconds.

-----Sorted Array-----
-------------------------------------------------------------------------------
729     784     861     1108    1304    2414    2495    3046    4267    5466
7875    9109    12511   12606   14620   14678   15159   15594   16052   19049
21068   23985   25199   27250   28630   29023   29604   29650   31036   32169
33847   34389   35155   35238   36487   36660   37791   37949   38370   43273
44521   44572   45676   46467   46669   48485   48621   48779   49181   49794
50626   51467   51779   51849   52337   55808   55873   56273   56727   56810
58021   59288   59944   60902   61461   62560   63535   63762   64718   65673
67601   68386   68777   69935   71620   71840   73748   74390   74942   78073
78087   79331   79510   81224   81892   82794   82919   84115   86000   87413
88176   89639   92467   95420   95549   95839   96598   99039   99317   99596
-------------------------------------------------------------------------------
```



```
INSERTION SORT:

-----Unsorted Array-----
--------------------------------------------------------------------------------
78273   13285   10139   47938   82236   43439   43751   15902   40922   72927
53347   50802   34006   56130   55036   76528   3598    51181   3074    4588
57253   19095   21194   80321   90818   93340   70124   78380   751     50300
9606    32143   7035    25369   73335   67696   18072   45693   58276   53239
830     40722   95831   89108   62150   91496   23099   86679   77200   89939
65057   72726   92478   71851   94558   19210   63383   41994   56523   98706
86212   12176   19012   93998   52214   51938   59113   77285   63358   21223
44768   57889   76948   8106    95249   5038    72401   1723    70536   43134
49013   73710   27959   50503   21217   99803   25905   20259   89374   23356
99152   57488   58804   91794   96656   21614   53780   63199   7890    92707
--------------------------------------------------------------------------------
Sort Complete:
Comparisons made: 2309.
Time Elapsed: 1000500 nanoseconds.

-----Sorted Array-----
--------------------------------------------------------------------------------
751     830     1723    3074    3598    4588    5038    7035    7890    8106
9606    10139   12176   13285   15902   18072   19012   19095   19210   20259
21194   21217   21223   21614   23099   23356   25369   25905   27959   32143
34006   40722   40922   41994   43134   43439   43751   44768   45693   47938
49013   50300   50503   50802   51181   51938   52214   53239   53347   53780
55036   56130   56523   57253   57488   57889   58276   58804   59113   62150
63199   63358   63383   65057   67696   70124   70536   71851   72401   72726
72927   73335   73710   76528   76948   77200   77285   78273   78380   80321
82236   86212   86679   89108   89374   89939   90818   91496   91794   92478
92707   93340   93998   94558   95249   95831   96656   98706   99152   99803
--------------------------------------------------------------------------------
```

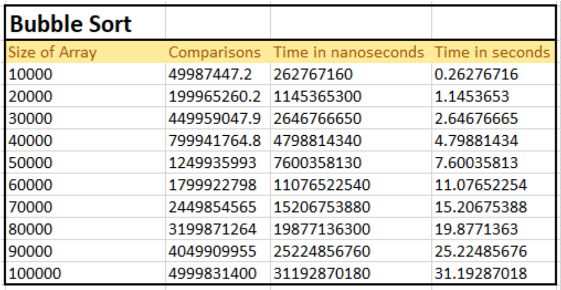
As one might point out, the bubble sort reported a time of zero nanoseconds. This is obviously not the case, because it cannot be an instantaneous action. The reason for this measurement is that the processor has difficulty recording times under 100000 nanoseconds, so time measurements for such small arrays are not very reliable. We will see more of this phenomenon later.

Part Four:

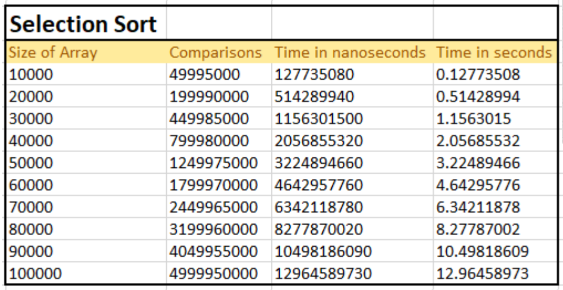
See appendix for the raw data charts. The array was created on the heap, so the size could be a variable (a specific requirement for C++). This means that during data collection, the program was executed ten times at size 10000, then the size was increased to 20000, and the program was executed another ten time, et cetera. So, only one variable needed to be changed to test the comparisons, and times, for each size array.

Part Five:

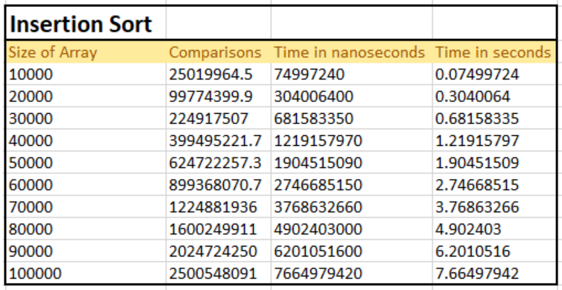
The data collected from part four resulted in the tables and graphs below:



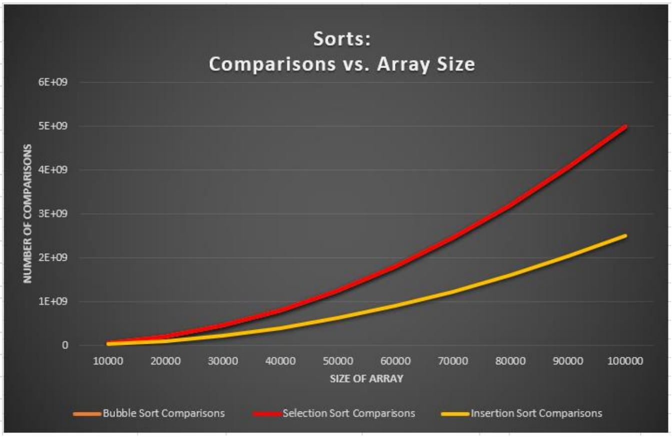
Bubble Sort			
Size of Array	Comparisons	Time in nanoseconds	Time in seconds
10000	49987447.2	262767160	0.26276716
20000	199965260.2	1145365300	1.1453653
30000	449959047.9	2646766650	2.64676665
40000	799941764.8	4798814340	4.79881434
50000	1249935993	7600358130	7.60035813
60000	1799922798	11076522540	11.07652254
70000	2449854565	15206753880	15.20675388
80000	3199871264	19877136300	19.8771363
90000	4049909955	25224856760	25.22485676
100000	4999831400	31192870180	31.19287018

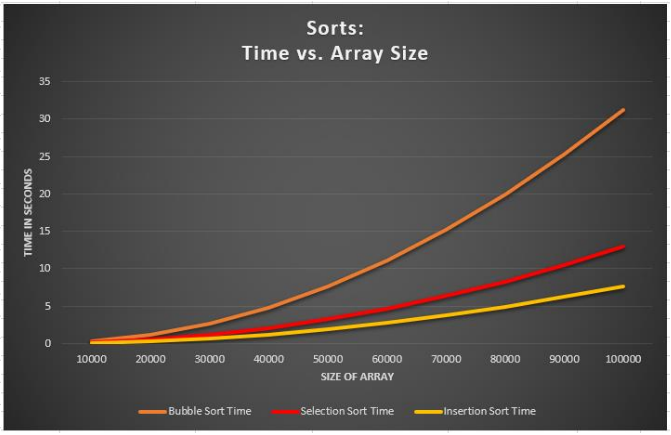


Selection Sort			
Size of Array	Comparisons	Time in nanoseconds	Time in seconds
10000	49995000	127735080	0.12773508
20000	199990000	514289940	0.51428994
30000	449985000	1156301500	1.1563015
40000	799980000	2056855320	2.05685532
50000	1249975000	3224894660	3.22489466
60000	1799970000	4642957760	4.64295776
70000	2449965000	6342118780	6.34211878
80000	3199960000	8277870020	8.27787002
90000	4049955000	10498186090	10.49818609
100000	4999950000	12964589730	12.96458973



Insertion Sort			
Size of Array	Comparisons	Time in nanoseconds	Time in seconds
10000	25019964.5	74997240	0.07499724
20000	99774399.9	304006400	0.3040064
30000	224917507	681583350	0.68158335
40000	399495221.7	1219157970	1.21915797
50000	624722257.3	1904515090	1.90451509
60000	899368070.7	2746685150	2.74668515
70000	1224881936	3768632660	3.76863266
80000	1600249911	4902403000	4.902403
90000	2024724250	6201051600	6.2010516
100000	2500548091	7664979420	7.66497942

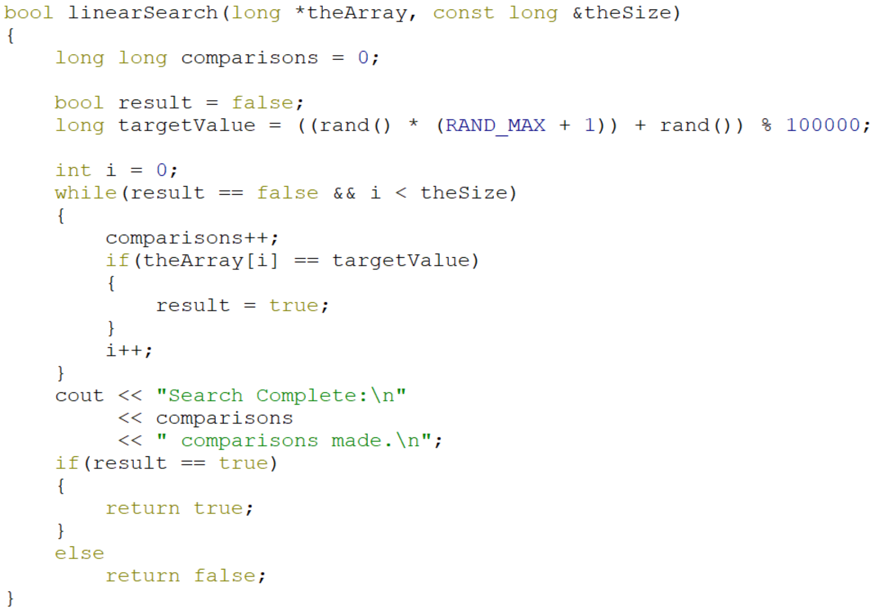




The graphs are directly based on the data recorded in the tables, with each sort corresponding to a different colored line; orange is bubble sort, red is selection sort, and yellow is insertion sort. On the comparisons vs. array size graph, there does not appear to be a line representing bubble sort, and that is because the number of comparisons for bubble sort and selection sort are almost identical at this scale. So, the bubble sort line is simply hiding behind the selection sort line. We can see from the graphs that for both comparisons made and time taken, insertion sort seems to be the most efficient. We can also see that although the comparisons made for bubble sort and selection sort are close, selection sort is far superior when it comes to time.

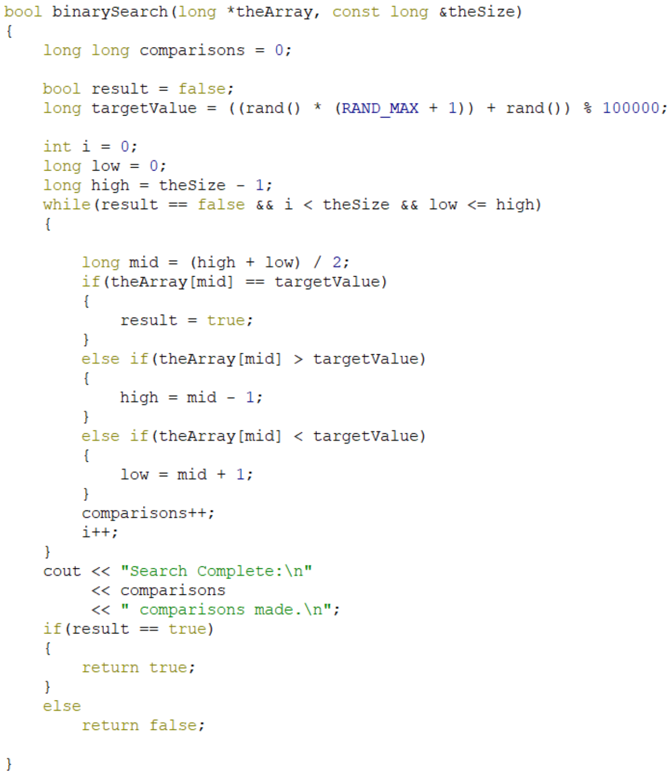
Parts Six & Seven:

Here I wrote two new functions; one for linear search, and one for binary search.



```
bool linearSearch(long *theArray, const long &theSize)
{
    long long comparisons = 0;

    bool result = false;
    long targetValue = ((rand() * (RAND_MAX + 1)) + rand()) % 100000;

    int i = 0;
    while(result == false && i < theSize)
    {
        comparisons++;
        if(theArray[i] == targetValue)
        {
            result = true;
        }
        i++;
    }
    cout << "Search Complete:\n"
         << comparisons
         << " comparisons made.\n";
    if(result == true)
    {
        return true;
    }
    else
        return false;
}
```

The linear search function generates a random target value between 0 and 99999 and walks through the array checking every single element to see if it is equal to the target value. If it finds the target value, it kicks out of the while loop and returns true, otherwise it continues through the whole array, then returns false. It also keeps track of its own comparisons and prints those out.



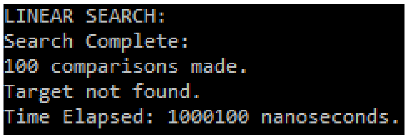
```
bool binarySearch(long *theArray, const long &theSize)
{
    long long comparisons = 0;

    bool result = false;
    long targetValue = ((rand() * (RAND_MAX + 1)) + rand()) % 100000;

    int i = 0;
    long low = 0;
    long high = theSize - 1;
    while(result == false && i < theSize && low <= high)
    {

        long mid = (high + low) / 2;
        if(theArray[mid] == targetValue)
        {
            result = true;
        }
        else if(theArray[mid] > targetValue)
        {
            high = mid - 1;
        }
        else if(theArray[mid] < targetValue)
        {
            low = mid + 1;
        }
        comparisons++;
        i++;
    }
    cout << "Search Complete:\n"
         << comparisons
         << " comparisons made.\n";
    if(result == true)
    {
        return true;
    }
    else
        return false;

}
```

The binary search function also generates a random target value between 0 and 99999, then it starts at the middle element and compares it to the target value. If the element is the same as the target value, then the search is complete, and the number of comparisons made is printed out. If the element is less than the target value, then the function looks at the second half of the array, and if the element is greater than the target value, then the function looks at the first half of the array. It should be noted that binary search only works if the array is sorted previously.

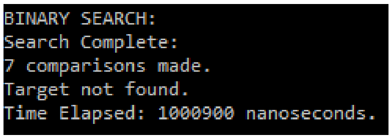
Much like for the sorts, for the searches I wrote three more functions that time each of the two searches. The first one populates and array, starts the timer, runs the linear search, and then stops the timer, and prints out how much time elapsed. The second one does all of the same things, except with binary search, and with the exception that it does not populate the array. That is because unlike linear search, binary search needs the array to be sorted before it can search.

```
void printLinearSearch(long *theArray, const long &theSize)
{
    cout << "\nLINEAR SEARCH:\n";
    auto startTime = chrono::high_resolution_clock::now();
    bool resultL = linearSearch(theArray, theSize);
    auto endTime = chrono::high_resolution_clock::now();
    if(resultL == true)
        cout << "Target found.\n";
    else
        cout << "Target not found.\n";
    cout << "Time Elapsed: "
         << chrono::duration_cast<std::chrono::nanoseconds>(endTime-startTime).count()
         << " nanoseconds.\n\n";
}

void printBinarySearch(long *theArray, const long &theSize)
{
    cout << "BINARY SEARCH:\n";
    auto startTime = chrono::high_resolution_clock::now();
    bool resultB = binarySearch(theArray, theSize);
    auto endTime = chrono::high_resolution_clock::now();
    if(resultB == true)
        cout << "Target found.\n";
    else
        cout << "Target not found.\n";
    cout << "Time Elapsed: "
         << chrono::duration_cast<std::chrono::nanoseconds>(endTime-startTime).count()
         << " nanoseconds.\n\n";
}
```

The following images label the type of search, show how many comparisons are made, and how long it takes. There is one image for each type of search.



```
LINEAR SEARCH:
Search Complete:
100 comparisons made.
Target not found.
Time Elapsed: 1000100 nanoseconds.
```



```
BINARY SEARCH:
Search Complete:
7 comparisons made.
Target not found.
Time Elapsed: 1000900 nanoseconds.
```

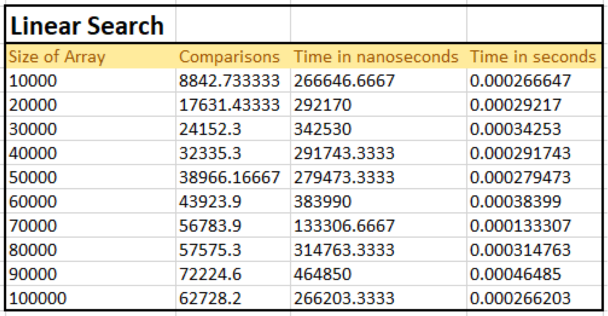
Note that both searches display a time, but that is not always the case. As stated earlier, the processor cannot effectively measure a time below 100000 nanoseconds. It is also good to point out that binary search uses a small number of comparisons, compared to the linear search.

Part Eight:

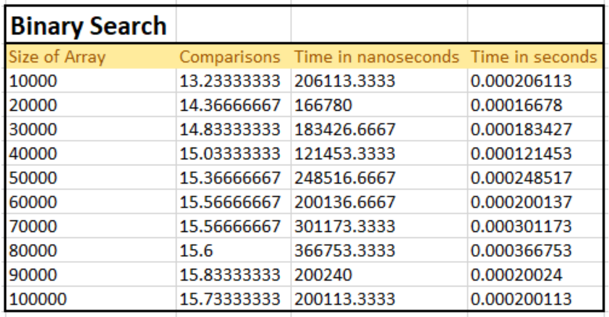
See appendix for the raw data charts. The array was created on the heap, so the size could be a variable (a specific requirement for C++). This means that during data collection, the program was executed thirty times at size 10000, then the size was increased to 20000, and the program was executed another thirty time, et cetera. So, only one variable needed to be changed to test the comparisons, and times, for each size array.

Part Nine:

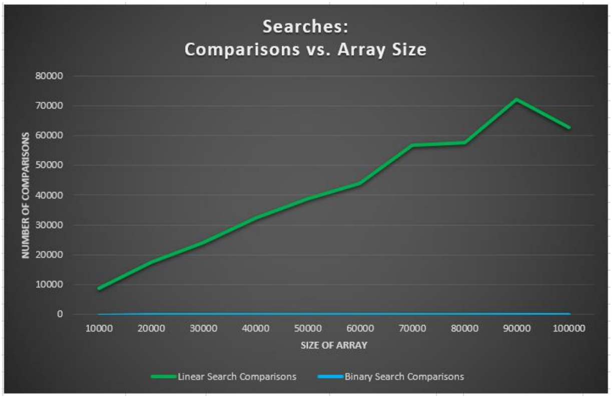
The data collected in part eight can be seen below:

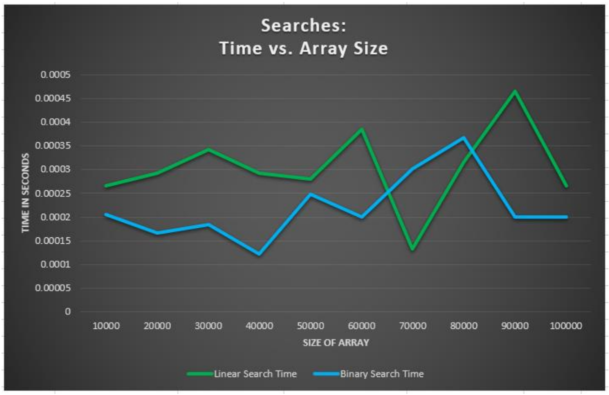


Linear Search			
Size of Array	Comparisons	Time in nanoseconds	Time in seconds
10000	8842.733333	266646.6667	0.000266647
20000	17631.43333	292170	0.00029217
30000	24152.3	342530	0.00034253
40000	32335.3	291743.3333	0.000291743
50000	38966.16667	279473.3333	0.000279473
60000	43923.9	383990	0.00038399
70000	56783.9	133306.6667	0.000133307
80000	57575.3	314763.3333	0.000314763
90000	72224.6	464850	0.00046485
100000	62728.2	266203.3333	0.000266203



Binary Search			
Size of Array	Comparisons	Time in nanoseconds	Time in seconds
10000	13.23333333	206113.3333	0.000206113
20000	14.36666667	166780	0.00016678
30000	14.83333333	183426.6667	0.000183427
40000	15.03333333	121453.3333	0.000121453
50000	15.36666667	248516.6667	0.000248517
60000	15.56666667	200136.6667	0.000200137
70000	15.56666667	301173.3333	0.000301173
80000	15.6	366753.3333	0.000366753
90000	15.83333333	200240	0.00020024
100000	15.73333333	200113.3333	0.000200113





Just like the tables and graphs for the sorts, the search graphs are based directly on the tables. In this case, we have linear search represented by the green lines, and binary search represented by the blue lines. As far as comparisons go, binary search blows linear search way out of the water, as is evident by the comparisons vs. array size graph. The time is another story however. As stated earlier, the processor cannot measure extremely small units of time, so the accuracy on the searches is, well, not accurate. The sizes of the arrays we are using are simply too small to effectively measure. The time vs. array size graph does show one important thing though; searches are quick. The longest time measured was only about 0.00046 seconds.

References

Peers:

I collaborated on this project with Megan Traeger, Jacob Malmstadt, Mitchel Walker, and Jacob McLoud.

Websites:

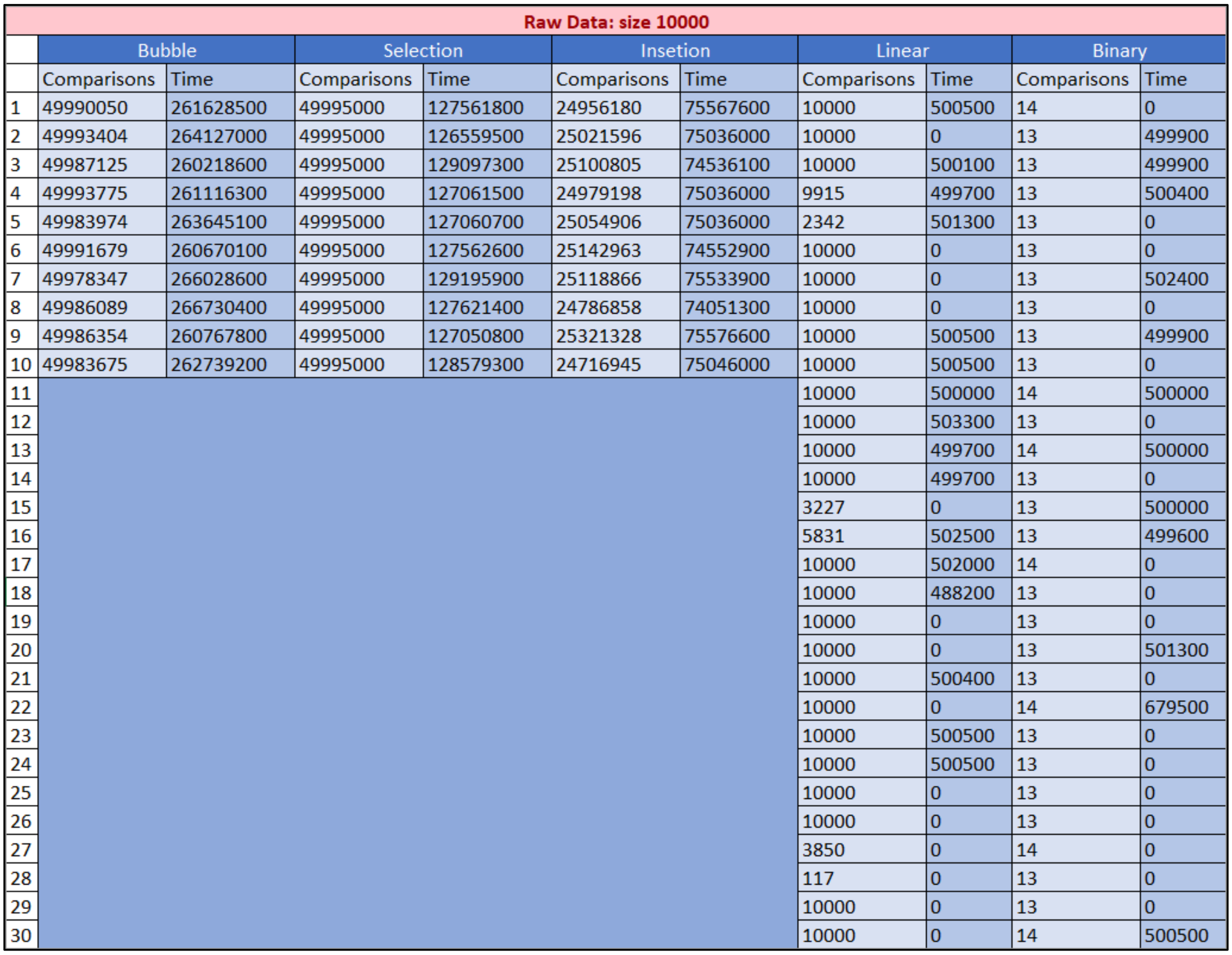
https://www.cs.cmu.edu/~adamchik/15-121/lectures/Sorting%20Algorithms/sorting.html

http://www.cplusplus.com/reference/ctime/

http://en.cppreference.com/w/cpp/chrono

https://stackoverflow.com/questions/36042637/how-to-calculate-execution-time-in-milliseconds

Appendix



Raw Data: size 10000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	49990050	261628500	49995000	127561800	24956180	75567600	10000	500500	14	0
2	49993404	264127000	49995000	126559500	25021596	75036000	10000	0	13	499900
3	49987125	260218600	49995000	129097300	25100805	74536100	10000	500100	13	499900
4	49993775	261116300	49995000	127061500	24979198	75036000	9915	499700	13	500400
5	49983974	263645100	49995000	127060700	25054906	75036000	2342	501300	13	0
6	49991679	260670100	49995000	127562600	25142963	74552900	10000	0	13	0
7	49978347	266028600	49995000	129195900	25118866	75533900	10000	0	13	502400
8	49986089	266730400	49995000	127621400	24786858	74051300	10000	0	13	0
9	49986354	260767800	49995000	127050800	25321328	75576600	10000	500500	13	499900
10	49983675	262739200	49995000	128579300	24716945	75046000	10000	500500	13	0
11							10000	500000	14	500000
12							10000	503300	13	0
13							10000	499700	14	500000
14							10000	499700	13	0
15							3227	0	13	500000
16							5831	502500	13	499600
17							10000	502000	14	0
18							10000	488200	13	0
19							10000	0	13	0
20							10000	0	13	501300
21							10000	500400	13	0
22							10000	0	14	679500
23							10000	500500	13	0
24							10000	500500	13	0
25							10000	0	13	0
26							10000	0	13	0
27							3850	0	14	0
28							117	0	13	0
29							10000	0	13	0
30							10000	0	14	500500

Raw Data: Size 20000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	199951497	1154110800	199990000	515469800	99422825	304215900	20000	500400	15	499700
2	199964349	1136923700	199990000	516655500	99846698	304991000	20000	500900	14	500100
3	199979560	1136553800	199990000	507855000	99953479	299238400	20000	0	14	0
4	199989724	1147454200	199990000	520990400	100161720	311717900	3851	500100	14	0
5	199963435	1148080300	199990000	521109500	99673867	310761600	7119	736500	14	0
6	199960110	1136707400	199990000	527433500	99370966	298197300	20000	0	14	0
7	199980684	1148356100	199990000	511421400	99844343	303044200	20000	501300	15	0
8	199956330	1150555500	199990000	509858900	99750361	301737500	20000	500800	15	0
9	199919124	1154032800	199990000	506005900	100172851	302903800	20000	500000	14	0
10	199987789	1140878400	199990000	506099500	99546889	303256400	20000	0	14	0
11							20000	500400	15	0
12							20000	0	14	0
13							20000	0	14	500100
14							20000	500400	14	0
15							20000	0	12	500400
16							20000	0	14	0
17							20000	503400	15	499200
18							15124	512000	14	502100
19							20000	502500	15	0
20							20000	500100	15	500400
21							20000	502500	15	499200
22							20000	0	15	0
23							2482	0	15	0
24							20000	499600	15	0
25							20000	500500	14	0
26							20000	503700	14	0
27							20000	0	14	0
28							13026	0	14	502100
29							20000	0	15	500100
30							7341	0	15	0

Raw Data: Size 30000

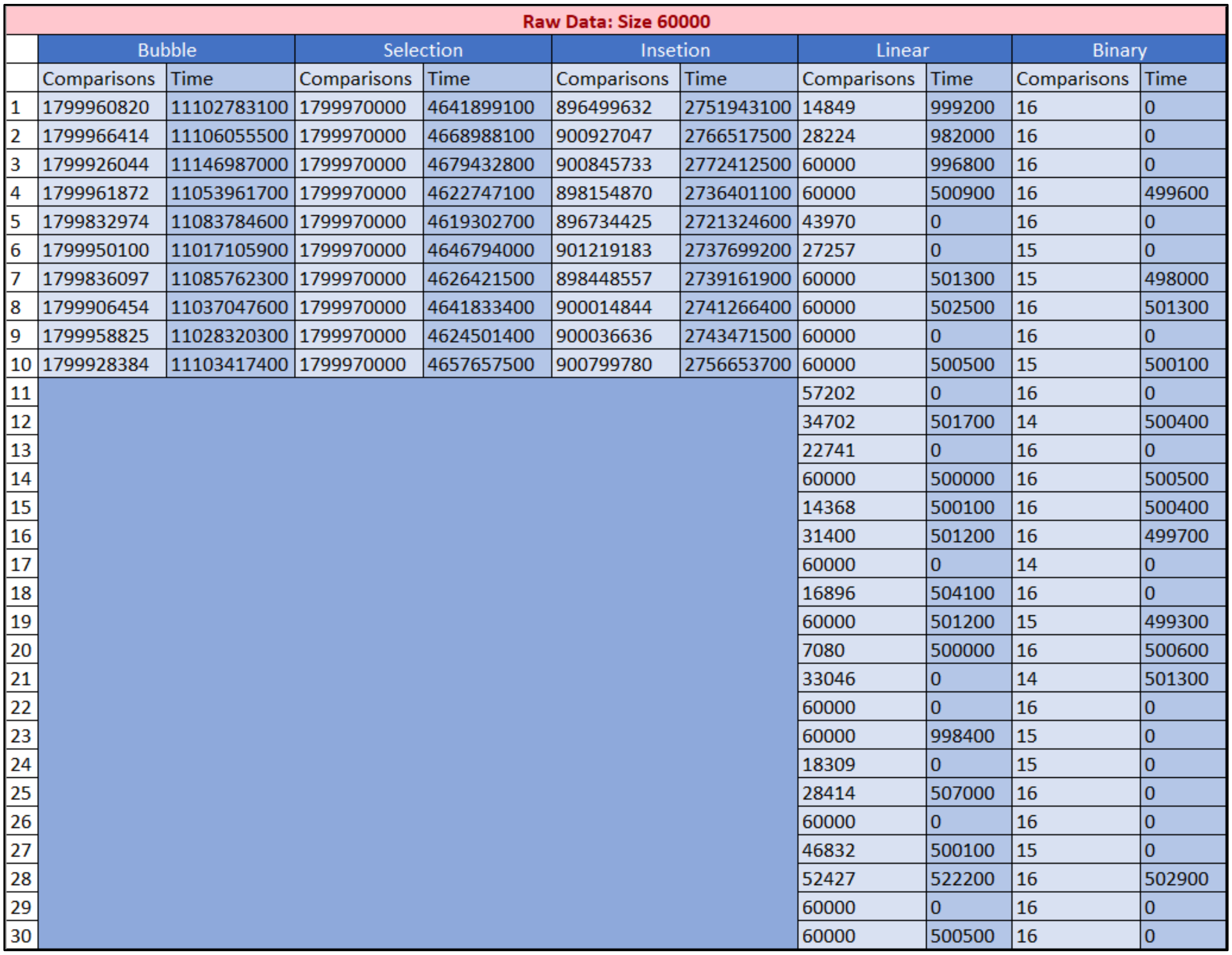
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	449919659	2644282800	449985000	1156015800	226103709	690541800	30000	511100	15	0
2	449948954	2628897500	449985000	1152024800	225072098	679581100	30000	0	14	0
3	449920020	2676265500	449985000	1146541100	225326160	676500300	30000	767300	15	501700
4	449976354	2659511200	449985000	1149736800	225665912	680998700	30000	500000	15	0
5	449971797	2629316300	449985000	1168239300	224512049	686661700	20387	0	15	0
6	449978097	2649298400	449985000	1149083200	223156893	675098000	30000	0	15	0
7	449977125	2646534600	449985000	1149297200	225609495	682565300	6050	502500	13	509100
8	449948954	2644253600	449985000	1172789800	224552904	675093800	1063	500400	15	0
9	449980629	2643540500	449985000	1166221500	224983701	678214000	30000	0	15	0
10	449968890	2645766100	449985000	1153065500	224192149	690578800	30000	4004500	15	0
11							30000	0	15	0
12							30000	0	15	0
13							30000	0	15	500400
14							30000	499300	15	501200
15							30000	0	15	0
16							21179	503300	15	499300
17							30000	0	15	500400
18							28889	0	15	0
19							17142	499200	15	500000
20							20341	0	14	489800
21							30000	500100	15	500400
22							30000	500900	15	0
23							30000	487300	15	0
24							2143	0	14	0
25							30000	0	15	500100
26							30000	0	15	500400
27							30000	500000	15	0
28							2139	0	15	0
29							5236	0	15	0
30							30000	0	15	0

Raw Data: Size 40000

	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	799958885	4779900900	799980000	2061541700	399590838	1223591600	40000	513600	15	505800
2	799972860	4781734800	799980000	2056816300	399317077	1217964300	1632	0	12	0
3	799970955	4790704900	799980000	2049337300	399184847	1216590600	40000	500000	15	0
4	799895334	4819484500	799980000	2045878200	399966881	1224087600	40000	500900	15	0
5	799968675	4771741300	799980000	2065335200	401484298	1224736700	15628	0	16	0
6	799974849	4813406300	799980000	2048339700	396751395	1208695400	40000	521400	15	0
7	799968372	4788959600	799980000	2063284100	399641183	1220800300	40000	0	15	0
8	799799700	4799846000	799980000	2058540200	400054047	1218348100	40000	502500	15	500800
9	799963529	4795580900	799980000	2063126000	399217080	1216652900	40000	625300	16	501700
10	799944489	4846784200	799980000	2056354500	399744571	1220112200	13896	500500	15	0
11							40000	0	16	0
12							40000	499700	15	0
13							40000	500500	15	0
14							8166	0	15	0
15							40000	0	16	0
16							40000	0	16	0
17							40000	499200	14	500500
18							40000	0	15	0
19							535	0	16	0
20							40000	0	12	0
21							32114	585400	15	503300
22							40000	0	15	0
23							40000	500400	15	0
24							653	0	15	505000
25							40000	500400	15	0
26							38096	499600	16	0
27							40000	502100	15	0
28							19339	0	15	0
29							40000	500000	15	0
30							40000	500800	16	626500

Raw Data: Size 50000

	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	1249950469	7587974700	1249975000	3216344700	624149689	1899280700	50000	499700	16	0
2	1249871260	7603235500	1249975000	3222264300	623094310	1893679300	50000	0	16	0
3	1249972225	7613825600	1249975000	3247451300	627069648	1917250400	24785	0	16	500500
4	1249882765	7595478200	1249975000	3246983800	627121156	1918300100	11310	0	16	521800
5	1249940284	7597822500	1249975000	3204507400	622946783	1897035900	33917	503300	14	499200
6	1249915315	7598756500	1249975000	3221237600	621376711	1895422000	50000	0	15	500000
7	1249970344	7602107700	1249975000	3212847600	625185459	1906132800	43787	500400	15	500100
8	1249937050	7596343300	1249975000	3234327400	624747204	1911092200	50000	0	15	213900
9	1249967374	7601610200	1249975000	3219769800	625083892	1897287900	50000	499200	16	656100
10	1249952845	7606427100	1249975000	3223212700	626447721	1909669600	30068	500000	16	500500
11							50000	0	16	0
12							50000	500000	16	500500
13							50000	500500	16	500400
14							50000	0	16	0
15							2371	0	15	0
16							50000	0	16	0
17							50000	0	16	0
18							50000	503300	15	0
19							4332	499600	16	0
20							32672	0	16	0
21							50000	0	13	0
22							50000	500900	11	499600
23							18995	548500	15	502900
24							50000	796100	16	525900
25							50000	0	16	0
26							17275	503000	15	500400
27							50000	526000	16	0
28							47243	502500	15	0
29							2230	501200	16	0
30							50000	0	15	533700



Raw Data: Size 60000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	1799960820	11102783100	1799970000	4641899100	896499632	2751943100	14849	999200	16	0
2	1799966414	11106055500	1799970000	4668988100	900927047	2766517500	28224	982000	16	0
3	1799926044	11146987000	1799970000	4679432800	900845733	2772412500	60000	996800	16	0
4	1799961872	11053961700	1799970000	4622747100	898154870	2736401100	60000	500900	16	499600
5	1799832974	11083784600	1799970000	4619302700	896734425	2721324600	43970	0	16	0
6	1799950100	11017105900	1799970000	4646794000	901219183	2737699200	27257	0	15	0
7	1799836097	11085762300	1799970000	4626421500	898448557	2739161900	60000	501300	15	498000
8	1799906454	11037047600	1799970000	4641833400	900014844	2741266400	60000	502500	16	501300
9	1799958825	11028320300	1799970000	4624501400	900036636	2743471500	60000	0	16	0
10	1799928384	11103417400	1799970000	4657657500	900799780	2756653700	60000	500500	15	500100
11							57202	0	16	0
12							34702	501700	14	500400
13							22741	0	16	0
14							60000	500000	16	500500
15							14368	500100	16	500400
16							31400	501200	16	499700
17							60000	0	14	0
18							16896	504100	16	0
19							60000	501200	15	499300
20							7080	500000	16	500600
21							33046	0	14	501300
22							60000	0	16	0
23							60000	998400	15	0
24							18309	0	15	0
25							28414	507000	16	0
26							60000	0	16	0
27							46832	500100	15	0
28							52427	522200	16	502900
29							60000	0	16	0
30							60000	500500	16	0

Raw Data: Size 70000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	2449905315	15200360100	2449965000	6327376400	1222931914	3764088300	45602	0	16	0
2	2449943472	15223181500	2449965000	6373872400	1227548243	3748436700	70000	0	16	0
3	2449872335	15275485400	2449965000	6326742200	1223388860	3744603800	70000	0	15	0
4	2449803972	15255585000	2449965000	6363622700	1228749571	3747782200	70000	0	17	1000900
5	2449946472	15225820400	2449965000	6365012400	1226835283	3820103600	70000	1000500	12	0
6	2449964439	15180472000	2449965000	6329466600	1225247685	3746656500	70000	0	16	1000500
7	2449717544	15193625400	2449965000	6345488100	1224294966	3754638300	20942	0	16	0
8	2449801122	15157415700	2449965000	6311336000	1222167953	3832748000	37627	0	15	0
9	2449757310	15227528800	2449965000	6324537900	1223131223	3781681400	70000	0	15	0
10	2449833672	15128064500	2449965000	6353733100	1224523666	3745587800	54104	0	16	1001300
11							57802	0	16	1014900
12							2151	998800	15	0
13							70000	0	16	1000100
14							70000	0	16	1001700
15							70000	1001000	16	0
16							70000	0	16	0
17							42302	0	16	0
18							70000	0	16	1014800
19							70000	0	16	1000500
20							70000	0	15	0
21							70000	0	15	0
22							70000	0	16	1000500
23							70000	0	14	0
24							70000	0	14	0
25							70000	0	15	0
26							11783	0	16	0
27							70000	0	16	0
28							384	0	17	0
29							70000	998900	16	0
30							30820	0	16	0

Raw Data: Size 80000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	3199942795	19918934800	3199960000	8258752000	1598665694	4889010100	15134	0	16	1003800
2	3199958170	19829310100	3199960000	8254577100	1606586485	4908318900	80000	0	15	0
3	3199940100	19931172700	3199960000	8330896900	1596242968	4893502300	28632	0	17	999300
4	3199947910	19842892600	3199960000	8264399000	1600467990	4900536500	39458	974300	17	0
5	3199783879	19865222600	3199960000	8264535400	1600942853	4930281900	80000	0	16	0
6	3199741209	19927491000	3199960000	8289849200	1602486156	4901947200	80000	1000100	16	0
7	3199927104	19908817700	3199960000	8294359100	1602330009	4904497100	4128	0	16	992300
8	3199798404	19833205000	3199960000	8279971100	1603708763	4912951900	80000	0	16	0
9	3199812304	19834795400	3199960000	8274839200	1590870319	4890066000	80000	999600	16	0
10	3199860765	19879521100	3199960000	8266521200	1600197869	4892918100	30477	0	16	1000900
11							14887	1000100	16	0
12							69897	0	17	0
13							80000	0	15	0
14							80000	979600	16	0
15							80000	0	16	1001300
16							38220	0	9	1000500
17							68745	990200	12	0
18							65808	0	15	1002100
19							80000	1000500	16	0
20							12755	0	16	0
21							80000	0	16	1000500
22							80000	0	17	1000900
23							80000	0	16	1000900
24							40693	500000	16	0
25							15520	998800	12	0
26							80000	0	17	0
27							14193	0	17	0
28							80000	0	16	1000100
29							68712	0	16	0
30							80000	999700	16	0

Raw Data: Size 90000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	4049883747	25125719900	4049955000	10469388100	2021233355	6197301400	90000	1001400	15	0
2	4049934294	25250824400	4049955000	10457110200	2028922663	6234157900	35591	0	17	0
3	4049879145	25248841000	4049955000	10493414100	2032966317	6204526300	48340	1001300	16	0
4	4049953404	25226818500	4049955000	10573494800	2023675761	6210636000	90000	988200	13	0
5	4049862765	25298993000	4049955000	10526250700	2018822374	6199030900	76214	0	17	0
6	4049826729	25147822100	4049955000	10568048000	2026996738	6207426000	82809	0	17	1001400
7	4049914530	25148754000	4049955000	10535434200	2023144991	6203377500	90000	0	17	0
8	4049950722	25178744400	4049955000	10462095900	2017182772	6157369800	79237	0	16	0
9	4049944989	25280818000	4049955000	10431692500	2024849234	6214370400	90000	1000500	16	0
10	4049949222	25341232300	4049955000	10464932400	2029448291	6182319800	90000	999300	15	0
11							49745	0	17	0
12							8452	0	14	0
13							90000	0	15	999700
14							90000	0	17	0
15							90000	0	17	0
16							90000	0	17	1001300
17							90000	999700	16	1000500
18							80017	991500	13	0
19							15746	0	17	0
20							90000	0	16	1000500
21							90000	1000100	14	0
22							84894	992700	17	0
23							20596	1000500	16	0
24							40994	0	17	0
25							83466	982400	16	0
26							90000	1000500	17	0
27							37026	1000100	13	0
28							90000	0	16	0
29							73611	0	16	0
30							90000	987300	15	1003800

Raw Data: Size 100000										
	Bubble		Selection		Insetion		Linear		Binary	
	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time	Comparisons	Time
1	4999772690	31224681600	4999950000	12932899200	2497281105	7652617200	56616	0	13	0
2	4999797372	31193296200	4999950000	12981697100	2498368604	7715700500	45518	1000500	17	0
3	4999719140	31126117700	4999950000	13029456700	2492914865	7623287000	100000	1042800	16	0
4	4999851654	31252640200	4999950000	12927121900	2500558932	7649133300	9803	0	13	0
5	4999941222	31166083600	4999950000	13020923500	2505158451	7662669500	45905	0	17	0
6	4999648524	31209834700	4999950000	12925724500	2507395797	7706053900	76580	985700	16	1000500
7	4999903029	31161449700	4999950000	12953189500	2500625462	7642481300	87953	0	17	1000500
8	4999926347	31310071500	4999950000	12909959400	2496432598	7651386800	65627	0	17	1000500
9	4999782669	31148467100	4999950000	13037918100	2501511509	7682537000	96759	0	16	0
10	4999971354	31136059500	4999950000	12927007400	2505233585	7663927700	7653	0	16	0
11							94595	0	15	0
12							72522	1000100	14	0
13							100000	0	17	0
14							42677	0	17	0
15							100000	977600	15	0
16							63833	0	15	0
17							61735	0	16	0
18							100000	978000	17	0
19							15608	0	16	0
20							78998	1000900	15	0
21							74853	0	17	0
22							50387	1000500	14	1000500
23							10390	0	17	0
24							80789	0	16	0
25							100000	0	15	0
26							43627	0	14	1000500
27							41669	0	14	1000900
28							100000	0	17	0
29							914	0	16	0
30							56835	0	17	0