CSE 204

Data Structures & Algorithms

A runtime comparison between selection sort and insertion sort

Submitted By

Soham Khisa

S201705120

Submitted to

Prof. Dr. Md Abul Kashem Mia

Asst. Prof. Dr. Sadia Sharmin

Asst. Prof. Ishat E Rabban

Date of Submission

June 9, 2019

Objective:

The objective of the assignment was to get a practical idea about the runtime of two sorting techniques: selection sort and insertion sort. Plotting the observed runtime against input size of an array would give insight into the efficiency of the algorithms.

Used Platforms:

CodeBlocks IDE was used to run the code in C++14. The data about runtime was plotted using Microsoft Excel.

Procedure:

The program would take an input from the user as the size of an array to be sorted. The program would generate the elements of the array as it is instructed to. At first it would generate a sorted array to evaluate the best case by calling the function of selection sort and insertion sort. It would record the run time needed for each of this functions. Again it would generate an unsorted array and sort it by calling the function of selection sort and insertion sort to evaluate the average case and record the runtime. Finally, it would generate an unsorted array actually sorted in the descending order repeat the steps again to measure the worst case. After all these are done we had to plot the input size of the array against the runtime of each function on the graph separately for the best case, the average case and for the worst case.

Time Complexity Analysis:

Selection sort:

```
for(unsigned int i=0; i<n-1; i++) {
        mini = i;
      for(j=i+1; j<n; j++)
        if(ar[j]<ar[mini]) mini = j;
      temp = ar[mini];
      ar[mini] = ar[i];
      ar[i] = temp;
  }
```

Best Case: In the best case the array is already sorted, the program will execute the first line which is a loop. The outer loop will conduct for n-1 number of times inside the this loop there is another loop which although the array is sorted will still conduct some number of times depending on each time the outer loop runs. So the time complexity of the inner loop becomes O(n). In this case it will not satisfy the 'if' statement inside the inner loop and will not conduct the statement holds. Since the outer loop runs for n-1 of times, the time complexity will be, n*O(n) = O(n^2).

Average Case: In this case the array is sorted in some portions and not sorted in some portions. Its description is almost similar to the previous case only difference is that inside the inner loop the if statement will conduct for the times when the parts of the array will be founded unsorted and will not conduct when sorted. But the time complexity of the statement inside the 'if' has a linear time

complexity of O(1), which is not dominant. So in this case the time complexity remains the same as for the best case. It is O(n^2).

Worst Case: For the worst case, the array is sorted in the descending order, means inversely sorted. The 'if' statement will satisfy every single time the loop runs and so the statement inside it whose complexity is not dominant here. The rest of the time complexity remains the same as the previous two. So in the worst case it still holds the time complexity of O(n^2).

Insertion sort:

```
for(int i=0; i<n-1; i++) {
      j = i;
      while(ar[i]>ar[i+1])
      {
         temp = ar[i+1];
         ar[i+1] = ar[i];
         ar[i] = temp;
         i--;
         if(i<0) break;
      }
      i = j;
   }
```

Best Case: For the best case when the array is sorted the first line of the code will conduct. The line is a loop which runs for n-1 number of times. In the third line there is an inner loop. But as the condition for the inner loop does not satisfy as the array is sorted it will not run. So the time complexity becomes O(n).

Average Case: In the average case the inner loop will conduct for sometimes but not every time the outer loop runs. Still it holds the time complexity of O(n). The outer loop runs for n-1 number of times which means the overall time complexity will be n*O(n) that is O(n^2).

Worst Case: In case of worst case the inner loop runs every time the outer loop runs. But still its time complexity is O(n). So it remains same as the average case which is O(n^2).

Machine Configuration:

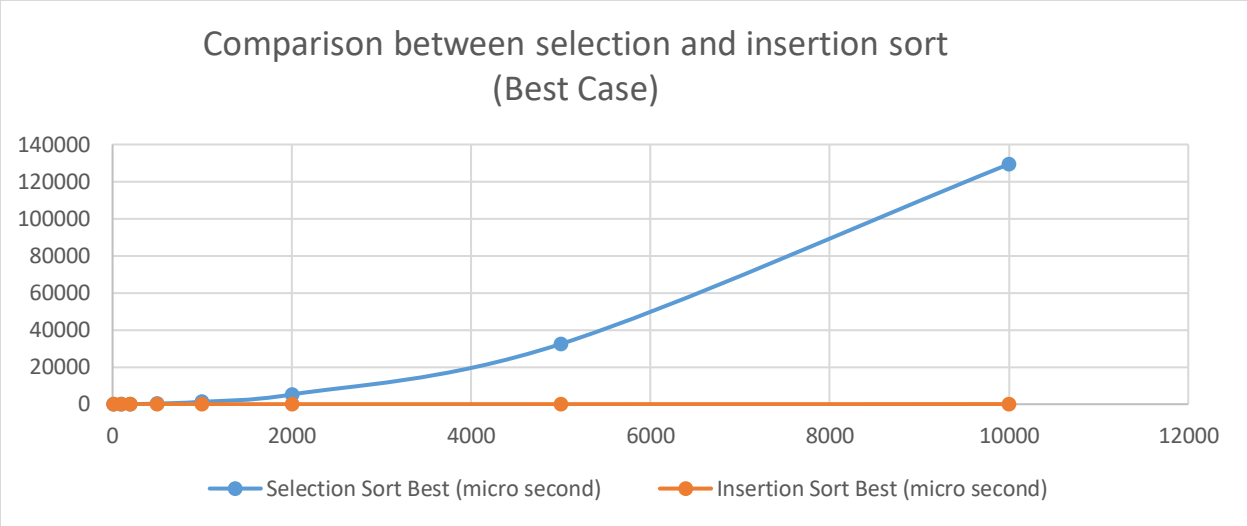
Processor: Inter(R) Core(TM) i5 – 7200U @ 2.50 GHz 2.71 GHz

Installed memory (RAM): 8.00 GB (7.89 GB usable)

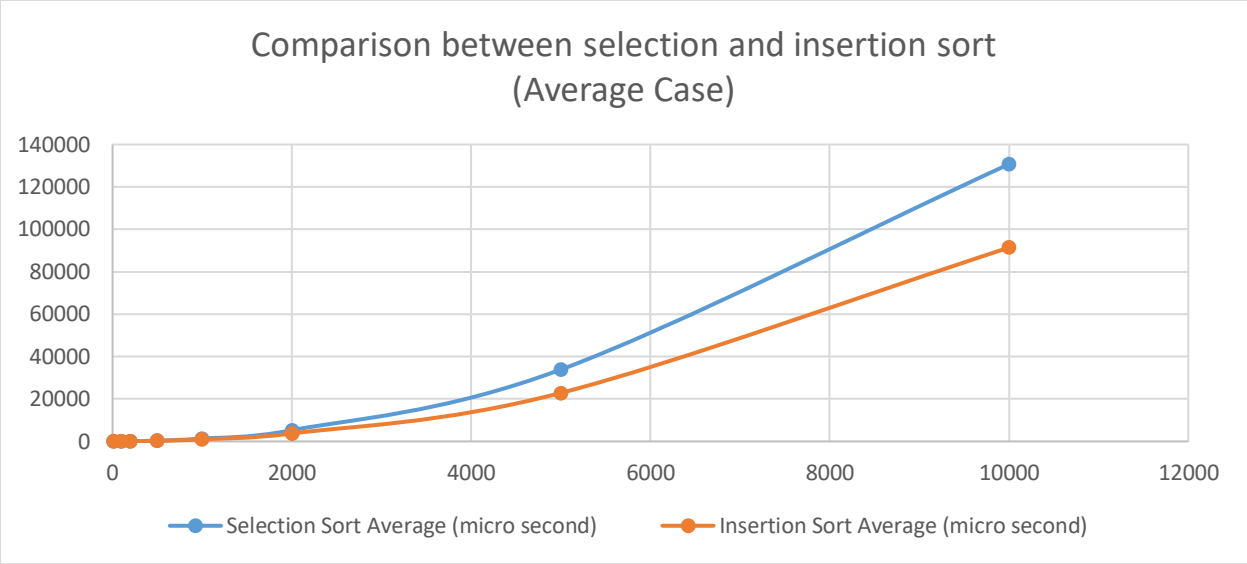
System type: 64-Bit Operating System, x64-based processor

Graph:

Input size n	Selection Sort Best (micro second)	Insertion Sort Best (micro second)
10	0.377581	0
100	13.5929	0.377581
200	52.1062	1.13274
500	328.873	2.64307
1000	1327.57	4.90855
2000	5231.01	9.8171
5000	32453.5	23.41
10000	129409	46.82



Input size n	Selection Sort Average (micro second)	Insertion Sort Average (micro second)
10	0.377581	0.377581
100	15.4808	10.1947
200	69.4749	58.9026
500	343.976	237.498
1000	1349.47	1011.54
2000	5266.12	3744.85
5000	33853.9	22825.5
10000	130724	91362.5



Input size n	Selection Sort Worst(micro second)	Insertion Sort Worst(micro second)
10	0.755162	0.377581
100	18.5015	20.7669
200	64.5663	131.398
500	401.369	542.584
1000	1397.43	2316.08
2000	5591.6	8087.41
5000	34860.9	50572.8
10000	141157	204778