Offline 1

Name : Md Sabbir Rahman

Roll : 1705076

Section : B1

Course : CSE 204, Data Structure and Algorithms Sessional

Submission Date : 10-06-2019

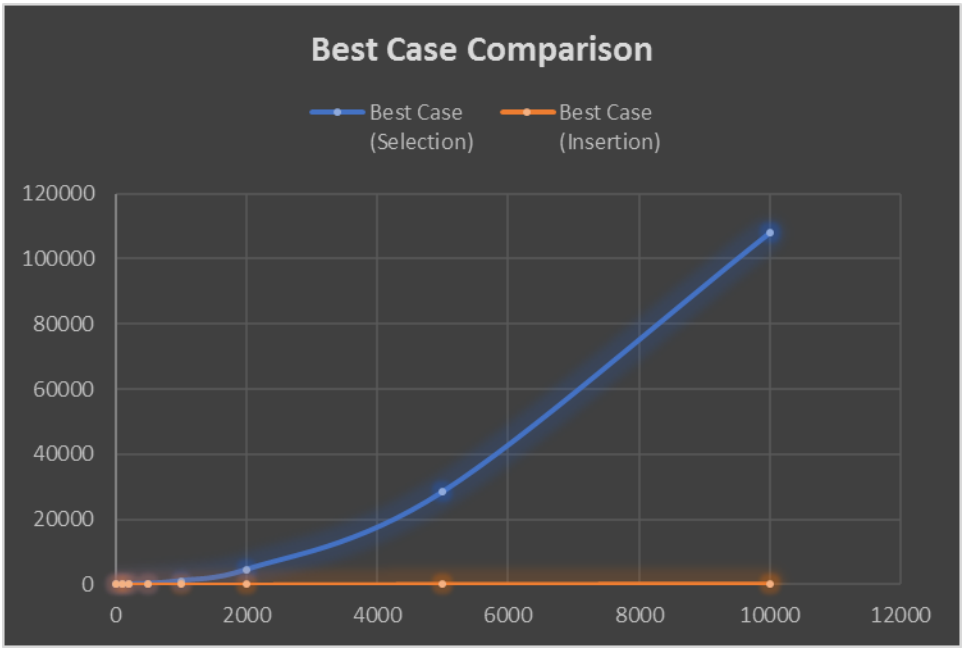
Objective: The objective is to run insertion sort and selection sort on arrays of different sizes and compare their worst case, best case and average case runtime.

Machine Specs: Intel Core i5-4460 CPU @ 3.20 GHz, 8.00 GB, x64 based processor. Windows 8.1 Operating System.

Data: In the enclosed cpp file, Insertion sort and Selection sort were implemented and run on randomly generated array of various sizes (after sorting). For average case, a random array was taken, for best case, an already sorted array was generated, for worst case a completely reverse sorted array was taken. The execution time has been added as a excel file with the report. We notice that for worst and average case both function show comparable result, though selection sort is slower. But for best case, insertion sort is a lot faster than selection sort. Almost all cases show quadratic complexity except best case of insertion sort which shows linear complexity.

	Selection Sort			Insertion Sort		
Array Size n	Average Case (Selection)	Best Case (Selection)	Worst Case (Selection)	Average Case (Insertion)	Best Case (Insertion)	Worst Case (Insertion)
10	0.64	0.32	0.32	0.32	0.32	0.32
100	16.99	12.51	17.64	8.98	0.64	14.43
200	56.77	54.53	60.29	29.5	0.96	59.97
500	308.55	282.25	348	174.8	2.25	348.64
1000	1187.7	1121.31	1477.3	683.5	4.17	1374.1
2000	4562.8	4485.8	6321.1	2768.6	8	5394.8
5000	28372	28413	43020	17772	23.9	35846
10000	107500	107953	158966	67006	40.1	130913


Average Case Comparison
Average Case (Selection)
Average Case (Insertion)
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000
10000
12000


Best Case Comparison
Best Case (Selection)
Best Case (Insertion)
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
8000
10000
12000

Complexity Analysis: Selection Sort and Insertion Sort both show quadratic time complexity. For Selection sort, the outer loop iterates O(n) times, everytime finding the next element in a place. To do that an inner loop iterates the remaining number of positions. So running time = n + (n-1) + ... + 3 + 2 + 1 = $\frac{n(n+1)}{2}$ = O(n^2).

For Insertion sort, the outer loop iterates O(n) times. but everytime it finds the next element's place in already processed places. In the process some elements are shifted to their next position. In the best case only one element is required to be shifted. resulting runtime = 1 + 1+ … + 1 + 1 + 1 = O(n). In the Worst case, all numbers are shited to next position, resulting runtime = 1 + 2 + … + n = $\frac{n(n+1)}{2}$ = O(n^2).

Discussion: The running time were measured by using windows' QueryPerformanceCounter function which is very accurate. The time was measured before and after calling the sort functions on copied arrays to only measure execution time of the function. With even bigger inputs, The actual complexity of the functions would become more clear.