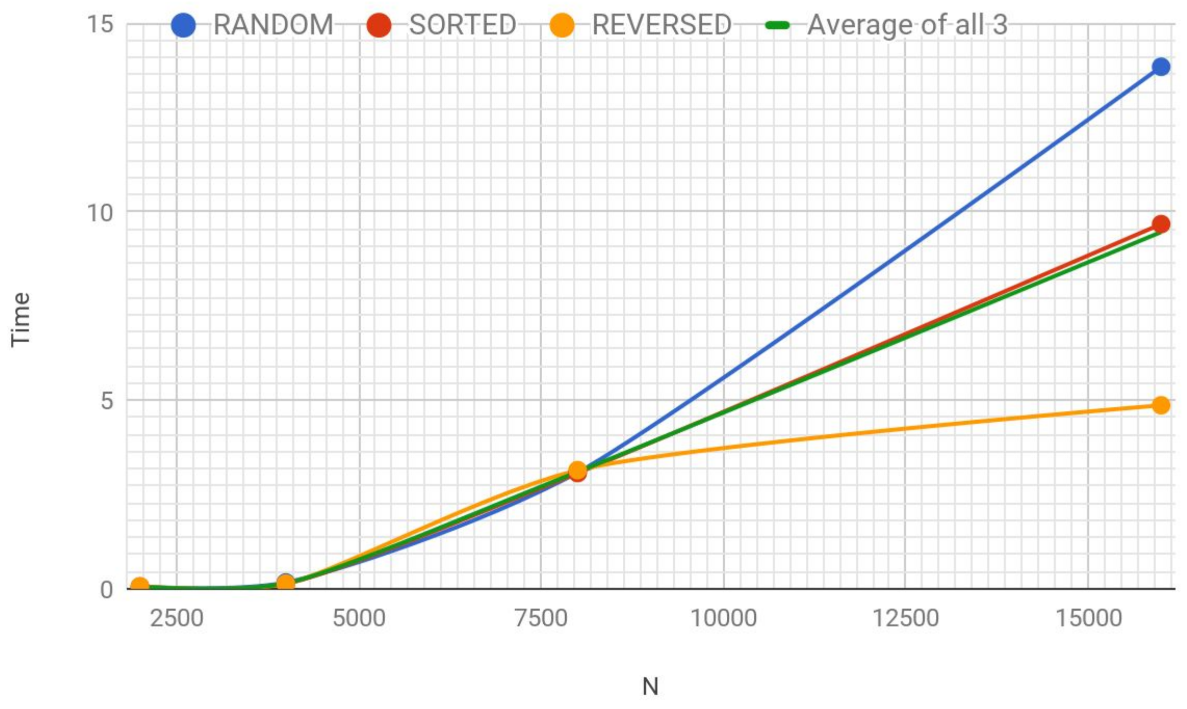
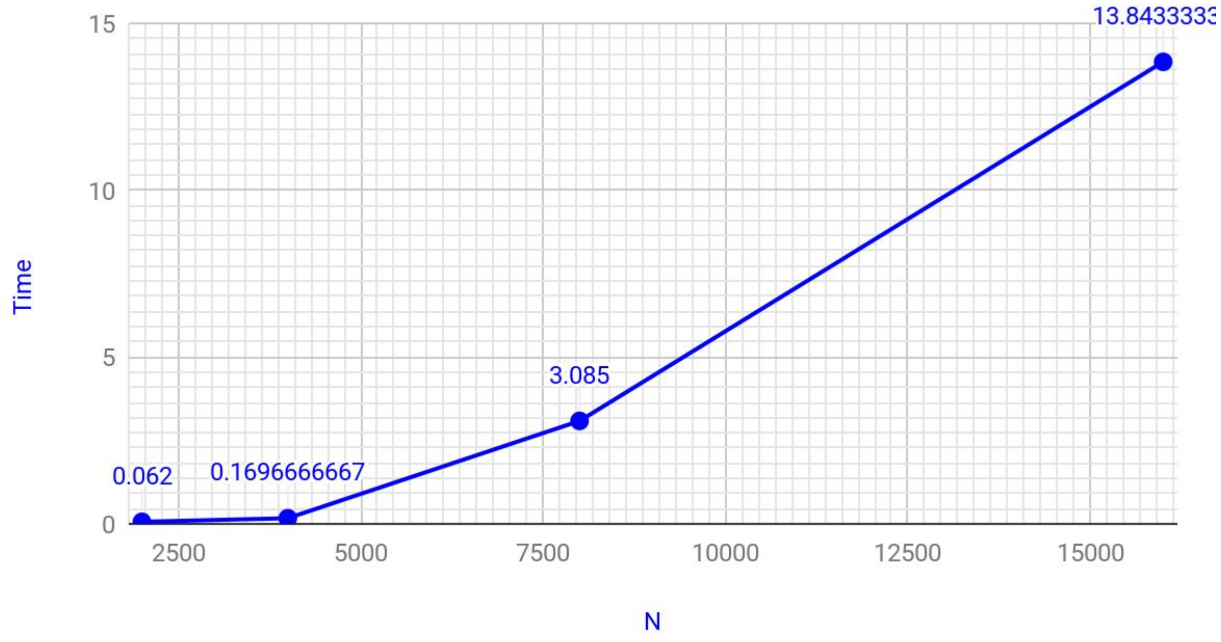
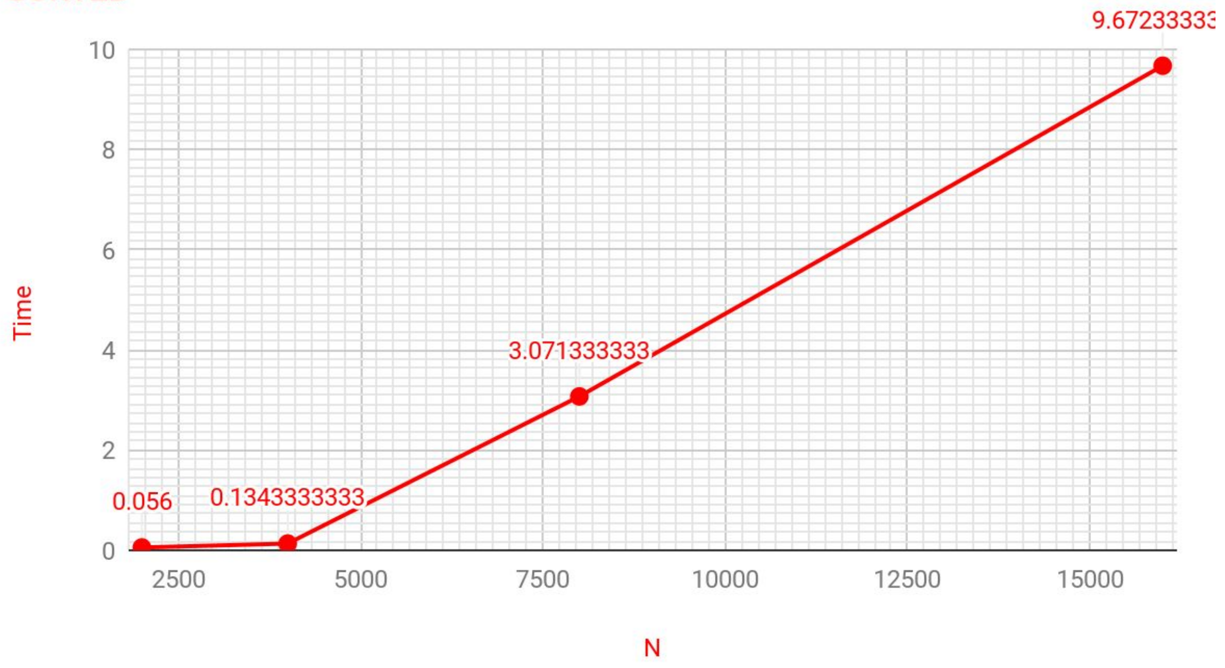

RANDOM
SORTED
REVERSED
Average of all 3
Time
N
15
10
5
0
2500
5000
7500
10000
12500
15000

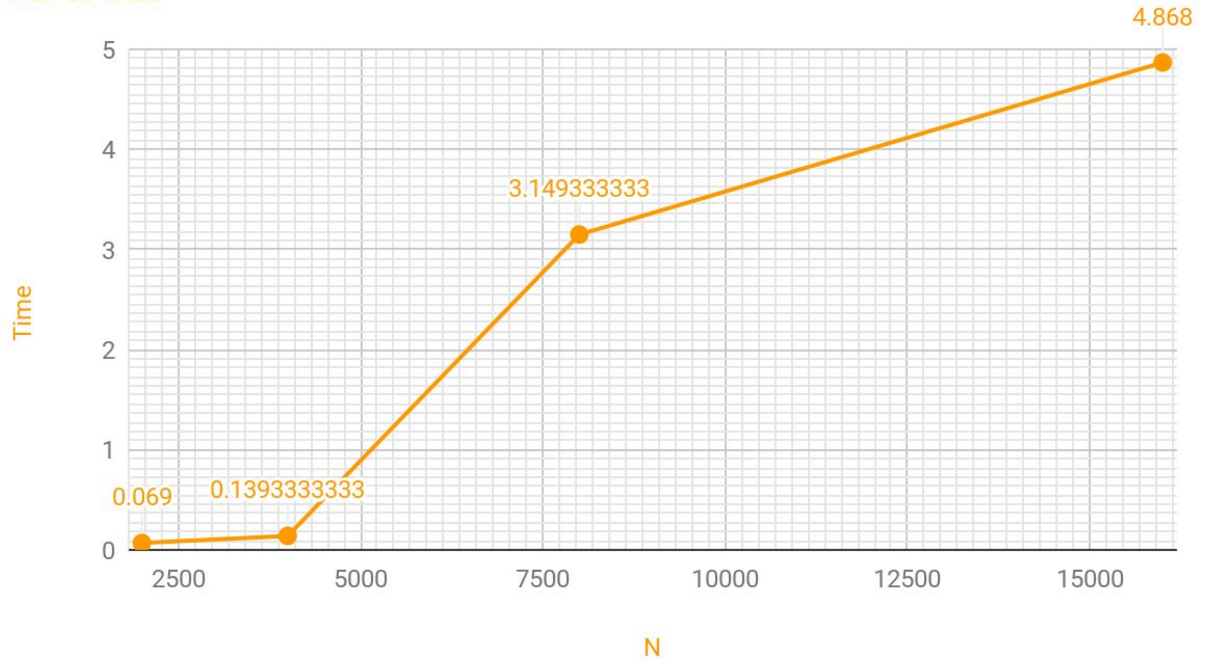
RANDOM


Time
N
0.062
0.1696666667
3.085
13.8433333
15
10
5
0
2500
5000
7500
10000
12500
15000

SORTED



REVERSED



3.
Analyzing the SelectionSort runtime using the doubling method shows a non-linear growth pattern. The average runtime trend between the 3 different types of input is similar to the sorted runtime trend. Using systems of equations, the average b between each of trials of N (0.619, 2.5, 2.8) is 1.973, or approximately 2. This results a runtime of N^2 (though there seems to be an obvious c value which is at play). From the graph it appears that every time the input is doubled, the output seems to be increasing x2, x4, x8 times which goes to further suggest a n^2 runtime

4.
Looking at the Selection code, the if statement where the array is accessed is nested inside two for loops. The first for loop loops N times, while the second loops ½ N for each loop of the first. This results in ½ N^2 times the if statement is looped. Because there are 2 array accesses for each time the if statement happens, the total number of array acesses is ~$O(N^2)$

5.

From my data, the order of the input does affect the running time. However, looking at the code, it suggests otherwise.

exch() is called regardless of whether or not the array is sorted. Because the selection sort code uses a for loop to increment through an array to test for necessary exchanges, exch() is called N times, where N times represents intterating through the array in entirety once. less(), however, is located within a nested for loop which also increments ½ N times. As shown from the code, regardless of the order input, the algorithm still has to itterate the same amount of times to verify each element.