CS 202 Spring 2021

Homework 1

Algorithm Efficiency and Sorting Report

Arda Önal

21903350

Section 3

Question 1:

(a) To show that $5n^3+4n^2+10$ is $O(n^4)$, we need to find c and n_0 which satisfies $0 \leq 5n^3+4n^2+10 \leq cn^4$ for all $n \geq n_0$. If we take c = 5 and $n_0 = 2$, it can be seen that $0 \leq 5n^3+4n^2+10 \leq 5n^4$ for all $n \geq 2$. Therefore, by Big-O definition c = 5 and $n_0 = 2$ are appropriate values which proves that $5n^3+4n^2+10$ is $O(n^4)$.

(b) Tracing of Insertion Sort:

```
void insertionSort(DataType theArray[], int n) {

  for (int unsorted = 1; unsorted < n; ++unsorted) {

    DataType nextItem = theArray[unsorted];
    int loc = unsorted;

    for (  ;(loc > 0) && (theArray[loc-1] > nextItem); --loc)
       theArray[loc] = theArray[loc-1];

    theArray[loc] = nextItem;
  }
}
```

Initial array:

[24, 8, 51, 28, 20, 29, 21, 17, 38, 27]

Store 8, shift 24 and insert 8:

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27]

Store 51 and insert 51:

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27]

Store 28, shift 51 and insert 28:

[8, 24, 28, 51, 20, 29, 21, 17, 38, 27]

Store 20, shift 24, 28, 51 and insert 20:

[8, 20, 24, 28, 51, 29, 21, 17, 38, 27]

Store 29, shift 51 and insert 29:

[8, 20, 24, 28, 29, 51, 21, 17, 38, 27]

Store 21, shift 24, 28, 29, 51 and insert 21:

[8, 20, 21, 24, 28, 29, 51, 17, 38, 27]

Store 17, shift 20, 21, 24, 28, 29, 51 and insert 17:

[8, 17, 20, 21, 24, 28, 29, 51, 38, 27]

Store 38, shift 51 and insert 38:

[8, 17, 20, 21, 24, 28, 29, 38, 51, 27]

Store 27, shift 28, 29, 38, 51 insert 27:

[8, 17, 20, 21, 24, 27, 28, 29, 38, 51]

The initial array is sorted.

Tracing of Bubble Sort:

```
void bubbleSort( DataType theArray[], int n) {
   bool sorted = false;

   for (int pass = 1; (pass < n) && !sorted; ++pass) {
      sorted = true;
      for (int index = 0; index < n-pass; ++index) {
         int nextIndex = index + 1;
         if (theArray[index] > theArray[nextIndex]) {
            swap(theArray[index], theArray[nextIndex]);
            sorted = false; // signal exchange
         }
      }
   }
}
```

Initial array:

[24, 8, 51, 28, 20, 29, 21, 17, 38, 27|] ("|" indicates differentiates sorted and unsorted part)

Pass 1:

[24, 8, 51, 28, 20, 29, 21, 17, 38, 27|] Swap

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27|] Don’t Swap

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27|] Swap

[8, 24, 28, 51, 20, 29, 21, 17, 38, 27|] Swap

[8, 24, 28, 20, 51, 29, 21, 17, 38, 27|] Swap

[8, 24, 28, 20, 29, 51, 21, 17, 38, 27|] Swap

[8, 24, 28, 20, 29, 21, 51, 17, 38, 27|] Swap

[8, 24, 28, 20, 29, 21, 17, 51, 38, 27|] Swap

[8, 24, 28, 20, 29, 21, 17, 38, 51, 27|] Swap

[8, 24, 28, 20, 29, 21, 17, 38, 27, |51] Pass 1 complete and 51 is placed into sorted part.

Pass 2:

[8, 24, 28, 20, 29, 21, 17, 38, 27, |51] Don’t Swap

[8, 24, 28, 20, 29, 21, 17, 38, 27, |51] Don’t Swap

[8, 24, 28, 20, 29, 21, 17, 38, 27, |51] Swap

[8, 24, 20, 28, 29, 21, 17, 38, 27, |51] Don’t Swap

[8, 24, 20, 28, 29, 21, 17, 38, 27, |51] Swap

[8, 24, 20, 28, 21, 29, 17, 38, 27, |51] Swap

[8, 24, 20, 28, 21, 17, 29, 38, 27, |51] Don’t Swap

[8, 24, 20, 28, 21, 17, 29, 38, 27, |51] Swap

[8, 24, 20, 28, 21, 17, 29, 27|38, 51] Pass 2 complete and 38 is placed into sorted part.

Pass 3:

[8, 24, 20, 28, 21, 17, 29, 27|38, 51] Don’t Swap

[8, 24, 20, 28, 21, 17, 29, 27|38, 51] Swap

[8, 20, 24, 28, 21, 17, 29, 27|38, 51] Don’t Swap

[8, 20, 24, 28, 21, 17, 29, 27|38, 51] Swap

[8, 20, 24, 21, 28, 17, 29, 27|38, 51] Swap

[8, 20, 24, 21, 17, 28, 29, 27|38, 51] Don’t Swap

[8, 20, 24, 21, 17, 28, 29, 27|38, 51] Swap

[8, 20, 24, 21, 17, 28, 27|29, 38, 51] Pass 3 complete and 29 is placed into sorted part.

Pass 4:

[8, 20, 24, 21, 17, 28, 27|29, 38, 51] Don’t Swap

[8, 20, 24, 21, 17, 28, 27|29, 38, 51] Don’t Swap

[8, 20, 24, 21, 17, 28, 27|29, 38, 51] Swap

[8, 20, 21, 24, 17, 28, 27|29, 38, 51] Swap

[8, 20, 21, 17, 24, 28, 27|29, 38, 51] Don’t Swap

[8, 20, 21, 17, 24, 28, 27|29, 38, 51] Swap

[8, 20, 21, 17, 24, 27|28, 29, 38, 51] Pass 4 complete and 28 is placed into sorted part.

Pass 5:

[8, 20, 21, 17, 24, 27|28, 29, 38, 51] Don’t Swap

[8, 20, 21, 17, 24, 27|28, 29, 38, 51] Don’t Swap

[8, 20, 21, 17, 24, 27|28, 29, 38, 51] Swap

[8, 20, 17, 21, 24, 27|28, 29, 38, 51] Don’t Swap

[8, 20, 17, 21, 24, 27|28, 29, 38, 51] Don’t Swap

[8, 20, 17, 21, 24|27, 28, 29, 38, 51] Pass 5 complete and 27 is placed into sorted part.

Pass 6:

[8, 20, 17, 21, 24|27, 28, 29, 38, 51] Don’t Swap

[8, 20, 17, 21, 24|27, 28, 29, 38, 51] Swap

[8, 17, 20, 21, 24|27, 28, 29, 38, 51] Don’t Swap

[8, 17, 20, 21, 24|27, 28, 29, 38, 51] Don’t Swap

[8, 17, 20, 21|24, 27, 28, 29, 38, 51] Pass 6 complete and 24 is placed into sorted part.

Pass 7:

[8, 17, 20, 21|24, 27, 28, 29, 38, 51] Don't Swap

[8, 17, 20, 21|24, 27, 28, 29, 38, 51] Don't Swap

[8, 17, 20, 21|24, 27, 28, 29, 38, 51] Don't Swap

[8, 17, 20|21, 24, 27, 28, 29, 38, 51] Pass 7 complete and 21 is placed into sorted part.

Since there were no swaps in Pass 7, the "sorted" boolean will remain true which indicates that the array is sorted and the for loop will end.

Question 2:

```
INITIAL NUMBERS:
[12, 7, 11, 18, 19, 9, 6, 14, 21, 3, 17, 20, 5, 12, 14, 8]

SELECTION SORT:
[3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21]
compCount: 120
moveCount: 45

MERGE SORT:
[3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21]
compCount: 46
moveCount: 128

QUICK SORT:
[3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21]
compCount: 45
moveCount: 102

RADIX SORT:
[3, 5, 6, 7, 8, 9, 11, 12, 12, 14, 14, 17, 18, 19, 20, 21]


Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

```
--------------------------------------------------------------------------
Analysis of Selection Sort (Random Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                54                  17997000            17997
10000               150                 49995000            29997
14000               290                 97993000            41997
18000               474                 161991000           53997
22000               708                 241989000           65997
26000               982                 337987000           77997
30000               1300                449985000           89997

--------------------------------------------------------------------------
Analysis of Selection Sort (Ascending Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                49                  17997000            17997
10000               133                 49995000            29997
14000               264                 97993000            41997
18000               434                 161991000           53997
22000               648                 241989000           65997
26000               901                 337987000           77997
30000               1198                449985000           89997

--------------------------------------------------------------------------
Analysis of Selection Sort (Descending Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                52                  17997000            17997
10000               141                 49995000            29997
14000               273                 97993000            41997
18000               450                 161991000           53997
22000               670                 241989000           65997
26000               942                 337987000           77997
30000               1251                449985000           89997

--------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------
Analysis of Merge Sort (Random Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                9                   67747               151616
10000               6                   120530              267232
14000               5                   175374              387232
18000               7                   231994              510464
22000               9                   290023              638464
26000               10                  348982              766464
30000               12                  408570              894464

--------------------------------------------------------------------------
Analysis of Merge Sort (Ascending Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                2                   39152               151616
10000               3                   69008               267232
14000               5                   99360               387232
18000               6                   130592              510464
22000               7                   165024              638464
26000               8                   197072              766464
30000               9                   227728              894464

--------------------------------------------------------------------------
Analysis of Merge Sort (Descending Arrays)

Array Size          Elapsed Time(ms)    compCount           moveCount
6000                2                   36656               151616
10000               3                   64608               267232
14000               5                   94256               387232
18000               6                   124640              510464
22000               7                   154208              638464
26000               8                   186160              766464
30000               10                  219504              894464

--------------------------------------------------------------------------
```

Analysis of Quick Sort (Random Arrays)

Array Size	Elapsed Time(ms)	compCount	moveCount
6000	1	85756	140703
10000	1	152262	261936
14000	2	216810	365134
18000	2	300629	525374
22000	2	405204	668131
26000	4	429141	667370
30000	4	529565	861252

Analysis of Quick Sort (Ascending Arrays)

Array Size	Elapsed Time(ms)	compCount	moveCount
6000	90	17997000	23996
10000	230	49995000	39996
14000	470	97993000	55996
18000	760	161991000	71996
22000	1140	241989000	87996
26000	1590	337987000	103996
30000	2120	449985000	119996

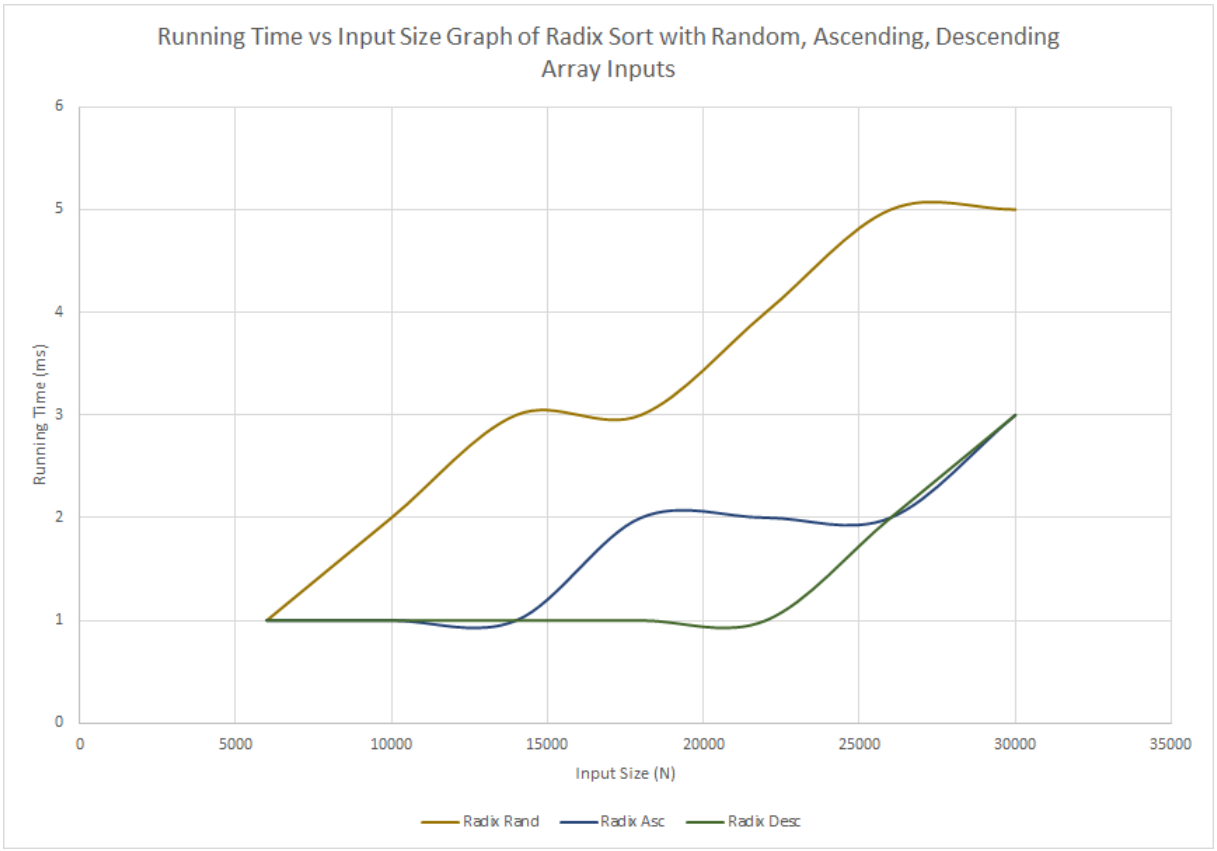
Analysis of Quick Sort (Descending Arrays)

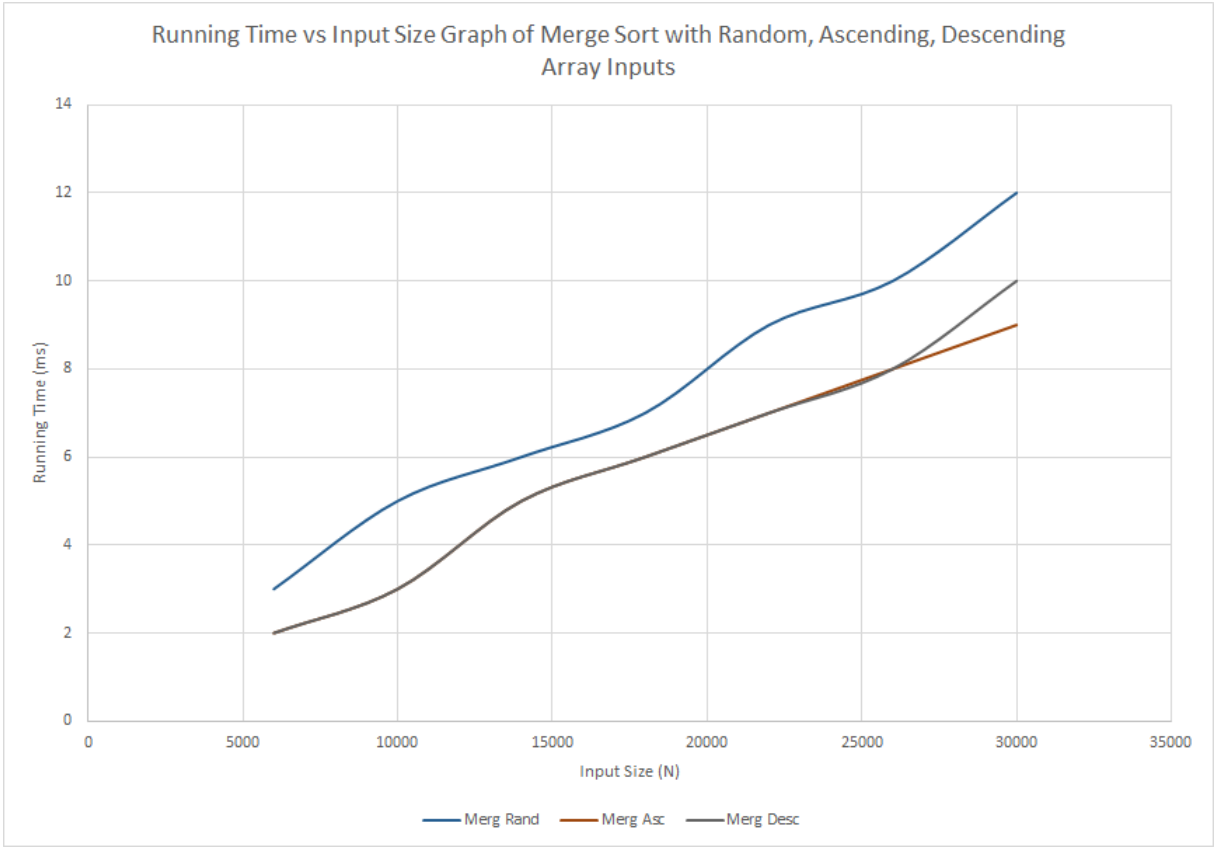
Array Size	Elapsed Time(ms)	compCount	moveCount
6000	120	17997000	27023996
10000	340	49995000	75039996
14000	660	97993000	147055996
18000	1100	161991000	243071996
22000	1640	241989000	363087996
26000	2280	337987000	507103996
30000	3050	449985000	675119996

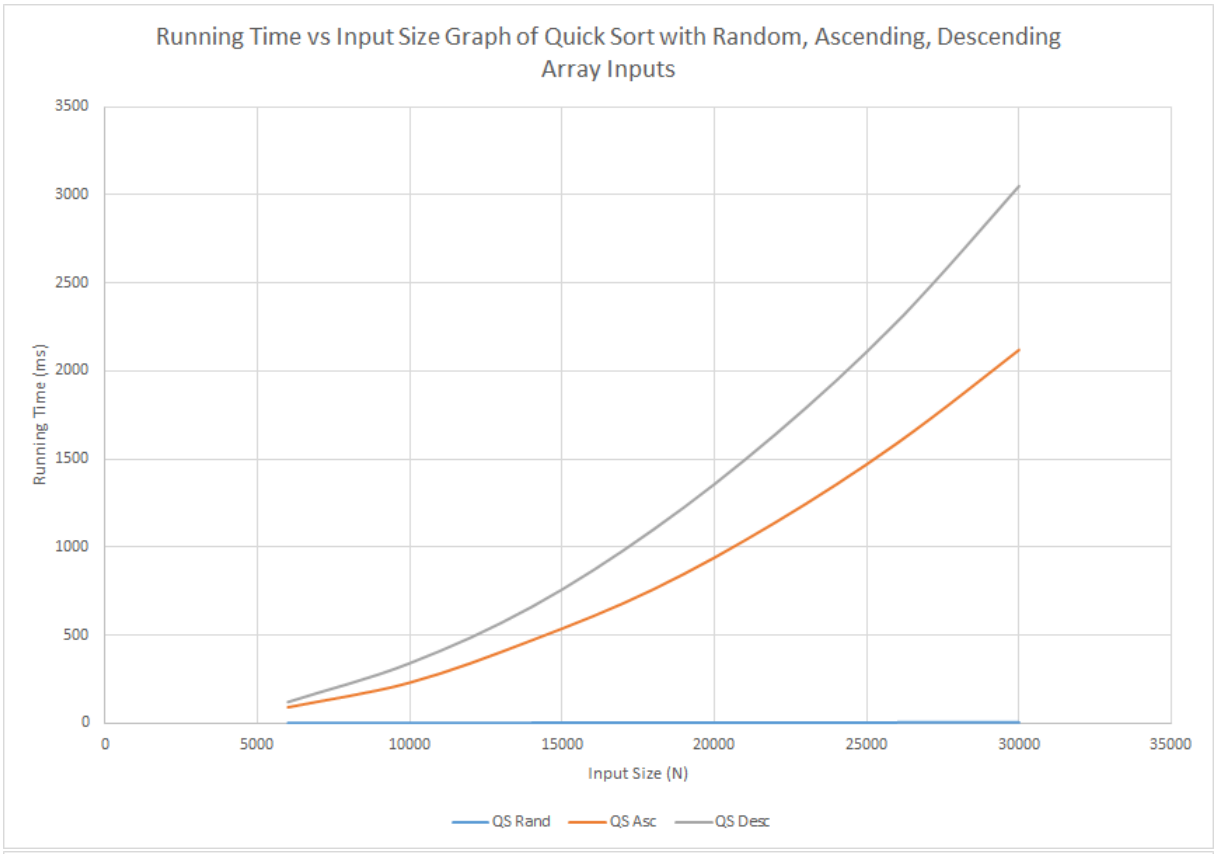
```
--------------------------------------------------------------------------------
Analysis of Radix Sort (Random Arrays)

Array Size          Elapsed Time(ms)
6000                1
10000               2
14000               3
18000               3
22000               4
26000               5
30000               5

--------------------------------------------------------------------------------
Analysis of Radix Sort (Ascending Arrays)

Array Size          Elapsed Time(ms)
6000                1
10000               1
14000               1
18000               2
22000               2
26000               2
30000               3

--------------------------------------------------------------------------------
Analysis of Radix Sort (Descending Arrays)

Array Size          Elapsed Time(ms)
6000                1
10000               1
14000               1
18000               1
22000               1
26000               2
30000               3
```

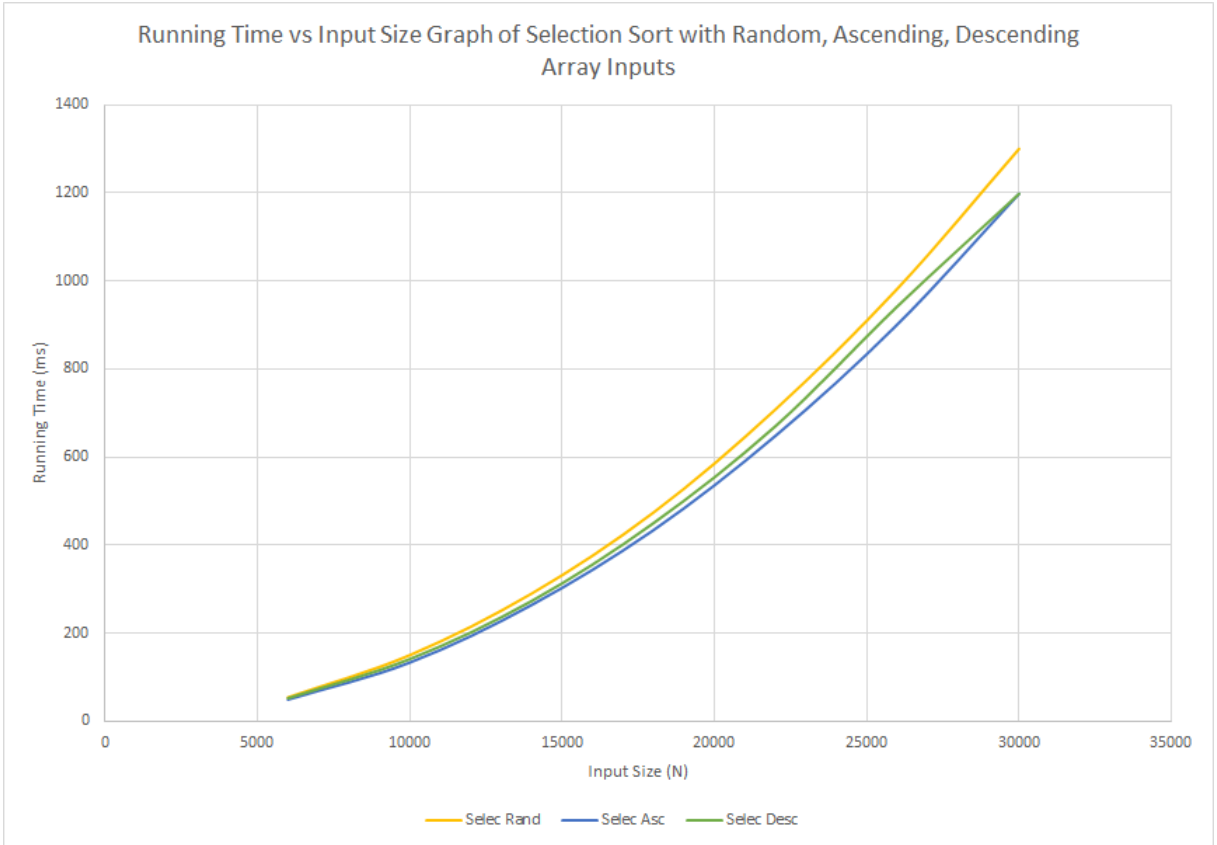
Question 3:

Combined graph:


Running Time vs Input Size Graph of Radix Sort with Random, Ascending, Descending Array Inputs
Running Time (ms)
0
1
2
3
4
5
6
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
Radix Rand
Radix Asc
Radix Desc


Running Time vs Input Size Graph of Merge Sort with Random, Ascending, Descending Array Inputs
Running Time (ms)
0
2
4
6
8
10
12
14
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
Merg Rand
Merg Asc
Merg Desc


Running Time vs Input Size Graph of Quick Sort with Random, Ascending, Descending Array Inputs
Running Time (ms)
3500
3000
2500
2000
1500
1000
500
0
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
QS Rand
QS Asc
QS Desc


Running Time vs Input Size Graph of Selection Sort with Random, Ascending, Descending Array Inputs
Running Time (ms)
1400
1200
1000
800
600
400
200
0
0
5000
10000
15000
20000
25000
30000
35000
Input Size (N)
Selec Rand
Selec Asc
Selec Desc

If we consider my data of selection sort, array size increases 5 times and the elapsed time increases around 25 times. This implies that selection sort is O (n^2). It is the same for random, ascending and descending inputs. This is because selection sort compares and moves the values regardless of the input. The theoretical time complexity of selection sort is also O (n^2). Hence, my empirical results match the theoretical values.

If we consider my data of merge sort, as array size increases, elapsed time does not increase rapidly. The elapsed time is actually very small that the time complexity cannot be observed with these sizes. The results are the same for random, ascending and descending cases. This occurs because merge sort works the same no matter what the input is. The theoretical value of merge sort is O (nlogn). This means that the elapsed time should not increase rapidly like a O(n^2) algorithm. Therefore, we can say that my empirical results match the theoretical values.

If we consider my data of quick sort, this time it is different for the random, ascending and descending case. For the random input, the elapsed time very similar to merge sort which is very small. This implies that quick sort with random inputs is O (nlogn). The theoretical value of it is also O (nlogn). Hence, my empirical results match the theoretical values. For the ascending input and descending input, quicksort behaves like selection sort. As array size increases, the elapsed time increases with squared. This implies that quick sort with ascending and descending array inputs is O (n^2). The theoretical value of it is also O(n^2). Hence, my empirical results match the theoretical values. The ascending and descending cases are worse than random because in our implementation, we choose the first item as pivot. Because of this, the algorithm tries to partition the array based on the first item but the first item is already smaller or bigger that all the other items in the array.

If we consider my data of radix sort, as my array size increases, the elapsed time increases linearly. It behaves as the time complexity of it is O (n). The theoretical time complexity of radix sort is also O(n). Hence, my empirical results match the theoretical values. It is the same for random, ascending and descending case because radix sort algorithm behaves the same for these 3 cases.