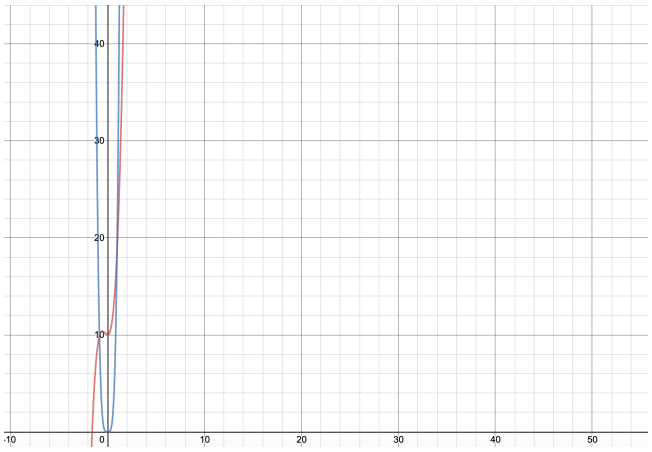
Question 1)
1- Show that $f(n) = 5n^3+4n^2+10 = O(n^4)$

We need to find two positive constants: c and n0 such that:
$0 <= 5n^3+4n^2+10 <= cn^4$ for all $n >= n0$
Choose c = 20 and n0 = 1, then:
$0 <= 5n^3+4n^2+10 <= 20n^4$ for all $n >= 1$

for n = 1	19	<= 20
for n = 2	66	<= 320
for n = 3	181	<= 1620
for n= 10	5410	<= 20*10^4

Moreover, this solution can be proved by the graph:



2-

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

Insertion sort:

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

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27] After pass1

[8, 24, 28, 51, 20, 29, 21, 17, 38, 27] After pass2

[8, 20, 24, 28, 51, 29, 21, 17, 38, 27] After pass3

[8, 20, 24, 28, 29, 51, 21, 17, 38, 27] After pass4

[8, 20, 21, 24, 28, 29, 51, 17, 38, 27] After pass5

[8, 17, 20, 21, 24, 28, 29, 51, 38, 27] After pass6

[8, 17, 20, 21, 24, 28, 29, 38, 51, 27] After pass7

[8, 17, 20, 21, 24, 27, 28, 29, 38, 51] After pass8

Buble sort:

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

[8, 24, 51, 28, 20, 29, 21, 17, 38, 27] Pass 1

[8, 24, 28, 51, 20, 29, 21, 17, 38, 27] Pass 1

[8, 24, 28, 20, 51, 29, 21, 17, 38, 27] Pass 1

[8, 24, 28, 20, 29, 51, 21, 17, 38, 27] Pass 1

[8, 24, 28, 20, 29, 21, 51, 17, 38, 27] Pass 1

[8, 24, 28, 20, 29, 21, 17, 51, 38, 27] Pass 1

[8, 24, 28, 20, 29, 21, 17, 38, 51, 27] Pass 1

[8, 24, 28, 20, 29, 21, 17, 38, 27, **51**] Pass 1

[8, 24, 20, 28, 29, 21, 17, 38, 27, **51**] Pass2

[8, 24, 20, 28, 21, 29, 17, 38, 27, **51**] Pass2

[8, 24, 20, 28, 21, 17, 29, 38, 27, **51**] Pass2

[8, 24, 20, 28, 21, 17, 29, 27, **38, 51**] Pass2

[8, 20, 24, 28, 21, 17, 29, 27, **38, 51**] Pass3

[8, 20, 24, 21, 28, 17, 29, 27, **38, 51**] Pass3

[8, 20, 24, 21, 17, 28, 29, 27, **38, 51**] Pass3

[8, 20, 24, 21, 17, 28, 27, **29, 38, 51**] Pass3

[8, 20, 21, 24, 17, 28, 27, **29, 38, 51**] Pass4

[8, 20, 21, 17, 24 , 28, 27, **29, 38, 51**] Pass4

[8, 20, 21, 17, 24 , 27, **28, 29, 38, 51**] Pass4

[8, 20, 17, 21, 24 , **27, 28, 29, 38, 51**] Pass5

[8, 17, 20, 21, 24 , **27, 28, 29, 38, 51**] Pass6

[8, 17, 20, 21, **24 , 27, 28, 29, 38, 51**] Pass6

[8, 17, 20, **21, 24 , 27, 28, 29, 38, 51**] Pass7

[8, 17, **20, 21, 24 , 27, 28, 29, 38, 51**] Pass8

[8, **17, 20, 21, 24 , 27, 28, 29, 38, 51**] Pass9

[**8, 17, 20, 21, 24 , 27, 28, 29, 38, 51**] Pass10

Question 2-

(c)

Output of the part c of question 2:

```
SelectionSort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount is 120
moveCount is 45
--------------
MergeSort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount is 46
moveCount is 128
--------------
QuickSort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
compCount is 45
moveCount is 102
--------------
RadixSort
3 5 6 7 8 9 11 12 12 14 14 17 18 19 20 21
--------------
```

(d)

Sort's comparisons with Random numbers

RANDOM NUMBERS
Analysis of Selection Sort

ArraySize	Elapsed time	compCount	moveCount
6000	36.096000	17997000	17997
10000	104.355000	49995000	29997
14000	212.748000	97993000	41997
18000	375.907000	161991000	53997
22000	479.769000	241989000	65997
26000	666.432000	337987000	77997
30000	883.299000	449985000	89997

RANDOM NUMBERS
Analysis of Merge Sort

ArraySize	Elapsed time	compCount	moveCount
6000	0.802000	67894	151616
10000	1.396000	120425	267232
14000	3.198000	175411	387232
18000	2.562000	231957	510464
22000	3.576000	289916	638464
26000	4.060000	348814	766464
30000	5.801000	408627	894464

RANDOM NUMBERS
Analysis of Quick Sort

ArraySize	Elapsed time	compCount	moveCount
6000	0.640000	91389	141605
10000	1.291000	173176	228961
14000	1.923000	267657	289339
18000	2.139000	403452	455462
22000	2.864000	534509	597852
26000	3.263000	657821	569665
30000	4.297000	804239	655265

RANDOM NUMBERS
Analysis of Radix Sort

ArraySize	Elapsed time
6000	1.476000
10000	2.395000
14000	4.253000
18000	4.260000
22000	5.627000
26000	7.286000
30000	9.049000

Sort's comparisons with ascending numbers

ASCENDING NUMBERS
Analysis of Selection Sort

ArraySize	Elapsed time	compCount	moveCount
6000	32.237000	17997000	17997
10000	99.015000	49995000	29997
14000	186.564000	97993000	41997
18000	305.942000	161991000	53997
22000	462.006000	241989000	65997
26000	660.265000	337987000	77997
30000	849.797000	449985000	89997

ASCENDING NUMBERS
Analysis of Merge Sort

ArraySize	Elapsed time	compCount	moveCount
6000	0.450000	39152	151616
10000	0.798000	69008	267232
14000	1.198000	99360	387232
18000	1.562000	130592	510464
22000	1.841000	165024	638464
26000	2.242000	197072	766464
30000	3.083000	227728	894464

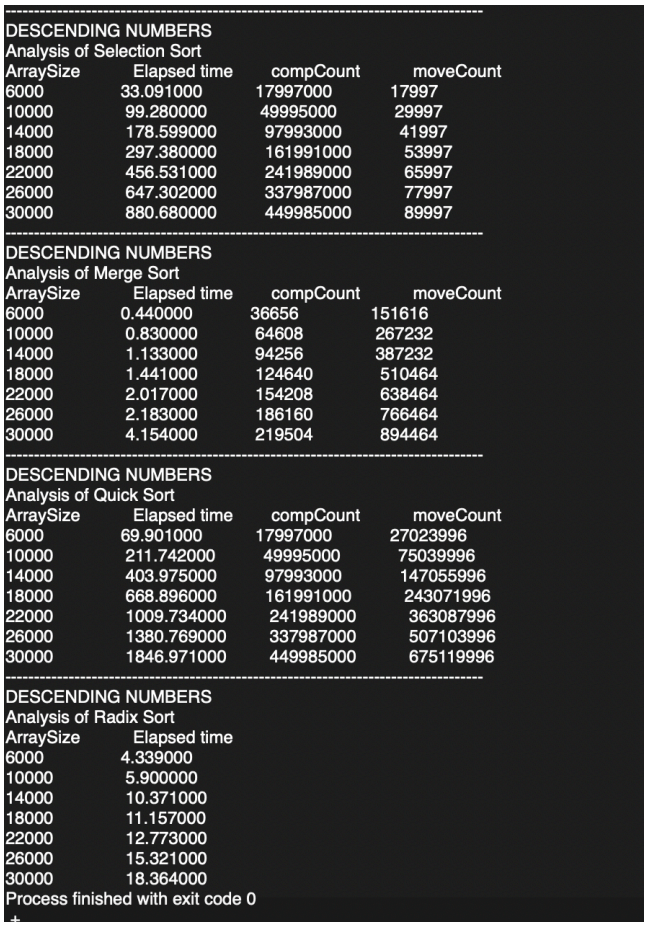
ASCENDING NUMBERS
Analysis of Quick Sort

ArraySize	Elapsed time	compCount	moveCount
6000	31.135000	17997000	23996
10000	94.363000	49995000	39996
14000	174.925000	97993000	55996
18000	292.004000	161991000	71996
22000	432.581000	241989000	87996
26000	625.897000	337987000	103996
30000	830.979000	449985000	119996

ASCENDING NUMBERS
Analysis of Radix Sort

ArraySize	Elapsed time
6000	2.169000
10000	3.912000
14000	6.394000
18000	7.717000
22000	9.730000
26000	11.605000
30000	14.257000

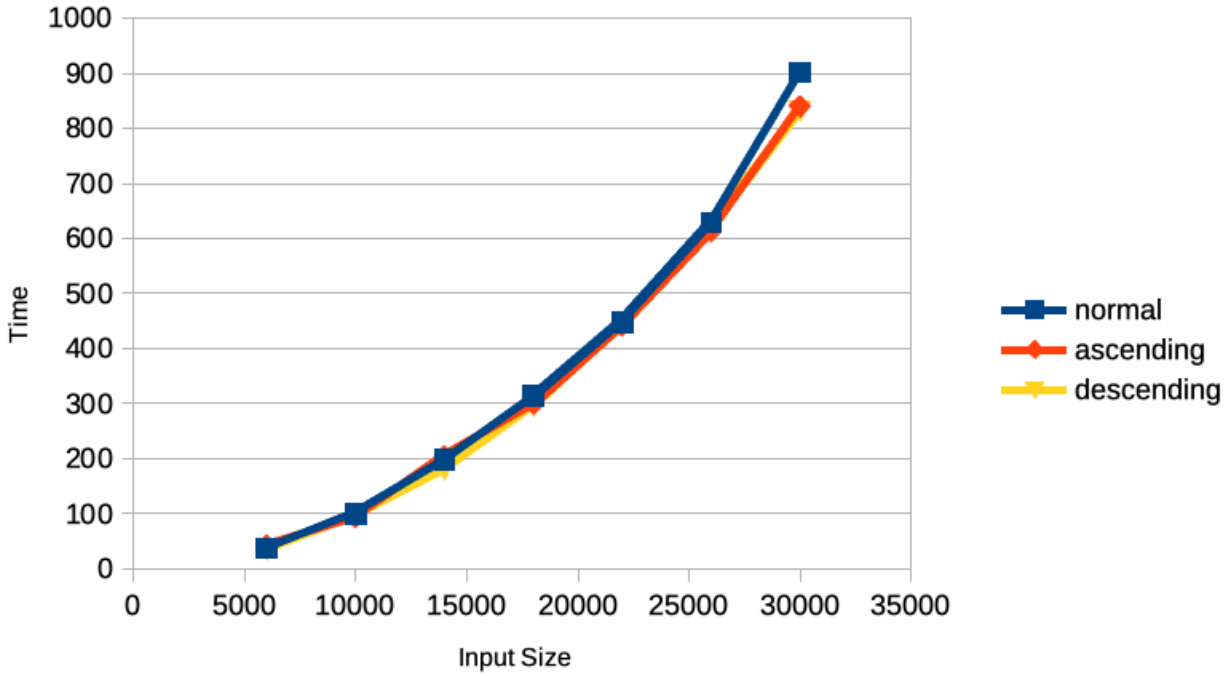
Sort's comparisons with descending numbers



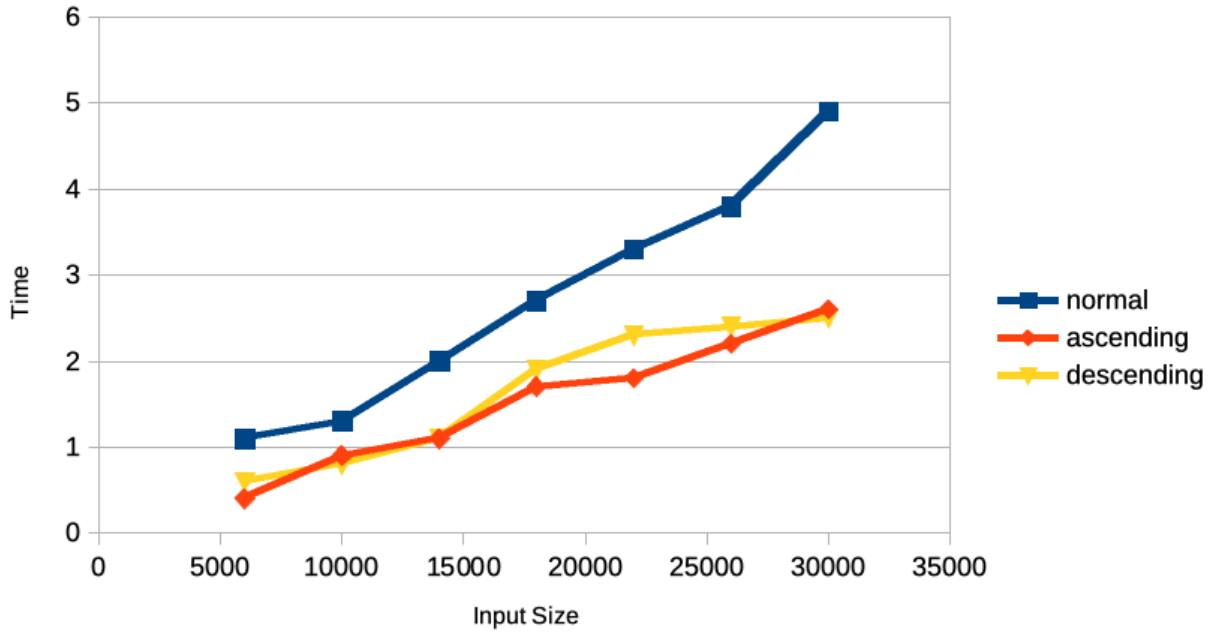
```
--------------------------------------------------------------------------------
DESCENDING NUMBERS
Analysis of Selection Sort
ArraySize       Elapsed time      compCount       moveCount
6000            33.091000         17997000        17997
10000           99.280000         49995000        29997
14000           178.599000        97993000        41997
18000           297.380000        161991000       53997
22000           456.531000        241989000       65997
26000           647.302000        337987000       77997
30000           880.680000        449985000       89997
--------------------------------------------------------------------------------
DESCENDING NUMBERS
Analysis of Merge Sort
ArraySize       Elapsed time      compCount       moveCount
6000            0.440000          36656           151616
10000           0.830000          64608           267232
14000           1.133000          94256           387232
18000           1.441000          124640          510464
22000           2.017000          154208          638464
26000           2.183000          186160          766464
30000           4.154000          219504          894464
--------------------------------------------------------------------------------
DESCENDING NUMBERS
Analysis of Quick Sort
ArraySize       Elapsed time      compCount       moveCount
6000            69.901000         17997000        27023996
10000           211.742000        49995000        75039996
14000           403.975000        97993000        147055996
18000           668.896000        161991000       243071996
22000           1009.734000       241989000       363087996
26000           1380.769000       337987000       507103996
30000           1846.971000       449985000       675119996
--------------------------------------------------------------------------------
DESCENDING NUMBERS
Analysis of Radix Sort
ArraySize       Elapsed time
6000            4.339000
10000           5.900000
14000           10.371000
18000           11.157000
22000           12.773000
26000           15.321000
30000           18.364000
Process finished with exit code 0
```

Question 3-

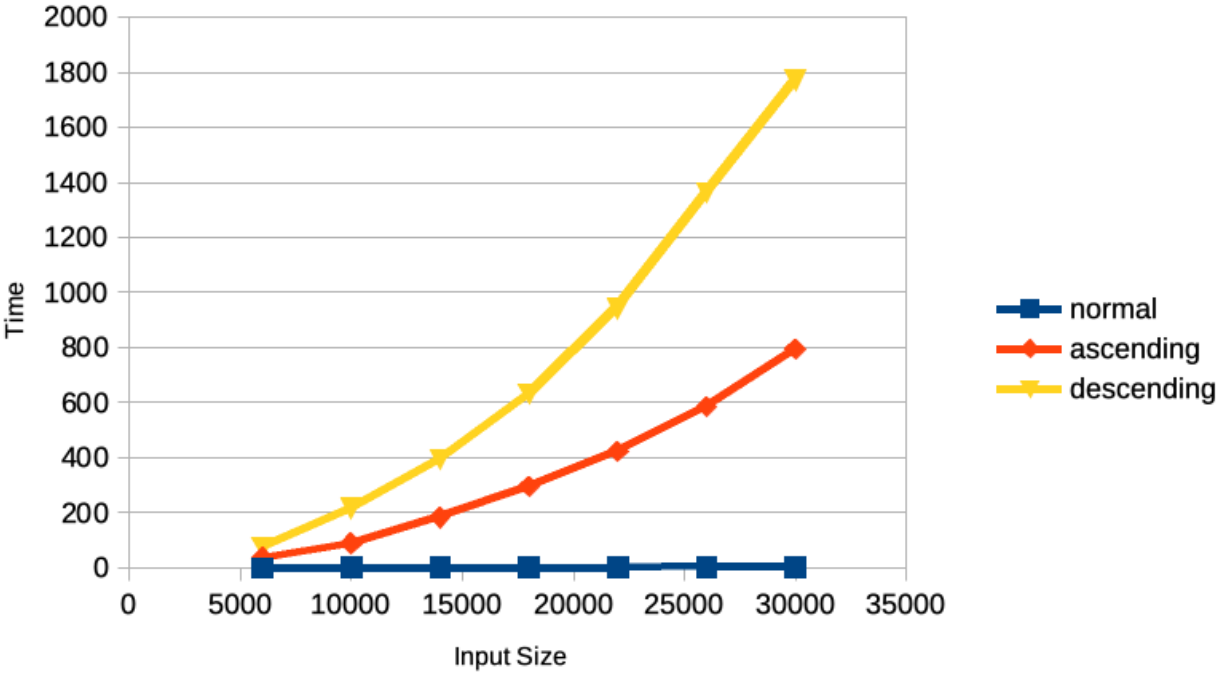
The graph of selection sorts with random, ascending and descending inputs



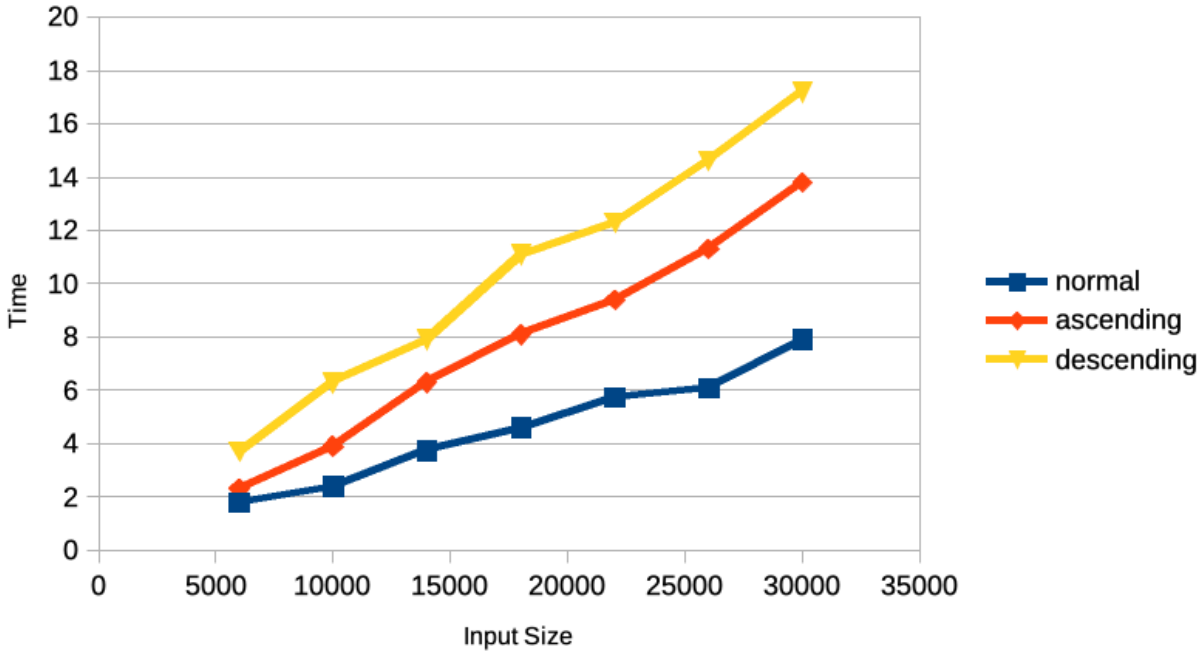
The graph of merge sorts with random, ascending and descending inputs



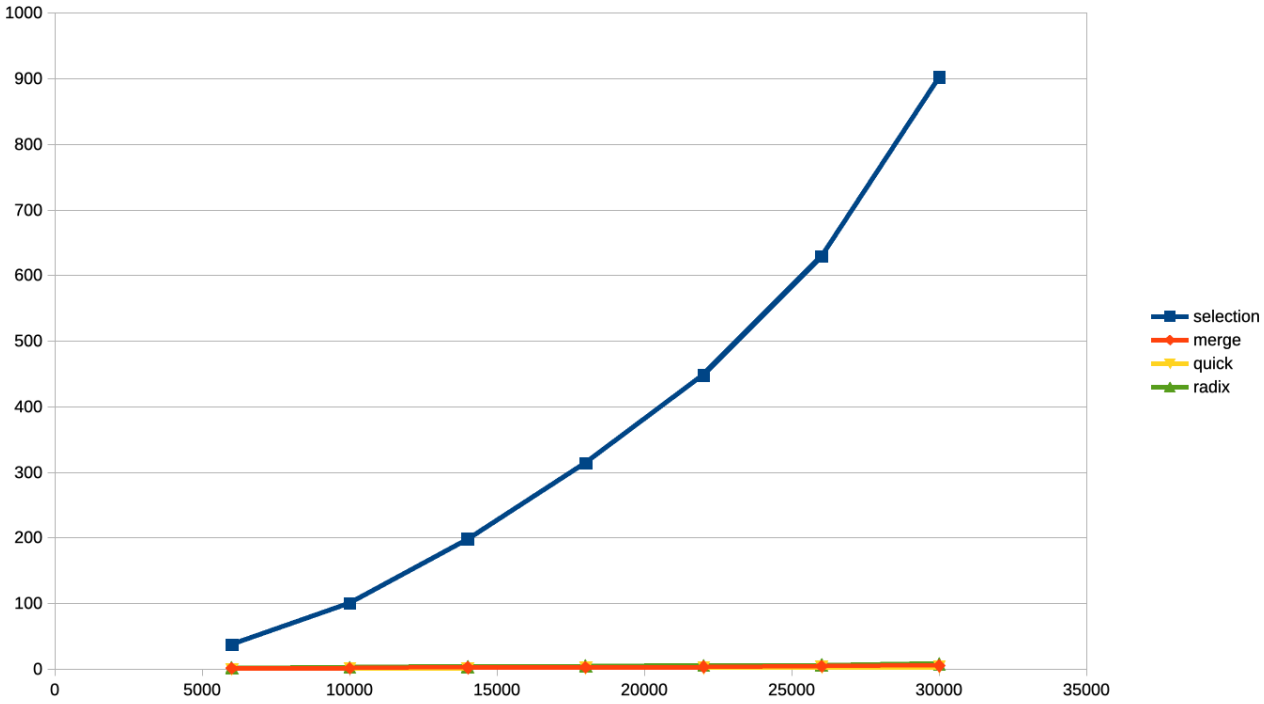
The graph of quick sorts with random, ascending and descending inputs



The graph of radix sorts with random, ascending and descending numbers



The comparison graph of selection, merge, quick, radix sorts with random numbers



In conclusion, from the experiment I understand that selection sort is the is the worst sorting algorithm in terms of time complexity with random numbers. From the selection sort graph we see that for each kinds of input(random, descending, ascending) selection sort's time complexity is $O(n^2)$. However, from the merge sort graph I understand it's time complexity is not $O(n^2)$ and O(n) it's big o between n^2 and n and also since it looks like n*logn graph it's time complexity is O(n*logn). From the quick sort graph it can be said that quick sort is a very fast sorting algorithm with random numbers. However, it is not efficient with ascending and descending numbers. Moreover, as seen in the graph it's worst case is with descending numbers and its looks like n^2 graph. Therefore, the worst case for the quick sort is $O(n^2)$ and to have worst case, all numbers must be ordered in descending form. Radix is a really fast algorithm and from the graph it looks like it works in O(n). Thus, I think that my empirical results are really similar to the theoretical ones. Also for the quick sort algorithm the difference in the time with different kinds of input is caused by the implementation of the algorithm. What I mean is that in the implementation of the quick sort we choose the first element as a pivot then we split our array to 2group which are bigger than and less than pivot. After this split we recursively repeat this and at the end, we have a sorted array. However, when having descending number the pivot becomes the largest number and using this algorithm is becoming inefficient.