# CS 201, Summer 2015

# Homework Assignment 2

***Analysis Of Different Algorithms In Terms Of Time Complexity***

**Name:** Gülsüm

**Surname:** Güdükbay

**Student ID:** 21401148

Gülsüm Güdükbay
21401148

## 1. Specifications Of The Computer

The experimental execution time data of this homework assignment was obtained in a MacBook Pro, OS X Yosemite, Version 10.10.2, with a 2.2 GHz Intel Core i7 processor, 4 GB 1333 MHz DDR3 memory and AMD Radeon HD 6750 M 1024 MB graphics.

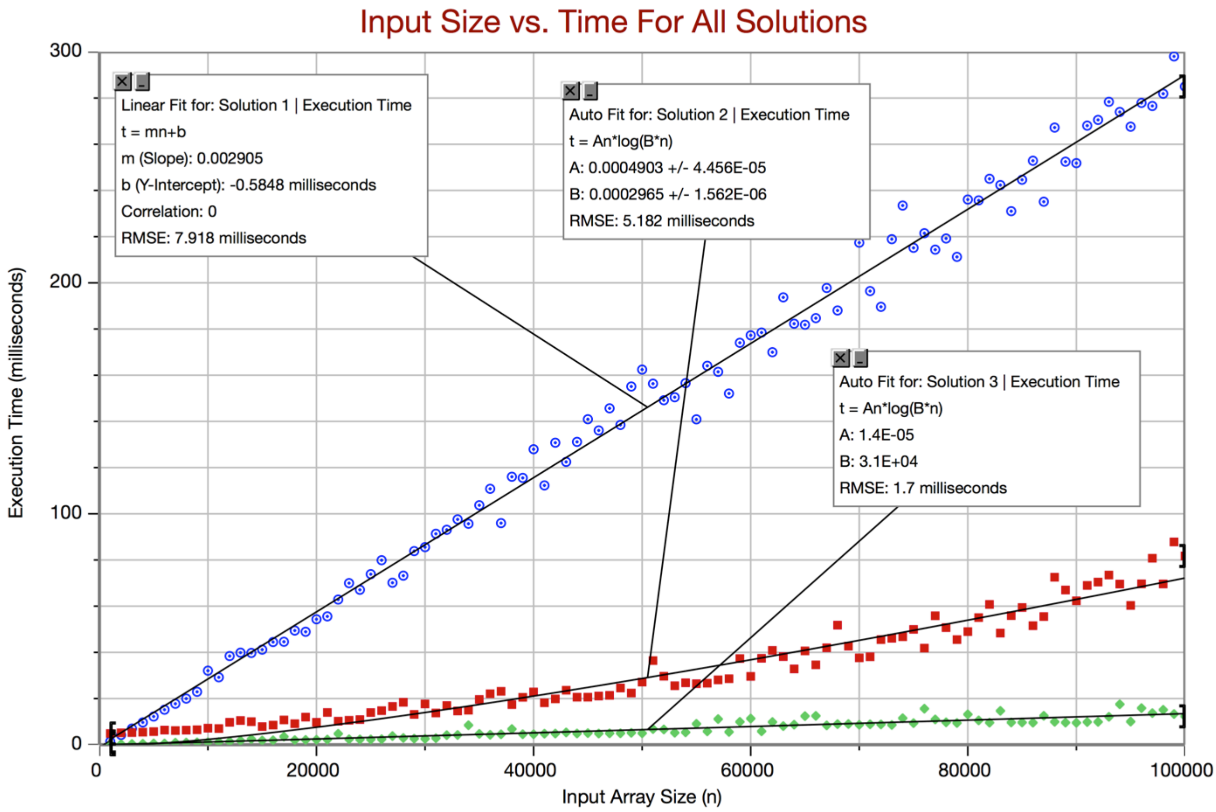
## 2. Data Collection For All Algorithms

| | Solution 1 | | Solution 2 | | Solution 3 | |
|---|---|---|---|---|---|---|
| | Input Array Size (n) | Execution Time (milliseconds) | Input Array Size (n) | Execution Time (milliseconds) | Input Array Size (n) | Execution Time (milliseconds) |
| 1 | 1000 | 1.381 | 1000 | 4.861 | 1000 | 0.174 |
| 2 | 2000 | 4.207 | 2000 | 5.17 | 2000 | 0.235 |
| 3 | 3000 | 6.987 | 3000 | 5.254 | 3000 | 0.346 |
| 4 | 4000 | 9.66 | 4000 | 5.424 | 4000 | 0.439 |
| 5 | 5000 | 12.219 | 5000 | 5.708 | 5000 | 0.551 |
| 6 | 6000 | 15.171 | 6000 | 6.364 | 6000 | 0.542 |
| 7 | 7000 | 17.744 | 7000 | 6.162 | 7000 | 0.893 |
| 8 | 8000 | 19.971 | 8000 | 6.309 | 8000 | 0.935 |
| 9 | 9000 | 22.911 | 9000 | 6.492 | 9000 | 1.017 |
| 10 | 10000 | 32.064 | 10000 | 7.16 | 10000 | 1.067 |
| 11 | 11000 | 29.187 | 11000 | 6.975 | 11000 | 1.174 |
| 12 | 12000 | 38.397 | 12000 | 9.714 | 12000 | 1.365 |
| 13 | 13000 | 39.867 | 13000 | 10.474 | 13000 | 1.747 |
| 14 | 14000 | 39.739 | 14000 | 9.972 | 14000 | 2.432 |
| 15 | 15000 | 41.167 | 15000 | 7.872 | 15000 | 1.849 |
| 16 | 16000 | 44.463 | 16000 | 8.479 | 16000 | 1.92 |
| 17 | 17000 | 44.583 | 17000 | 10.735 | 17000 | 3.505 |
| 18 | 18000 | 49.438 | 18000 | 9.117 | 18000 | 2.002 |
| 19 | 19000 | 48.908 | 19000 | 11.97 | 19000 | 2.088 |
| 20 | 20000 | 54.325 | 20000 | 9.687 | 20000 | 2.157 |
| 21 | 21000 | 55.564 | 21000 | 13.982 | 21000 | 2.241 |
| 22 | 22000 | 62.855 | 22000 | 10.168 | 22000 | 4.755 |
| 23 | 23000 | 69.969 | 23000 | 10.603 | 23000 | 2.379 |
| 24 | 24000 | 67.089 | 24000 | 10.887 | 24000 | 2.292 |
| 25 | 25000 | 73.848 | 25000 | 13.932 | 25000 | 2.305 |
| 26 | 26000 | 79.89 | 26000 | 14.761 | 26000 | 2.4 |
| 27 | 27000 | 70.141 | 27000 | 16.567 | 27000 | 3.748 |
| 28 | 28000 | 73.205 | 28000 | 18.385 | 28000 | 2.933 |
| 29 | 29000 | 83.836 | 29000 | 13.138 | 29000 | 2.66 |
| 30 | 30000 | 85.542 | 30000 | 17.69 | 30000 | 2.583 |
| 31 | 31000 | 91.344 | 31000 | 13.72 | 31000 | 2.844 |
| 32 | 32000 | 93.072 | 32000 | 16.965 | 32000 | 4.138 |
| 33 | 33000 | 97.573 | 33000 | 14.628 | 33000 | 4.212 |
| 34 | 34000 | 95.682 | 34000 | 14.951 | 34000 | 8.445 |
| 35 | 35000 | 103.692 | 35000 | 19.607 | 35000 | 4.689 |
| 36 | 36000 | 110.809 | 36000 | 22.076 | 36000 | 4.297 |
| 37 | 37000 | 95.953 | 37000 | 23.134 | 37000 | 4.519 |
| 38 | 38000 | 116.032 | 38000 | 17.396 | 38000 | 6.785 |
| 39 | 39000 | 115.532 | 39000 | 20.527 | 39000 | 4.544 |
| 40 | 40000 | 127.939 | 40000 | 22.836 | 40000 | 4.55 |
| 41 | 41000 | 112.281 | 41000 | 18.196 | 41000 | 4.779 |
| 42 | 42000 | 130.767 | 42000 | 19.816 | 42000 | 4.837 |
| 43 | 43000 | 122.463 | 43000 | 23.616 | 43000 | 5.323 |
| 44 | 44000 | 131.136 | 44000 | 20.598 | 44000 | 4.871 |
| 45 | 45000 | 140.929 | 45000 | 20.599 | 45000 | 5.027 |

*Table 1 – Data Collection of the execution times for all algorithms*

| | Solution 1 | | Solution 2 | | Solution 3 | |
|---|---|---|---|---|---|---|
| | Input Array Size (n) | Execution Time (milliseconds) | Input Array Size (n) | Execution Time (milliseconds) | Input Array Size (n) | Execution Time (milliseconds) |
| 46 | 46000 | 136.088 | 46000 | 21.105 | 46000 | 4.848 |
| 47 | 47000 | 145.657 | 47000 | 21.378 | 47000 | 4.934 |
| 48 | 48000 | 138.473 | 48000 | 24.563 | 48000 | 4.962 |
| 49 | 49000 | 155.115 | 49000 | 22.372 | 49000 | 5.154 |
| 50 | 50000 | 162.428 | 50000 | 27.13 | 50000 | 4.975 |
| 51 | 51000 | 156.335 | 51000 | 36.456 | 51000 | 6.794 |
| 52 | 52000 | 149.165 | 52000 | 29.648 | 52000 | 6.616 |
| 53 | 53000 | 150.457 | 53000 | 25.53 | 53000 | 5.199 |
| 54 | 54000 | 156.569 | 54000 | 26.841 | 54000 | 5.368 |
| 55 | 55000 | 140.883 | 55000 | 26.413 | 55000 | 9.048 |
| 56 | 56000 | 164.086 | 56000 | 26.659 | 56000 | 5.793 |
| 57 | 57000 | 161.463 | 57000 | 28.075 | 57000 | 11.116 |
| 58 | 58000 | 152.076 | 58000 | 28.57 | 58000 | 5.533 |
| 59 | 59000 | 174.038 | 59000 | 37.314 | 59000 | 9.907 |
| 60 | 60000 | 177.279 | 60000 | 29.591 | 60000 | 11.335 |
| 61 | 61000 | 178.475 | 61000 | 37.395 | 61000 | 5.821 |
| 62 | 62000 | 169.96 | 62000 | 40.871 | 62000 | 9.828 |
| 63 | 63000 | 193.704 | 63000 | 38.163 | 63000 | 8.072 |
| 64 | 64000 | 182.304 | 64000 | 32.821 | 64000 | 8.689 |
| 65 | 65000 | 181.837 | 65000 | 40.601 | 65000 | 12.39 |
| 66 | 66000 | 184.708 | 66000 | 34.607 | 66000 | 12.495 |
| 67 | 67000 | 197.768 | 67000 | 41.969 | 67000 | 8.376 |
| 68 | 68000 | 188.03 | 68000 | 51.765 | 68000 | 8.924 |
| 69 | 69000 | 227.904 | 69000 | 42.7 | 69000 | 8.967 |
| 70 | 70000 | 217.374 | 70000 | 37.609 | 70000 | 8.634 |
| 71 | 71000 | 196.426 | 71000 | 38.063 | 71000 | 8.856 |
| 72 | 72000 | 189.608 | 72000 | 45.511 | 72000 | 8.803 |
| 73 | 73000 | 218.899 | 73000 | 46.106 | 73000 | 8.569 |
| 74 | 74000 | 233.461 | 74000 | 46.721 | 74000 | 11.49 |
| 75 | 75000 | 215.148 | 75000 | 49.921 | 75000 | 9.359 |
| 76 | 76000 | 221.483 | 76000 | 41.812 | 76000 | 15.606 |
| 77 | 77000 | 214.355 | 77000 | 55.819 | 77000 | 10.947 |
| 78 | 78000 | 219.197 | 78000 | 50.723 | 78000 | 9.71 |
| 79 | 79000 | 211.233 | 79000 | 45.514 | 79000 | 9.421 |
| 80 | 80000 | 236.062 | 80000 | 48.992 | 80000 | 13.141 |
| 81 | 81000 | 235.694 | 81000 | 55.056 | 81000 | 10.533 |
| 82 | 82000 | 245.048 | 82000 | 60.734 | 82000 | 9.559 |
| 83 | 83000 | 242.328 | 83000 | 48.354 | 83000 | 14.668 |
| 84 | 84000 | 231.018 | 84000 | 55.897 | 84000 | 9.647 |
| 85 | 85000 | 244.623 | 85000 | 59.402 | 85000 | 9.64 |
| 86 | 86000 | 252.862 | 86000 | 51.553 | 86000 | 9.556 |
| 87 | 87000 | 235.115 | 87000 | 55.476 | 87000 | 12.524 |
| 88 | 88000 | 267.228 | 88000 | 72.532 | 88000 | 9.927 |
| 89 | 89000 | 252.461 | 89000 | 66.98 | 89000 | 9.587 |
| 90 | 90000 | 251.825 | 90000 | 62.315 | 90000 | 9.507 |
| 91 | 91000 | 268.058 | 91000 | 68.98 | 91000 | 9.822 |
| 92 | 92000 | 270.55 | 92000 | 70.44 | 92000 | 9.878 |
| 93 | 93000 | 278.38 | 93000 | 73.483 | 93000 | 12.085 |
| 94 | 94000 | 273.977 | 94000 | 69.549 | 94000 | 17.519 |
| 95 | 95000 | 267.612 | 95000 | 60.335 | 95000 | 10.007 |
| 96 | 96000 | 277.888 | 96000 | 69.658 | 96000 | 15.853 |
| 97 | 97000 | 276.563 | 97000 | 80.75 | 97000 | 13.683 |
| 98 | 98000 | 281.888 | 98000 | 69.631 | 98000 | 15.124 |
| 99 | 99000 | 298.034 | 99000 | 87.795 | 99000 | 13.29 |
| 100 | 100000 | 285.008 | 100000 | 81.718 | 100000 | 12.293 |

*Table 1(continued) – Data Collection of the execution times for all algorithms*




*Graph 1 – Graph of the comparison of the execution times for all three algorithms, for different input array sizes*

The graph above shows that different algorithms can have different execution times and growth rates of execution times for different input array sizes, for the problem of selecting the k largest numbers in an array of n integers. For Solution 1, the best line of fit was found to be a linear one, since the rate of change of execution times for different input sizes is constant, so the time complexity can be shown by the function O(n).

For Solution 2, the growth rate of the execution times were clearly fitted by the time complexity O(nlogn), since the growth rate isn't constant for increasing n and it increases faster when we move toward the maximum n specified.

The graph of Solution 3 was actually not clear as the other solutions, since the best fit line is both fitting with a linear fit and the function t = nlogn, where t is the execution time. However, if we zoom in, and if we compare the linear fit with the nlogn curve of fit, we can realize that the curve nlogn is much more closer to all the data points obtained. Therefore the time complexity of Solution 3 is found to be O(nlogn).

# 3. Experimental vs. Theoretic Graphs

## i. *Solution 1*

| Input Size (n) | Experimental Execution Time (ms) | Theoretic Execution Time (ms) |
|---|---|---|
| 1000 | 1.381 | 3.206 |
| 5000 | 12.219 | 16.032 |
| 10000 | 32.064 | 32.064 |
| 50000 | 162.428 | 160.320 |
| 100000 | 285.008 | 320.640 |

*Table 2 – Table of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 1*


*Graph 2 – Graph of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 1*

The graph and the table above shows how does increasing the input array's size effects the execution time for the algorithm of Solution 1 and does a comparison between the theoretic and the experimental execution times.

Since the swap method that is called in Solution 1 has a constant time complexity O(1) and the for loop inside Solution 1 traverses all the elements in the input array at least once as the worst case, indicating n iterations, the overall theoretic time complexity of the first algorithm is O(n). Other assignments and declarations are not considered, since they take constant unit time O(1). Therefore, to calculate the theoretic execution time, n = 10000 was taken as a sample. By doing cross multiplication, the other theoretic execution times were calculated (e.g. for n = 1000, the theoretic execution time was calculated as 32.064 / 10).

As the input array's size increases, the experimental execution time also increases linearly, showing that the first algorithm's time complexity is O(kn), where k is the output size and n is the input size. k, here, is treated as a constant of gradient.

If we are to compare the theoretic and the experimental times of execution of the first algorithm, the graph of experimental times had a smaller gradient, showing that the theoretic execution times were less as input size increased. This indicates a faster execution time because of the random initialization of the array, since the indices of some maximum values were closer to the beginning of input array, so a fewer amount of iterations were performed.

### *ii. Solution 2*

| Input Size (n) | Experimental Execution Time (ms) | Theoretic Execution Time (ms) |
|---|---|---|
| 1000 | 4.861 | 0.216 |
| 5000 | 5.708 | 2.502 |
| 10000 | 7.16 | 7.16 |
| 50000 | 27.13 | 83.125 |
| 100000 | 81.718 | 166.35 |

*Table 3 – Table of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 2*


*Graph 3 – Graph of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 2*

The graph and the table above shows how does increasing the input array's size effects the execution time for the algorithm of Solution 2 and does a comparison between the theoretic and the experimental execution times.

Since the quickSort algorithm is called in Solution 2 and quickSort is a recursive sorting algorithm with O(n$\log_2$n) time complexity, considering that the other assignments and declarations are not necessary to take into consideration in the theoretical time complexity calculations, the overall time complexity of Solution 2 is the time complexity of quickSort, which is O(n$\log_2$n). Therefore, to calculate the theoretic execution time, n = 10000 was taken as a sample. By doing a ratio calculation, the other theoretic execution times were calculated (e.g. for n = 1000, the theoretical execution time was calculated as 7.17 / (10*$\log_2$(10))).

As the input array's size increases, the experimental execution time also increases but in the pattern of the function t = nlogn, showing that the second algorithm's time complexity is O(nlogn), where n is the input size.

If we are to compare the theoretic and the experimental times of execution of the second algorithm, the rate of change of experimental times was less, showing that the theoretic execution times increased faster. This indicates better execution time because of the random initialization of the array, since the indices of some maximum values were closer to the end while sorting the input array.

### *iii.* ***<u>Solution 3</u>***

| **Input Size (n)** | **Experimental Execution Time (ms)** | **Theoretic Execution Time (ms)** |
|---|---|---|
| 1000 | 0.174 | 0.032 |
| 5000 | 0.551 | 0.373 |
| 10000 | 1.067 | 1.067 |
| 50000 | 4.975 | 12.329 |
| 100000 | 12.293 | 35.279 |

*Table 4 – Table of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 3*


*Graph 4 – Graph of the Comparison of Theoretic Execution Times and Experimental Execution Times for Solution 3*

The graph and the table above shows how does increasing the input array's size effects the execution time for the algorithm of Solution 3 and does a comparison between the theoretic and the experimental execution times.

The quickSort algorithm is called in select method and quickSort is a recursive sorting algorithm with $O(n\log_2 n)$ time complexity. However, rather than sorting the whole array, if we sort a portion of it, it is much more efficient. This is less efficient than simple selecting, since it takes O(n + klogk) (where n is the input array size and k is the output array size), but it is, as stated before, much more efficient than sorting the entire array. It also partitions the array as sorted and unsorted with the partition method. Therefore the overall mode time complexity of the third solution is O(nlogn), however it would be better to keep in consideration that the worst case time complexity(which wasn't seen so frequently) is O(n). Therefore, to calculate the theoretic execution time, n = 10000 was taken as a sample. By doing a ratio calculation, the other theoretic execution times were calculated (e.g. for n = 1000, the theoretical execution time was calculated as 1.067 / $(10*\log_2(10))$). The average case time complexity function was used, since worst case is seen very rarely with this algorithm.

As the input array's size increases, the experimental execution time also increases but in the pattern of the function t = nlogn, showing that the third algorithm's time complexity is O(nlogn), where n is the input size.

If we are to compare the theoretic and the experimental times of execution of the third algorithm, the rate of change of experimental times was less, showing that the theoretic execution times increased faster. This indicates better execution time because of the random initialization of the array, since the indices of some maximum values were closer to the end while sorting the input array and that mostly, the worst case O(n) wasn't reached during the selection by sorting process.

# 4. Conclusion

Finally, it would be beneficial to state that even though the first and second algorithms were easier and shorter to write and understand, the third algorithm was the quickest algorithm based on the experimental execution time for different input sizes. Even though the third algorithm's worst case time complexity was O(n), generally, the average case time complexity was observed, which is O(nlogn). Besides the time complexity, the growth rate of the graph of the execution times for different input times of Solution 3 was very small compared to the other two solutions. If we are to compare the first and the second solutions, the second algorithm used quickSort, which is a recursive sorting algorithm with time complexity O(nlogn), which dominated the time complexity of the whole algorithm. So considering the fact that the time complexity of the first solution was O(n), which is linear, the growth rate of the first solution was much larger than the second solution, so the order of the algorithms, starting with the worst time complexity is Solution 1, Solution 2, then Solution 3, being the best algorithm.