

# Homework 2: Sorting algorithms

## Introduction

---

In order to compare the different sorting algorithms, I decided to start by comparing their time complexity and space complexity. This will be a theoretical comparison. After this comparison, I will go to comparing the times of execution tested by my Python program.

## 1. Theoretical comparison

---

### Time complexity

I have decided to look into the following set of algorithms: Bubble, Selection, Insertion, Merge, Heap, Quick, Radix and Count. These are the most commonly used sort algorithms. Time complexity can be calculated for each algorithm from its pseudocode or code. It generally represents the time the algorithm spends looping through the data. The time complexity in the worst case is the more relevant one but the best and average cases should not be neglected as it depends on the data used.

### Space complexity

Space complexity should also be looked at and not only time complexity. Like time complexity, space complexity can be calculated from the number of variables used in the pseudocode or code. It generally represents the size of memory used by the algorithm.

### Results

By calculating the time and space complexities myself and by verifying results found online for every sorting algorithm from different sources. These are the results found.

Sort algorithm	Time complexity			Space complexity
	Best case	Average case	Worst case	Worst case
Bubble Sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(1)$
Insertion Sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
Merge Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$
Heap Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(1)$
Quick Sort	$O(N \log N)$	$O(N \log N)$	$O(N^2)$	$O(\log N)$
Radix Sort	$O(N k)$	$O(N k)$	$O(N k)$	$O(N + k)$
Count Sort	$O(N + k)$	$O(N + k)$	$O(N + k)$	$O(k)$

Reference: <https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/>

## Comments on the results

According to the results found, we can come to the conclusion that there is no algorithm that is clearly better than the others. It will depend on which is the priority: time or space. However, we could say that for a general use case it is wise to pick an algorithm that is consistent with different sizes of data and balances both time and space for example: Merge, Heap and Quick.

## 2. Practical comparison

I have compared the time taken by each algorithm to compare the same list. I have tested the 8 algorithms with lists of different sizes. I have used random lists of numbers from 0 to 100. Each test is done in 5 runs. I compare the averages of the 5 runs.

The code used for every sorting algorithm was heavily inspired by <https://www.geeksforgeeks.org/sorting-algorithms/>  
 However, every sorting function has been heavily tested before starting the comparisons.

## Results

	List size				
Sort algorithm	10	100	1000	5000	10000
Bubble Sort	1.3828 277587 890625 e-05	0.0010 130405 426025 39	0.0694 253444 671630 9	1.5125 749111 175537	5.8484 897613 52539
Selection Sort	1.4781 951904 296875 e-05	0.0004 901885 986328 125	0.0279 650688 171386 72	0.6332 373619 07959	2.4196 562767 028814 3
Insertion Sort	1.1682 510375 976562 e-05	0.0004 606246 948242 1875	0.0279 350280 761718 75	0.6689 014434 814453	2.6201 961040 496826
Merge Sort	3.6478 042602 53906e -05	0.0004 038810 729980 469	0.0026 183128 356933 594	0.0168 693065 643310 55	2.6201 961040 496826
Heap Sort	2.6464 462280 273438 e-05	0.0002 923011 779785 156	0.0031 023025 512695 312	0.0196 471214 294433 6	0.0339 403152 465820 3
Quick Sort	1.93119 049072 26562e -05	0.0001 585483 551025 3906	0.0014 195442 199707 031	0.01132 488250 732421 9	0.0428 593158 721923 8

Radix Sort	0.0064 423084 259033 2	0.0360 453128 814697 3	0.3156 902790 069586 345	1.6244 516372 680664	3.0173 208713 531494
Count Sort	1.3351 440429 6875e- 05	2.5987 625122 070312 e-05	0.0001 795291 900634 7656	0.0007 667541 503906 25	0.0015 606880 187988 281

## Comments on the results

- We can see that Heap and Quick are constantly fast with different list sizes.
- Bubble, Selection, and Insertion become slow when the list sizes increase.
- Radix is very slow because I used integers between 0 and 100. This means that  $k=100$ . When  $n=100$  :  $O(n*k)=O(100*100)=O(n^2)$  etc.
- Count sort was the fastest. However, this because it had the unfair advantage of knowing the biggest number in the list before the sort.

## Conclusion

With both theoretical and practical comparisons, we can make the following conclusions:

- There isn't one sort algorithm that is the best in all conditions.
- The choice of the best sort algorithm depends on the data.
- Radix is very slow for a large spread of data ex:  $[0, 100]$ .
- If you already know the max of your list Count is the fastest.
- Bubble, Selection, and Insertion are only good for very small size data.
- Heap and Quick are fast in most cases.

You should also consider whether you value space or time.

- If you value space complexity you will prefer Heap.
- If you value time complexity you will prefer Quick.