Jan. 2020 - Bari Abbassi

Comparing several 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.

	Time complexity			Space complexity
Sort algorithm	Best case	Average case	Worst case	Worst case
Bubble Sort	O(N)	O(N2)	O(N2)	O(1)
Selection Sort	O(N2)	O(N2)	O(N2)	O(1)
Insertion Sort	O(N)	O(N2)	O(N2)	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(N2)	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	1.3828277587890625e-05	0.001013040542602539	0.06942534446716309	1.5125749111175537	5.848489761352539
Selection Sort	1.4781951904296875e-05	0.0004901885986328125	0.027965068817138672	0.633237361907959	2.41965627670288143
Insertion Sort	1.1682510375976562e-05	0.0004606246948242 1875	0.027935028076171875	0.6689014434814453	2.62019610404968 26
Merge Sort	3.647804260253906e-05	0.0004038810729980469	0.002618312835693359 4	0.016869306564331055	2.62019610404968 26
Heap Sort	2.646446228027343 8e-05	0.0002923011779785156	0.0031023025512695312	0.019647121429443 36	0.033940315246582 03
Quick Sort	1.9311904907226562e-05	0.0001585483551025 3906	0.0014195442199707031	0.011324882507324219	0.042859315872192 38

Radix Sort	0.0064423084259033 2	0.0360453128814697 3	0.3156902790069586345	1.6244516372680664	3.0173208713531494
Count Sort	1.3351440429 6875e-05	2.5987625122070312e-05	0.0001795291900634 7656	0.0007667541503906 25	0.0015606880187988281

Comments on the results

- We can see that Heap and Quick are constantly fast with differant 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(n2) 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.