Name: Sebastian J Zapart Date: 09/2023

Compulsory 2 - Algorithms and Data Structures

Git Hub link: https://github.com/Shrimpy02/Algorithms-and-DataStructures_Compulsory2

Complexity explanations:

Time complexity defines the number of times a particular instruction is carried out. It is always used in comparison of the worst, best and average cases. In order to calculate runtime, you must multiply this number with the time your computer uses for each of the instructions.

Space complexity defines the total memory space required by the instruction for its execution. Both time and space complexity are calculated as a function of size(n). Ω is the lower bound of runtime, O is the upper bound of runtime and θ is the average between Ω and O.

Beneath is generalized math that loses accuracy the more elements there are.

Bubble Sort

Bubble sorting is an iterative sorting algorithm. It works by comparing the two first indexes of an array or the like, and swapping their places if the first index is larger than the second index. It then iterates for each element in the array. After it has gone through the entire array you check if it has been sorted, if not then you run the sorting algorithm again. This is a very efficient method for small data sets but becomes exponentially worse the more elements there are.

Time complexity

Time complexity of the bubble sort algorithm is as follows:

Best case: Ω(n)

Average case: θ(n^2)

Worst case: O(n^2)

For sorting time I will calculate with the average case since the vector is random and with the average runtime of 0.0002 milliseconds for one element to be sorted.

10: θ(n^2) * 0.0002 = θ (10^2)*0.0002 = 100*0.0002 = 0.02 milliseconds

100: θ(n^2) * 0.0002 = θ (100^2)*0.0002 = 10000*0.0002 = 2 milliseconds

1000: θ(n^2) * 0.0002 = θ (1000^2)*0.0002 = 1000000*0.0002 = 200 milliseconds

10000: θ(n^2) * 0.0002 = θ (10000^2)*0.0002 = 100000000*0.0002 = 20,000 milliseconds = 20 seconds

Bubble sort is at its core unoptimized for large datasets; therefore, the system runs out of memory with data sets larger than 3000.

Space complexity

Space complexity of the bubble sort algorithm is as follows:

O(1) – since there is only one vector passed into the function and it does not create additional ones during runtime, however the size of the vector does come into a count as memory but not regarding space complexity.

Merge Sort

Merge sorting is a recursive sorting algorithm. It works by halving an array or the like into sub arrays until there is only one element per array. It then recursively merges these arrays back together in order by comparing the elements size at the same time. This algorithm is verry time efficient but uses a lot of space, account on the many sub arrays that needs to be produced.

Time complexity

Time complexity of the merge sort algorithm is as follows:

Best case: Ω(n log(n))

Average case: θ(n log(n))

Worst case: O(n log(n))

For sorting time I will calculate with the average case since the vector is random and with the average runtime of 0.0018 milliseconds for one element to be sorted.

10: θ(n log(n))* 0. 0018 = 10*0. 0018 = 0.018 milliseconds

100: θ(n log(n))* 0. 0018 = 200*0. 0018 = 0.36milliseconds

1000: θ(n log(n))* 0. 0018 = 3000*0. 0018 = 5.4 milliseconds

10000: θ(n log(n))* 0. 0018 = 40000*0. 0018 = 72 milliseconds

Space complexity

Space complexity of the merge sort algorithm is as follows:

O(n)– since the vector is recursive and iterates for each element it takes a proportional amount of memory to the elements it is sorting.

Quick Sort

Quick sort is also a recursive sorting algorithm. It works by defining a pivot point in an array or the like, then comparing values placing smaller values than the pivot to the left of it and higher values to the right of it. It works well with larger data sets and is space efficient, however depending on where the pivot is placed it has a heigh time complexity.

Time complexity

Time complexity of the quick sort algorithm is as follows:

Best case: Ω(n log(n))

Average case: θ(n log(n))

Worst case: O(n^2)

For sorting time I will calculate with the average case since the vector is random and with the average runtime of 0.0018 milliseconds for one element to be sorted.

10: θ(n log(n))* 0. 0018 = 10*0. 0018 = 0.018 milliseconds

100: θ(n log(n))* 0. 0018 = 200*0. 0018 = 0.36milliseconds

1000: θ(n log(n))* 0. 0018 = 3000*0. 0018 = 5.4 milliseconds

10000: θ(n log(n))* 0. 0018 = 40000*0. 0018 = 72 milliseconds

Space complexity

Space complexity of the quick sort algorithm is as follows:

O(n) since the function is recursive and iterates for each element it takes a proportional amount of memory to the elements it is sorting.

Reflection

Bubble sort is an optimal algorithm for small data structures but as seen the runtime grows exponentially with the number of elements. Comparing and sorting a data sets under 100 elements is the most optimal since it has such a small space complexity in addition to the short runtime. This would be my choice if it comes to smaller datasets.

However, with large datasets both merge sort and quicksort work well. The average runtime of the quicksort algorithm is the same as the merge sorts runtime, however the worst case is much more severe. Placing the pivot is random and can lead to a time complexity comparable to bubble sort. In my opinion merge sort is best suited for large datasets since it does not rely on any random chance during runtime. It divides arrays equally each time and has the same prosses for rebuilding the arrays afterwards. It is stable and efficient, which is why that would be my preference.