Paul Davis

CSCI 311

Project 1

02/28/2022

1. 1. Bubble Sort
      1. Worst: O(n^2)

This is because in a reverse sorted list, every element needs to be visited and moved, which is accomplished through a doubly nested loop.

* + 1. Average: O(n^2)

On average the behavior would be n^2 at scale. Although fewer than n^2 loops would occur on average, asymptotically the reduction is not meaningful.

* + 1. Best: O(n)

This is because a sorted list, no swaps will occur, and every element will only be visited once.

* 1. Insertion Sort
     1. Worst: O(n^2)

In the worst case, every element needs to be moved, which requires a double nested structure.

* + 1. Average: O(n^2):On average the behavior would be n^2 at scale. Although fewer than n^2 loops would occur on average, asymptotically the reduction is not meaningful.
    2. Best: O(n)

This is because a sorted list, no swaps will occur, and every element will only be visited once.

* 1. Selection Sort
     1. Worst: O(n^2)

This algorithm always visits every node in every instance of both loops.

* + 1. Average: O(n^2)

This algorithm always visits every node in every instance of both loops.

* + 1. Best: O(n^2)

This is because even in a sorted list, there is not early exit in the algorithm so every instance of the double nested loop will be executed.

* 1. Quick Sort
     1. Worst: O(n^2)

This occurs when a list is already sorted, either ascending or descending. That is because the pivot always only works on half the recursion, and without the recursion working for us, performance becomes n^2.

* + 1. Average: O(nlog(n))

Average case, trend toward best case, but this depends a lot on the nature of the data. All else equal, pivots typically will not be at one extreme of the data set.

* + 1. Best: O(nlog(n))

Best case occurs when we have a good pivot the is in the middle of the data set numerically, or close to it. This allows both sides of the recursion to function optimally.

1. 1. Bubble Sort
      1. Worst: Reverse Sort
      2. Average: Random
      3. Best: Sorted
   2. Insertion Sort
      1. Worst: Reverse Sort
      2. Average: Random
      3. Best: Sorted
   3. Selection Sort
      1. Worst: Any
      2. Average: Any
      3. Best: Any
   4. Quicksort
      1. Worst: Sorted, either forward or reverse
      2. Average: Random
      3. Best: Random
2. Chart, line chart

   Description automatically generatedGraphs

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

1. 1. Worst:

Worst case here is a reverse sorted list for all algorithms. We do expect to see quicksort as one of the slowest here. It is interesting how much better selection sort does in this scenario.

* 1. Average:

Average was the random vector. This show us again as we expected, Quicksort is fastest in this scenario. Here we can see insertion and selection sort are similar asymptotically, but bubble sort is not on the same level of performance.

* 1. Best:

This set of tests were vectors sorted ascending. As expected, bubble and insertion sort take almost no time as the lists are already sorted. Selection sort we can see as expected is almost identical top both the worst and average scenarios. Last, we can see that quicksort again performs poorly on sorted lists, regardless of how the list is sorted.