Mhammed Alhayek

RUID: 156-00-6919

Programming Methodology Homework 4

Question 1: code submitted on Sakai.

Question 2: Implementation also in same file on Sakai.

Data:

Unsorted Inputs Data:

|  |  |
| --- | --- |
| Number of Elements | Quicksort RunTime |
| 4 | 27 |
| 16 | 160 |
| 64 | 806 |
| 256 | 4006 |
| 1024 | 19596 |
| 4096 | 92314 |
| 65536 | 1968351 |
| 1048576 | 39849514 |
| 16777216 | 766264664 |

Plot:

Plot is on a log-log graph with a base of 2.0.

Analysis:

Based off the data above and the plot, it can be found that the complexity of the Quicksort algorithm I wrote is roughly O(nlog(n)), with a prefactor of roughly 2.0. Thus, the expected run time for n elements is 2\*n\*log(n). This algorithm is significantly faster than the algorithms for selection, insertion, or bubble sort. The beauty of this is the quicksort algorithm was actually able to handle the inputs from dus-20.txt and dus-24.txt, which had over a million elements. This is why the quicksort algorithm is way more practical and should be used even if it is more complex to write.

Question 3: Using a sorted array as the input for Quicksort.

Data:

|  |  |
| --- | --- |
| Number of Elements | Quicksort Run Time |
| 4 | 25 |
| 16 | 119 |
| 64 | 636 |
| 256 | 2971 |
| 1024 | 14916 |
| 4096 | 77448 |
| 65536 | 2359979 |
| 1048576 | 62601376 |
| 16777216 | 1430566477 |

Plot:

Plot is on a log-log graph with a base of 2.0.

Analyis:

Inputting a sorted array into QuickSort makes the algorithm a little bit faster, however, the complexity is still O(nlog(n)), with a slightly smaller prefactor. The reason it is faster is because for my algorithm, the pivot is chosen to be the median value of the first, middle, and last element in the array. If the smallest element was chosen as the pivot on a sorted array, then the worst case scenario would take place, and the complexity would be O(n^2).

Question 4 part (a): MergeSort with additional memory. Implementation submitted on sakai.

Data:

With Unsorted Input:

|  |  |
| --- | --- |
| Number of Elements | MergeSort (Additional Memory) Run Time |
| 4 | 29 |
| 16 | 306 |
| 64 | 1757 |
| 256 | 9671 |
| 1024 | 49543 |
| 4096 | 238356 |
| 65536 | 5125828 |
| 1048576 | 102995592 |
| 16777216 | 1983320252 |

With Sorted Input:

|  |  |
| --- | --- |
| Number of Elements | MergeSort (Additional Memory) Run Time |
| 4 | 10 |
| 16 | 46 |
| 64 | 190 |
| 256 | 766 |
| 1024 | 3070 |
| 4096 | 12286 |
| 65536 | 196606 |
| 1048576 | 3145726 |
| 16777216 | 50331646 |

Plots:

Plots are on a log-log graph with a base of 2.0.

Analysis:

Based off the data above and the plot, it can be found that the complexity of the MergeSort algorithm I wrote is roughly O(nlog(n)), with a prefactor of roughly 0.4. Thus, the expected run time for n elements is 0.4\*n\*log(n). This algorithm is significantly faster than the algorithms for selection, insertion, or bubble sort. This makes the algorithm more practical than other sorting algorithms with O(n^2) complexity because it is able to handle larger inputs. Although this implementation of merge sort is faster, it takes up twice as much space in memory because it has to make a copy of the original array. Thus you compromise space with a faster run time. A better alternative would be to use quick sort, which has the same time complexity but better space complexity. With a sorted array as the input, the complexity is roughly O(n), which shows that as opposed to quick sort, merge sort may be even faster than O(nlog(n)) if the array inputted is in a good order already.

Question 4 part (b): MergeSort in place. Implementation submitted on sakai.

Data:

With Unsorted Input:

|  |  |
| --- | --- |
| Number of Elements | In Place MergeSort Run Time |
| 4 | 23 |
| 16 | 313 |
| 64 | 1970 |
| 256 | 17930 |
| 1024 | 170246 |
| 4096 | 2315082 |
| 65536 | 547642812 |
| 1048576 | Computer Can't Handle This |
| 16777216 |

With Sorted Input:

|  |  |
| --- | --- |
| Number of Elements | In Place MergeSort Run Time |
| 4 | 10 |
| 16 | 46 |
| 64 | 190 |
| 256 | 766 |
| 1024 | 3070 |
| 4096 | 12286 |
| 65536 | 196606 |
| 1048576 | 3145726 |
| 16777216 | 50331646 |

Plots:

Plots are on a log-log graph with a base of 2.0.

Analysis:

The merge sort in place algorithm is significantly worse than the regular merge sort algorithm in terms of time complexity. As n gets larger and larger, the in-place merge sort algorithm gets more and more slower than the additional memory implantation of merge sort. The in-place algorithm was actually so much worse that it couldn’t handle the arrays with so many elements of the data files dus-20.txt and dus-24.txt. The additional memory implementation can handle this. However, in the case that the arrays are already sorted, the in-place merge sort and the additional memory merge sort implementations are actually the same. This is because the merge function never does most of its methods, and returns after realizing the array is already sorted. The additional memory of the original merge sort implementation seems to be worth it considering how much more efficient the algorithm is in terms of run time.