Executive Summary: Divide & Conquer vs. Brute Force Algorithms

CS 2223 - Algorithms
Jonathan Gaines
April 14, 2017

In this experiment, lists of size *n* were sorted using divide and conquer and brute force algorithms with time benchmarks being taken for each. Quicksort was chosen as the divide and conquer algorithm (average case time efficiency: O(*nlgn*)) and bubble sort was chosen as the brute force algorithm (average case time efficiency: O(n^2)).

Pseudocode

Bubble Sort:

```
func bubblesort(list)
  for i from 1 to n
    for j from 0 to n-1
      if list[j] > list [j+1]
        swap(list[j], list[j+1])
```

Quicksort:

```
func quicksort(list, begin, end)
  if(begin < end)
    pivot = partition(list, begin, end)
    quicksort(list, begin, pivot)
    quicksort(list, pivot+1, end)

func parition(list, begin, end)
  pivot = list[begin]
  wall = begin
  for i = begin + 1 to end
    if list[i] < pivot
      swap(list[i], list[wall])
      wall = wall+1
  swap(pivot, list[wall])

  return wall
```

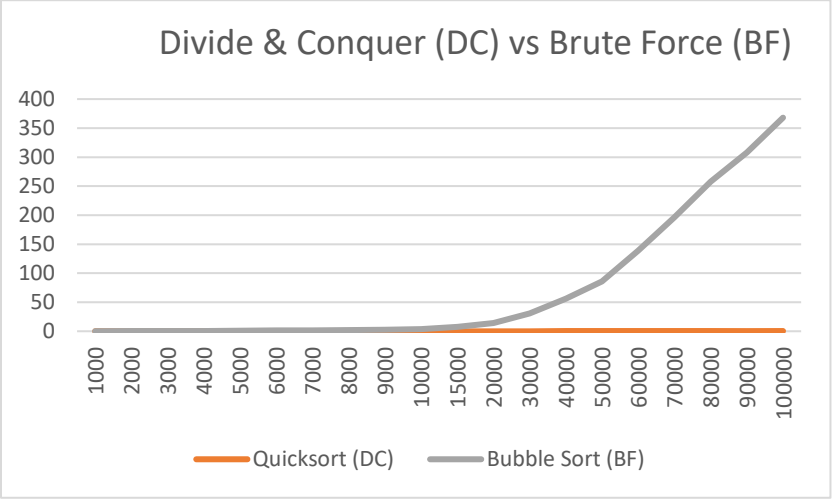
Set Sizes

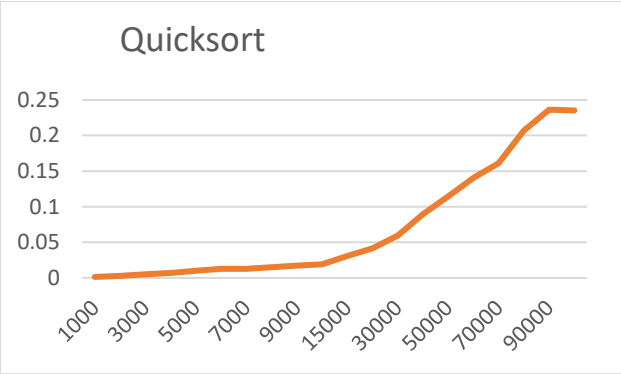
For this experiment, small, medium, and large dataset sizes were chosen. Prior to deciding the ranges of these datasets, a basic analysis was done by iterating through the algorithms with increasingly larger datasets. Once all of the data was collected, it was decided that a small dataset would be defined as a set ranging in size from 1 – 1000, a medium dataset would be 1001 – 15000, and a large dataset would be 15001 – 100000 all composed of randomly assigned integers. The logic behind these numbers is that a small dataset will never take longer than ~1/10 seconds to run both algorithms, a medium sized dataset will take no longer than ~10 seconds, and a large dataset will take ~10-370 seconds (These times were based on run times taken from my computer and may differ on other computers).

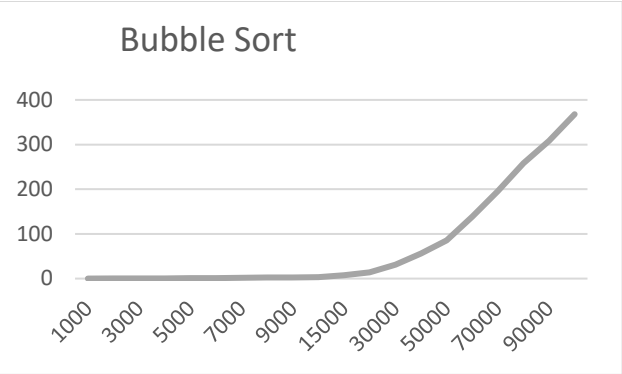
The data taken from this experiment is as shown below:

Set Size	Quicksort (DC)	Bubble Sort (BF)
1000	0.001376	0.030355
2000	0.002945	0.12595
3000	0.005091	0.278459
4000	0.006916	0.512601
5000	0.010056	0.802646
6000	0.012742	1.189979
7000	0.012745	1.591645
8000	0.015035	2.134203
9000	0.017143	2.647658
10000	0.019254	3.40316
15000	0.030709	7.512191
20000	0.04174	13.91132
30000	0.059371	30.85234
40000	0.089808	56.22572
50000	0.1141	85.78661
60000	0.14025	138.9139
70000	0.161242	195.8198
80000	0.206978	257.8942
90000	0.236214	308.0473
100000	0.235173	367.9427

These graphs are of the data shown above and has the set sizes on the x-axis and the time it took to complete (sec) on the y-axis.







Finding Largest and Smallest Values

It was decided that the best method to generate the largest and smallest values with each algorithm was to pass the entire list into a function which sorts that given list. Once a sorted list is generated, the first element and the last element will be the smallest and largest values. Using sorting algorithms as opposed to searching algorithms throughout this experiment was better for the time analysis because it showed a more profound difference between using brute force to divide and conquer. This is because when using a brute force searching algorithm such as a linear search, the time efficiency is O(n) as opposed to bubble sort's O(n^2) time efficiency.

Space Efficiency

The algorithms implemented throughout this experiment were written to minimize the amount of storage that each took. This was to prevent the data from being skewed by time constants used to access and write data. For the bubble sort algorithm, used to demonstrate brute force, the space efficiency is O(1) whereas quicksort, used as the divide and conquer algorithm, is O(n).

Conclusion

As seen in the data shown earlier, the divide and conquer algorithm, quicksort, was much more time efficient than the brute force algorithm, bubble sort. In a set size of 100000 random integers, quicksort sorted the list in less than one quarter of a second while bubble sort took over six minutes to sort. When considering that the time constants in the quicksort algorithm are significantly higher due to its O(n) space efficiency (as opposed to bubble sort's O(1) space efficiency), quicksort is still much more time efficient.