Timing Quicksort

Ryan Flageolle

Introduction

The first assignment was to analyze different variations of quicksort to see what deviations makes the algorithm run the fastest. The language I chose to complete the assignment in was Swift. The different variations that I checked were the Median of Three, Hybrid Sort, Lomuto partitioning, and Hoare partitioning; by timing them and comparing them to the base timing of quicksort. I found help with this assignment from the Ray Wenderlich github on sorts, the specific urls are mentioned in the comments above each method.

Setup

To test these different sorting algorithms, I initialized an array with integers from 1 to 100, 10,000, and 1,000,000 for the different tests; then using the shuffled method to randomize the array. To time the each of these situations I utilized the mach\_absolute\_time function to take the initial time and the time when the sort finished, which allowed me to get the overall time of each sort.

Median of Three

The median of three refines the choice of the pivot value to reduce the number of partitions by taking the first element, last element, and middle element and using the median of the three elements as the pivot value. On average this algorithm was quicker than basic quicksort but only slightly.

Hybrid Sort

Hybrid Sort takes into account that as the size of an array decreases the efficiency of quicksort as compared to insertion sort. This sort therefore will recursively quicksort the array until a small enough region is supplied and at that point it will insertion sort that region. This was less about solely timing against quicksort and more attempting to refine the region size at which this algorithm is most effective. To do this I started with the size 11, and progressively increasing until I noticed a peak of efficiency which was at size 25. The average time in decrease was 0.5 seconds at 1,000,000 elements.

Partitioning Variations

The two partitioning schemes tested was the lomuto and hoare. In my original quicksort method I used neither partitioning scheme instead using the swift filter() method, I believe that this slowed down the times for my non-partitioning specific methods to . I found that the lomuto scheme is significantly faster than any other tested sort except the hoare scheme, which highlights the obvious fact that if the number of swaps is reduced using a refined partitioning algorithm the elapsed time for quick sort is drastically reduced.

Results:

100 elements:

Base quicksort completed the sort of 100 elements in 0.001366952 seconds

median completed the sort of 100 elements in 0.0011322650000000001 seconds

bitter completed the sort of 100 elements in 0.000761576 seconds

lomuto completed the sort of 100 elements in 8.7775e-05 seconds

hoare completed the sort of 100 elements in 6.5944e-05 seconds

10,000 elements:

Base quicksort completed the sort of 10,000 elements in 0.195990404 seconds

median completed the sort of 10,000 elements in 0.18560017 seconds

bitter completed the sort of 10,000 elements in 0.14977273900000002 seconds

lomuto completed the sort of 10,000 elements in 0.026352542000000003 seconds

hoare completed the sort of 10,000 elements in 0.013911741 seconds

1,000,000

Base quicksort completed the sort of 1,000,000 elements in 24.836770716 seconds

median completed the sort of 1,000,000 elements in 26.293074062000002 seconds

bitter completed the sort of 1,000,000 elements in 22.225808525 seconds (size: 11)

bitter completed the sort of 1,000,000 elements in 21.713576568 seconds (size: 25)

lomuto completed the sort of 1,000,000 elements in 3.1297119490000003 seconds

hoare completed the sort of 1,000,000 elements in 1.8959021020000002 seconds