Jonathan Nguyen-Pham

016297682

10/08/2020

Lab 2 analysis

NOTE: made change to value range from 1-100,000 to 1-10,000 time seemed to stall for my laptop or it just took such an incredibly long amount of time to run once.

1. The sorted from 1-10,000 in increasing order would be the fastest due to how insertion sort searches for the next value to swap that is less than the current value and swaps it. In this case it searches the entire array and never needs to swap values thereby making it faster by bypassing the if statement entirely; its time would be similar to T(n) instead of T(n2).
2. The slowest would likely be the 10,000- 1 decreasing order array due to having to checking and swapping every single value in the array to their correct position. Making it a T(n2) which would be even worst than the random array’s T(n2) due to the likelihood that some subsets of values that are in order and not needing to be swapped; making random array ever so slightly better but still T(n2).
3. The fastest would be the sorted 1-10,000 in increasing order due to the balanced partitioning and the lack of swapping necessary to complete the sort. Compared to the others, like random, where it would be potentially balanced and it could potentially not need to sort particular values that are already in order. And for the decreasing order it would likely be slightly worse due to having to swap every value even though it is perfectly balanced.
4. Insertion sort would be faster due to not having the overhead of recursive calls being needed to run the algorithm; I believe it is the logn part that is recursive. Thus, the T(n) of insertion sort for this case would be outperform quicksort’s T(nlogn).
5. Quicksort would be faster on average on the random array due to the average of T(nlogn) for quicksort and the T(n2) of insertion sort; however, in the unlikely case that random is somehow randomly placed in increasing order insertion sort would win. The logn recursive time complexity of quicksort’s swapping and partitioning would beat out insertion sort’s T(n) if statement of swapping and iterating through of values.
6. My prediction that Insertion Sort being the best for the sorted in increasing order array was correct. My prediction for Insertion Sort being the slowest on the reverse array was incorrect possibly due to the reverse array being still ordered albeit in reverse order may have been the deciding factor. My prediction for Quicksort being the fastest on sorted was correct probably due to it not having the necessity of swapping values and only needing to traverse and create pivot points being the main factor for its time. My prediction for Insertion Sort being the fastest for the sorted array was correct due to the lack of swapping needed for Insertion Sort to do while Quicksort still needed to do recursive calls to itself to traverse values. I was incorrect on Quicksort being faster for the random array was likely due to values being sorted to some degree in the arrays thereby allowing Insertion Sort to not need to swap values while Quicksort still needing recursively to do the partitioning and pivot value locating.

EC: I predict that the classic Quicksort will be faster because of the inclusion of the pivot value in the left side of the subarray. Thereby, allowing it to be sorted as the max of the left side similar to Merge Sort using pivot points as max values of a subarray; albeit without the merging part.

The modified Quicksort consistently did the worst out of all the trials and data sets. It was nearly double the time of the classic Quicksort probably due to the imbalance of the partitioning of values on one side or the other. Insertion Sort of course did the best for the identical array of 10,000 1s due to the lack of swapping which lead to it linear T(n) time. The modified Quicksort’s worst time could have possibly been due to the extra criteria of a[i] < pivotValue && i % 2 == 0 making the swapping in the partitioning section more complicated and time consuming. However, the modified Quicksort did nearly the same but still worst time than the classic Quicksort for the identical data set.