CSC140 Advanced Algorithms
Spring 2024

Assignment 3 Report

Student Name: Illya Gordyy Student ID: 302682939

Part1: Algorithm Comparison

Report the results of your sorting experiments in the tables below. All times should be in milliseconds.

		10000	**100000**	**1000000**
Insertion Sort	Sorted	0.036 ms	0.324 ms	3.218 ms
	Nearly Sorted	0.039 ms	0.376 ms	3.751 ms
	Reversely Sorted	107.244 ms	10783.2 ms	1.08651 E+06 ms
	Random	54.178 ms	5387.99 ms	541801 ms

		10000	**100000**	**1000000**
MergeSort	Sorted	0.937 ms	10.527 ms	124.672 ms
	Nearly Sorted	0.923 ms	10.908 ms	124.093 ms
	Reversely Sorted	0.896 ms	10.309 ms	121.947 ms
	Random	1.456 ms	17.32 ms	210.817 ms

Quick Sort with the last element as the pivot for both 100000 and 1000000 causes Stack Overflow since with the pivot being the last element, the equation is T(n) = 2T(n-1) + cn resulting in θ (n^2) time complexity
The table below only gives the values for <u>Randomized pivot case.</u>

		10000	**100000**	**1000000**
QuickSort	Sorted	0.844 ms	9.399 ms	114.629 ms
	Nearly Sorted	0.844 ms	10.114 ms	116.622 ms
	Reversely Sorted	0.873 ms	10.205 ms	119.008 ms
	Random	1.201 ms	14.947 ms	177.817 ms

		10000	**100000**	**1000000**
LibSort	Sorted	0.528 ms	6.1 ms	70 ms
	Nearly Sorted	0.489 ms	5.689 ms	64.11 ms
	Reversely Sorted	0.444 ms	5.05 ms	55.973 ms
	Random	1.298 ms	16.327 ms	192.238 ms

Part2: Quicksort Focused Study

Report the results of your focused study of Quicksort in the table below.

	Recursion Depth	**Execution Time (ms)**
Simple Random	48 Recursion levels	177.817 ms
Median of Three	38 Recursion levels	189.324 ms
with InsertionSort	36 Recursion levels	158.515 ms
Your Best Result (Extra Work)	26 Recursion levels	145.326 ms

Extra Credit

Implementing both the median of three and the Insertion Sort gives slightly better results compared to the Library Sort. The first reason is because using the media of three gives us more chance of picking a true median value in the array. Therefore, rather than having to pick it only once, we pick it 3 times and choose the median value. Next, since the insertion sort has better performance time with shorter arrays we can avoid doing the recursion and expensive partitioning. Moreover, by using Insertion Sort we avoid addition memory usage, and overhead.

Discussion

Discuss the following points. This discussion is limited to the algorithms studied in class. So, library sort is excluded, because its internals are unknown.

1. For each input type (sorted, nearly sorted, reversely sorted, random), which algorithm has the best performance? Explain why?

   Sorted: Insertion Sort

   It has a faster time of sorting a sorted array since as the loop goes through each element it checks if a current element is less than the previous one, if the current element turns out to be greater than the previous, it simply breaks out of the loop and goes to the next element. Here it has a time complexity of θ (n) since it only assigns the current element to its own index without doing the inner loop.

   Nearly Sorted: Insertion Sort

   In this case, insertion sort simply skips all the sorted elements whereas merge sort and quick sort, due to a completely different structure, would still have to do **θ (n log n)** operations. Which is asymptotically greater compared to Insertion's sort best case of **θ (n). Lastly, the insertion sort, uses much less memory compared to other sorting algorithms.**

   Reversely Sorted: Quick Sort

   In my case, the time is slightly faster than the merge sort time. It turned out to be slightly better in reversely sorted array than the merge sort, mainly because of its partitioning algorithm which randomly assigns a pivot value. However, due to the nature of random pivot selection, quick sort time may also be a bit greater than the merge sort time. Considering this, quick sort is a less stable in its timing, compared to merge sort.

Random: Quick Sort
It performs much faster compared to merge sort due to the random pivot selection, and on average it will perform better than the merge sort, and better than the insertion sort.

2. How does the performance of each algorithm change when we change the input type? Explain why?

   Insertion Sort:
   - For input types of Sorted and Nearly Sorted arrays, the insertion sort has approximately the same computation time of θ(n).
   - As we reach the Reversely sorted array, the computer time is at its maximum, since it has the worst time complexity of θ(n^2). Here the algorithm has to loop through each element and then compare them to the previous ones, which in this case, would mean to re-assign each element. This results in the longest computation time
   - For the Randomly sorted data, it seems to be giving a much smaller time. This is because the data in this array is neither sorted(best case) nor reversely sorted(worst case) resulting in a more balanced mean time.

   MergeSort:
   - In the case of Merge Sort, it has approximately the same time for the first three input type of sorted, nearly sorted and reversely sorted. This is because in each of these cases they will approximately the same number of operations.

   - In the last case of randomly sorted array, it shows a significant increase in time compared to other input types which is the result of the lack of any order in the array compared to the previous three cases.

   QuickSort:
   - Quick Sort with random pivot has approximately the same times the first three cases. If not for the random pivot selection the time complexity would have been completely different. Specifically, all of the would result in θ(n^2), though a randomly sorted array would still have a similar time. Random pivot selection helps avoid the worst cases and keep an θ(n log n) time complexity

3. How does the performance of each algorithm change when we change the input size? Explain why?

Insertion Sort:
Input size significantly changes the time of the algorithm the number of iterations grows n^2 (quadratically). Therefore, larger the input size the more the input type starts to matter.

MergeSort:
In this case the input size doesn't make much difference on the time since the worst case is (n log n) which is far less than quadratic time increase. The time consistently growth nlogn consistently.

QuickSort:
Similar to merge sort, input size usually doesn't affect much since the average case for quick sort with randomized pivot is **θ (n log n).** But in case if the pivot selection leads to the worst case – of **θ (n^2**) the input will have a quadratic increase to the algorithm time.

4. Do you have any other observations or insights?

- When comparing Quick Sort and Merge Sort, even though Merge Sort for the most part shows slightly slower results than the Quick Sort, it has a much more stable structure and will almost certainly always provide similar timing. The Quick Sort on the other hand will be less stable.
- Merge Sort requires additional memory for the merging process, making it less memory-efficient than Quick Sort, which performs sorting in-place.
- Quick Sort is quite flexible to different adaptation such as, random pivot, median of three, and insertion sort.
- Median – of – three is supposed to give better times compared to random pivot, however, finding the median seems to be affecting the time significantly.
- Media of three with the Insertion Sort works better than the Library Sort