CSC206 Algorithms and Paradigms
Spring 2018
Assignment 3 Report

Student Name: Divya Sindhuri Devarakonda Student ID: 219220782

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	0.4	2.69	9.27
	Nearly Sorted	0.49	4.74	4.4
	Reversely Sorted	62.5	3009	315219.5
	Random	33.71	1536	152171.5

		10000	100000	1000000
MergeSort	Sorted	4.983	24.56	102.9
	Nearly Sorted	3.99	21.01	117.69
	Reversely Sorted	2.88	12.41	85.93
	Random	3.35	13.44	163.05

		10000	100000	1000000
QuickSort	Sorted	5.8	19.4	71.16
	Nearly Sorted	1.2	5.6	74.11
	Reversely Sorted	0.5	6.0	71.9
	Random	1.14	9.7	122.5

		10000	100000	1000000
LibSort	Sorted	0.5	2.7	6.3
	Nearly Sorted	1.7	14.4	88.2
	Reversely Sorted	0.4	2.7	5.0
	Random	1.3	43.9	106.5

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	125.6
Median of Three	45	118
with InsertionSort	40	117
Your Best Result (Extra Work)		

Simple Random: Here we choose a random number and swapping with the last index. Recursion depth differs based on the random element taken. This reduces the probability of taking largest or smallest number as pivot element.

If we use last element as pivot for sorted, nearly sorted and reversely sorted cases, we get stack over flow error because of huge recursion dept. But taking a random number and swapping with last index stack over flow error will not come as we are reducing probability of taking smaller or larger number as pivot.

Median of three: choosing three Random values from array and considering median value as pivot. Recursion depth slightly reduced when compared to random as we are choosing the median of three numbers.

With Insertion Sort: As we know insertion sort works pretty good for the smaller inputs so this helps us in reducing the recursion depth when input size becomes less.

Discussion

Discuss the following points:

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

   Sorted:

   Insertion Sort: running time is Θ(n) .When data is sorted, it is the best-case scenario for insertion sort. Instead of nested loop only one loop (outer loop) will get executed n times as data is sorted and the running time is linear. Compared to other sorting techniques insertion sort works good for sorted data.

   Nearly Sorted:

   Insertion Sort :For nearly sorted data the number of operations will be more than fully sorted.data i.e best case scenario but less than n-square. Running time will be between W(n) and O(n square). But compared to other algorithms this works fine with nearly sorted data with running time Θ(n).

   Reversely Sorted:

   STL Sort works good with reversely sorted data. As it is inbuilt library function it works fine with almost all kind of data.

   Random: In random sorted data, Quick sort and STL sort works good. In some cases, quick sort showed better performance. It depends on the selection of pivot.

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

Insertion Sort:

Sorted: As the data is sorted no of executions of inner loop is zero so outer loop has n iterations. Insertion sort has best performance with this input type.

Partially Sorted List: No of executions of the inner loop is more than zero but less than theta (Nlogn).

Reversely Sorted: Number of executions of the inner loop is n and outer loop is also n. Insertion sort's worse case scenario is this kind of input type. Running time of this case is Θ(n square) .

Randomly Sorted: Performance of this input will in between partially sorted and reversely sorted number of inner loop executions is between numbers of inner loops of partially sorted and reversely sorted.

MergeSort: Merge sort performance won't be affected much by the type of input as it is using divide and conquer technique in solving. It divides the problem into smaller sub problems

Quicksort: Quick sort execution time is almost same with all kind of inputs. Here performance of quick sort depends more on selection of pivot rather than the type of input given. From the above table, quick sorts execution time is slightly more when data is random.

STLSort: All kind of inputs Sorted, Nearly sorted, reversely sorted, Random work good with STL sort and this sorting technique gives good performance with all kind of data. Its execution time is less when compared to other techniques.

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

Insertion Sort: As the input size is getting increased performance is reduced in case of insertion sort. Number of comparisons will increase as data is increased. But this performance change is not very predominant in sorted input. But in reversely sorted data, with larger input size (one million) performance of insertion sort is drastically decreased.

MergeSort: In Merge sort, increasing input size decreased performance. But this is not very huge performance difference. But because of large input size more memory is required for merge sort which also increases space complexity. Number of operations are increased along with size.

QuickSort: In quick sort as input size is very large execution time is large when compared to smaller input size. But it is better than merger and insertion sort when input size is more. In quick sort rather than input size, pivot selection is effecting the performance.

LibSort: With larger inputs execution time is more and performance is less in lib sort. For random data there is performance change with Lib sort.

4.Do you have any other observations or insights?

1. Insertion sort works good with smaller inputs and sorted data
2. Merge sort has more space complexity when input size is large
3. Quick sort performance is depending more on pivot selection rather than input size