

06.04.2023

CME 2204 – ALGORITHM ANALYSIS

ASSIGNMENT 1
MERGE SORT AND QUICK SORT COMPARISON REPORT

Prepared by
Mustafa Eren Işıktaşlı 2020510043

Chapter 1 Results of Experiments:

	EQUAL INTEGERS			RANDOM INTEGERS			INCREASING INTEGERS			DECREASING INTEGERS		
	1,000	10,000	100,000	1,000	10,000	100,000	1,000	10,000	100,000	1,000	10,000	100,000
mergeSort **TwoParts**	3,862 ms	1,481 ms	12,169 ms	0,536 ms	4,192 ms	15,458 ms	0, 122 ms	0,441 ms	8,358 ms	0,235 ms	0,409 ms	7,756 ms
mergeSort **ThreeParts**	0,481 ms	1,355 ms	10,331 ms	0,148 ms	3,509 ms	14,283 ms	0,105 ms	3,537 ms	4,618 ms	0,281 ms	3,026 ms	2,642 ms
quickSort **FirstElement**	6,022 ms	24,43 ms	820 ms	1,458 ms	13,216 ms	615 ms	3,437 ms	15,794 ms	1721 ms	0,342 ms	16,430 ms	1821 ms
quickSort **RandomElement**	3,741 ms	17,49 ms	709 ms	0,153 ms	0,241 ms	14,787 ms	0,64 ms	0,334 ms	2,852 ms	0,69 ms	0,315 ms	3,063 ms
quickSort **MidOfFirstMidLastElement**	3,180 ms	13,44ms	904 ms	0,103 ms	0,689 ms	8,683 ms	0,28 ms	0,104 ms	1,015 ms	0,150 ms	0,104 ms	1,093 ms

Table 1.0 Program run time results

Chapter 2 Report

In this project, we have checked the differences in the sorting algorithms which are merge sort with two-way, merge sort with three-way, quick sort with first number pivot, quick sort with random number pivot and quick sort with mid of the first mid and last element pivot.

Also, these sort algorithms had been checked for different scenarios which are an increasing array, decreasing array, all members equal array, and also a random array. These scenerios are calculated for array size 1000, 10000 and 100000. These calculations had been reported on the table 1.0.

Firstly let's check the theoretical calculations for the sort algorithms:

Merge Sort: Merge sort is a sorting algorithm that calls itself recursively, we can calculate its run time with a recursion tree. The run time is Merge Sort is $O(n * (\log_2 n))$.

Merge Sort With Two Ways: This algorithm divides an array with two parts recursively when it reaches to the smallest part, it compares the elements and sorts them, then applies these states to all pieces from small ones to big ones. So we know that a tree will be created by end of this algorithm and the height of this tree is $(\log_2 n)$. Also we will apply the sort operations in all levels so we have $O(n * (\log_2 n))$ run time for this algorithm.

Merge Sort With Three Ways: This algorithm divides an array into three parts recursively when it reaches to the smallest part, it compares the elements and sorts them, then applies these states to all pieces from small ones to big ones. So we know that a tree will be created by end of this algorithm and the height of this tree is $(\log_3 n)$. Also we will apply the sort operations in all levels so we have $O(n * (\log_3 n))$ run time for this algorithm.

Quick Sort: Quick sort is a sorting algorithm which's principle is divided and conquer such as merge sort. Its difference is, while merge sort is dividing and sorting, quick sort works per element. It finds an element's final position in each movement and finally sorts the list. For doing that it uses a pivot number and in each time it places the integer whose index is the pivot number. The run time of the quick sort can be variable according to the pivot. it depends from $O(n * (\log_2 n))$ to $O(n^2)$.

Quick Sort With First Integer Pivot: In each loop, the pivot is the first element of the chosen list. The algorithm finds the final location of the element of the first index. There are some handicaps in this pivot, if the number in the first digit is a very high or very low-valued integer. Then quick sort starts to behave as the insertion sort so its run time greatly increases. So it's obvious that choosing the first element as the pivot is too risky and mostly it won't be efficient at all theoretically. Also by the studies that we had done. We obtained that the worst run time on the sorting algorithms is Quick Sort with the first digit Pivot.

Quick Sort With Random Element pivot: In each loop, an element of the list is chosen randomly as the pivot point. This approach is more better than first integer pivot because even in some points high or low values are selected, its obivous that mostly average values will be selected. So even it is not the most efficient algorithm it is quit good and its run time is almost $O(n*\log_2 n)$.

Quick Sort With Mid of FirstMiddleLast element pivot: In each loop, the middle value of the first, middle and last element is chosen as the pivot point. This approach is one of the best algorithms for the quick sort because there is more chance to pick the middle integer of the list. Also this approach works best with increasing or decreasing lists. İts run time is close to $O(n\log_2 n)$.

Secondly, lets Check the results of table 1.0:

It seems that in equal lists merge sort is way more successful than all quick sort types and also three-way merge sort is the best solution for this type according to the results. Also theoretically the results are fitts.

For randomly created lists the best solution is even the merge sort is quite fast, quick sort with midOf of has the best run time. Theoretically, the results fit again because quick sort gets quicker when pivot points are average.

For the Increasing list, same as the randomly created list if you pick the average points as pivot, quick sort has a very good run time as merge sort. But if you choose a pivot point as the first index of the list then its run time is the worst.

For the Decreasing list, same as the increasing list if you pick the average points as pivot, quick sort has a very good run time as merge sort. But if you choose a pivot point as the first index of the list then its run time is the worst.

As a result, theoretical results and practical results are almost the same, and as an addition in bigger lists, merge sort with three ways has a better performance than merge sort with two ways.

Answers for Question 3-a is down below:

The best sorting algorithm for 3-a among the project is: Quick sort would be a good choice because we know that in the dictionary, there are lots of elements and to handle with elements we can use either quick sort or merge sort also the best way is we can pick the exact pivot points in this dictionary for quick sort which is especially quick sort with mid of First Mid Last, it is one of the best choices for this problem but Considering the equal elements in this list and also stabilization, because there will be lots of same valued words in the dictionary and they will make quick sort slower. I would rather Merge sort with three ways, instead of quick sort.

The best sorting algorithm for 3-a among all algorithms is: I won't change my decision in this part too but in addition, heap sort would be a good choice for this problem too, because it has the same time complexity and it is as stable as merge sort but mostly in run time, we see that merge sort is faster than heap sort in most points. So decision is still merge sort with three ways.

Answers for Question 3-b is down below:

The best sorting algorithm for 3-b among the project is: The data that we will collect from e devlet is like a tree form. Also, we can say that data goes from past to future. So the list we will take from e devlet is sorted from oldest to youngest and we need to sort it from youngest to oldest. So it is like sorting decreasing order list in this project but its not the same because of the similarities in ages. There is a high possibility that a high amount of people will have the same birthday so there will be lots of equality in the list. So using a quick sort is too risky here the best solution will be a merge sort. Especially a merge sort with three ways.

The best sorting algorithm for 3-b among the all algorithms is: Heap sort can be used in this problem because the data that we collected is a tree and sorted from high to low. To sort this list from low to high one of the best ways is heap sort. Although, merge sort is as good as heap sort in this part, and if we think about the stabilization they both are quite stable. But anyway, my choice for this is Merge sort with three ways because it is easier to implement than heap sort, but also heap sort can be used and it is very efficient.