Srting algorithms: Mergesort, Heapsort, Quicksort

Nusultan Bazargaziyev

29-11-2021

1 Introduction

In this project I considered 3 sorting algorithm types: Mergesort, heapsort, quicksort. The project will show how much time each algorithm take and compare the results. After tests we can say which algorithm is the most efficient.

1.1 Merge sort

Merge sort is divide and conquer type algorithm, which means that algorithm divides array into two parts recursively.

Merge sort divides array into right and left parts, and then divides right and left parts until array size will be equal to one. Then we use function to merge right and left parts. In that function we should create new array and compare elements of right and left arrays, If the right element is less than the left element, we add it to new array and do opposite if left element is less than right element. And if there will be situation where one array is already added to new one, but the other one contains elements that weren't added to new array, we should use while loop where the elements of non-empty array will be added to the new one.

And because we divide the array into two parts Big O of merge sort will be: $O(n^*\log\ n)$

1.2 Heap sort

Heap sort is an algorithm with two steps. First step is to convert array into the heap(binary-tree), then sort it. When you convert the array into the heap you don't need to create new array, you operate on original one. To create heap we need to use next formulas: 2n+1, 2n+2. The heap should contain the following rules: Parent element is always bigger then child one and parent can't have more than two 2 elements.

Like mergesort, heapsort has a running time of $O(n^*\log n)$

1.3 Quick sort

Quick sort is another example of divide and conquer algorithm. First, we should chose pivot element out of array, then divide array into two parts: left part with elements which are less than pivot, right part with elements which are greater than pivot. These parts should be contained in two different arrays called "left" and "right". And recursively sort right and left parts until size of the right or left part will be equal zero.

The algorithm takes $O(n*\log\ n)$. In the worst case it takes $O(n\ *\ n)$.

2 Methodology

The algorithms were implemented in C++. The algorithms took randomly generated arrays of values from 0 to 10,000 and already sorted arrays.

The algorithms were tested on arrays of sizes from 100 to 10,000. Each arrays size were tested 100 times. And also the algorithms were tested on array of sizes from 10 to 1,000, so we could know if array size could affect to sort timing.

All information about compile time, average time for each array size, worst case of each array size and best case of each array size are written in txt and csv files.

3 Results

You can check out graphs with the results of compiling in corresponding folders called "Graphs" and "Avg_Worst_Best". In "result_Merge_Heap_Quick_Big_size" you can see comparison of Mergesort, Heapsort and Quicksort algorithms with big array sizes. As you can see graph shows that longest time to compile is mergesort algorithm. On graph called "result_Heap_Quick_Big_size" we can see that heapsort takes longer time than quicksort. Graph shows that heapsort increase faster than quick sort, when size of array increases. In "Avg_Worst_Best" folder you can fine Worst, Best and Average compile time of sorting algorithms.

4 Conclusion

On sorted arrays all algorithms performs faster than in unsorted, but mergesort always takes longer time than others. If consider non-sorted arrays, mergesort shows the worst results, while quicksort and heapsort performs much faster. Even so, quicksort performs 3 times faster than heapsort in big sizes. In small sizes everything stays the same, except heapsort and quicksort, they perform almost the same.

So we can conclude that quicksort is the best sorting algorithm between heapsort and mergesort.