Insertion, Quick, Heap and Radix Sort

Murad Ibadov

November 30, 2023

**1 Introduction**

In this paper we will investigate 4 well known sorting algorithms(Insertion, Quick, Heap and Radix).Through this investigation, we aim to gain insights into the effectiveness and performance of these sorting methods. The final goal of this research is to find the fastest sorting algorithm either by choosing from the existing ones or combining a multiple of them and we will call it as a hybrid sorting algorithm.

**1.1 Insertion Sort**

The insertion sort algorithm sorts one item at a time. It goes through the array, placing each element in the right spot, assuming the previous ones are already sorted in ascending order. When it reaches a number, it compares and either shifts smaller ones to make room or places the number if it's equal or bigger. This continues until all numbers are sorted. It's quick for small lists but less efficient for larger ones. The algorithm is stable and works in place. Average complexity: O(n^2), Worst case: O(n^2), Best case: O(n). The algorithm is in-place and stable.

**1.2 Quick Sort**

Quicksort recursively partitions an array and arranges elements based on comparisons with a chosen value known as the pivot. This algorithm follows a divide-and-conquer approach by breaking down the array into subarrays and sorting them.

While there are various ways to select the pivot, a commonly favored option is to choose the last element of the array. After the pivot is chosen, the data is divided into two subarrays—one containing elements less than the pivot and the other containing elements greater than the pivot. Subsequently, these subarrays undergo recursive sorting.

For the time complexity of the Quicksort algorithm, in a best case scenario it will be O(nlogn) and in a worst case scenario it will be O(n^2).

**1.3 Radix Sort**

Radix sort is a sorting algorithm that categorizes elements into buckets based on their individual digits, progressing from the least significant digit (LSD) to the most significant digit (MSD). Let's consider an 8-element array. Firstly, we find the largest number in the array because we must traverse all significant locations of all elements. Next, we sort the elements based on the value of the unit place. Subsequently, we sort the elements based on the value of the tenth position. This process is repeated until it reaches the last significant location.For the best case, average case, and worst case, the time complexity remains O(d \* (n + k)).

**1.4 Heap Sort**

Heap sort is a sorting method rooted in the Binary Heap data structure, relying on comparisons between elements. It shares similarities with selection sort, as both algorithms involve identifying the maximum element and positioning it at the end of the array. This process is iteratively repeated for the remaining elements until the entire array is sorted. In essence, heap sort organizes data by leveraging the hierarchical structure of a binary heap, ensuring that the maximum element consistently surfaces during each iteration. The comparison-based nature of heap sort contributes to its efficiency, making it a reliable technique for sorting arrays in ascending or descending order. For Worst, best and average case the time complexity O(nlogn)

**1.5 Hybrid Sort**

After our investigation of finding the most efficient algorithms we will consider our hybrid sort.

**2 Methodology**

The algorithms were implemented in C++. The algorithms were tested on arrays of sizes from 100, 200, 300, . . . , 10000 and also for small arrays as 10 to 100.

**3 Result**

Firstly, we will analyze how is the sorts going for smaller size arrays.

It is clear from first graph that radix sort and heap sort are the sort that takes the most time.

So, now we will remove the radix and heap sort from the graph and use another graph to compare the others.

s

From this graphs, quick and insertion sorts are better than others. After ≈40 quick sort takes least time, but before ≈40 insertion sort is the best.

For now let’s compare the sorts for greater sizes.

As you can see insertion sort is the worst case with larger dateset. Now, insertion sort is not good as before and takes more time than rest of the sorts. Generally, quick sort can be count as the best.

If we try again for bigger sizes, we will see:

Form the graph it is easily considerably that heap and insertion sorts are not efficient .Radix sort also performs well however the best performance we can observe on quick sort.

**4 Conclusions**

Quick sort is efficient if the size of the input is very large, but insertion is more efficient than quick sort in case of small arrays. So, we will combine the two algorithms to sort efficiently using both approaches.