Sorting slowly: comparison between insertion, selection and bubble sort

**Yehor Yudin**11/17/2023

**1 Introduction**

In this paper, we investigate a classical problem: sorting. The problem is to arrange an array of n integers according to some total order which is computed in O(1). We use < as the total order. In this paper we compare four comparison-based sorting algorithms: heap, insertion sort, quick sort, and radix sort.

* 1. **Insertion sort**

Insertion sort repeatedly inserts elements into a sorted sequence. Inserting an element into a sorted sequence is done by moving all elements that are larger than the value being inserted to the index one larger than currently occupied, starting from the largest. The value to be inserted is the moved to the array at index after the first value that is smaller or equal to the inserted value. First, the element located at index 0 forms the sorted part of the array. The algorithm then performs n − 1 insertions, starting from the second element in the array to the last element. The average complexity of insertion sort is O(n 2 ). The insertion sort has a best case: the array sorted in an ascending order. In such a case, the complexity of insertion sort is O(n). The worst case of insertion sort is the array sorted in descending order. In such a case, the complexity of insertion sort is O(n 2 ), same as average case. The algorithm is stable and in-place.

**2.2 Quick Sort**

Quick sort partitions the array into smaller segments and recursively sorts them. It has an average time complexity of O(n log n) and is in-place. Quick sort's performance can degrade to O(n^2) in the worst case.

**2.3 Radix Sort**

Radix sort processes the digits of the numbers, sorting them from the least significant digit to the most significant. It has a time complexity of O(nk), where k is the number of digits. Radix sort is not an in-place algorithm.

**2.4 Heap Sort**

Heap sort builds a binary heap and repeatedly extracts the maximum element. Its average time complexity is O(n log n), and it is in-place. Heap sort is not stable.

**3. Methodology**

The algorithms were implemented in C++. The results were generated for two categories: sorted array (best case of insertion sort and bubble sort) and randomly shuffled arrays of values 0, 1, . . . , n − 1. The algorithms were tested on arrays of sizes from 100, 200, 300, . . . , 10000. In both categories, each algorithm was given the same input data. Each array size was tested 100 times. The result for each size is the average time it took for each algorithm to sort the input array.

**4. Results**

As you can see on the graph –

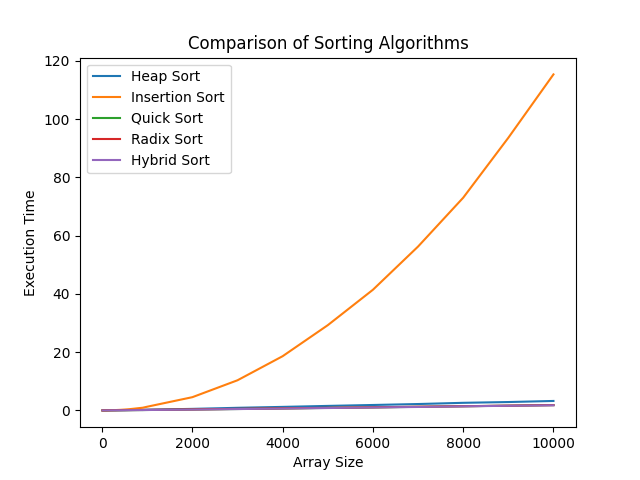
Hybrid is better than all other algorithms

Radix is better for numbers that larger than 9000 N > 9000,

Quick : 50 > N < 10000

Insertion : N < 50

Heap : Sucks all the time



4 Conclusions

Hybrid is better than all other algorithms