Benchmarking Sorting Algorithms In Python

INF221 Term Paper, NMBU, Autumn 2020

Jon-Mikkel Korsvik
jonkors@nmbu.no

Yva Sandvik
ysandvik@nmbu.no

ABSTRACT

In this paper, we analyse …

1 INTRODUCTION

Sorting algorithms are used to solve one of the key problems of computer science known as "The sorting problem". This involves an input sequence of n numbers $(a_1, a_2, \ldots, a_n)$, where the output is a permutation of the input sequence such that the numbers are ordered in an ascending order. The two main aspects to a sorting algorithm is its time complexity (speed) and its space complexity (memory usage). (Kilde: Lecture 6 ipynb, Plesser)

During this investigation we have assessed the performance of these sorting algorithms under certain assumptions regarding their time complexity. We explored their efficiency depending on the type of elements in a list that are to be sorted, as well as the length of the lists.

In the following subsections we will provide theory, pseudocode as well as details surrounding the methods used when comparing the following sorting algorithms:

- Quadratic algorithms
  - insertion sort
  - bubble sort
- Sub-quadratic algorithms
  - mergesort
  - quicksort
- Combined algorithm
  - mergesort switching to insertion sort for small data
- Built-in sorting functions
  - Python 'sort()'
  - NumPy 'sort()'
- Possibly a few more if we have enough time
  - Radix sort
  - Heap sort

2 THEORY

The first two algorithms we analyse are the quadratic sorting algorithms insertion sort and bubble sort. These two sorting algorithms are known as quadratic sorting algorithms because their Big O time complexity is $O(n^2)$.

Insertion sort is a simple comparison based sorting algorithm that iterates through a list one step at and starts by looking at the second value in an array and compares it to the one before, finding the correct position for the value in the sorted end of the input list (left side).

Bubble sort...

2.1 Algorithm 1 - Insertion sort

Listing 1 Insertion sort algorithm from **?**, Ch. 2.1.

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      i = j − 1
4      while i > 0 and A[i] > key
5          A[i + 1] = A[i]
6          i = i − 1
7      A[i + 1] = key
```

Pseudocode for the insertion sort is shown in listing 1. Best case runtime for this algorithm is:

$$T(n) = \Theta(n) \ . \tag{1}$$

This is achieved when the input array is already sorted. Meaning the input value in each iteration is only compared to the rightmost value of the sorted list.

Worst case runtime occurs if the input list is in reversed order. This gives a quadratic running time:

$$T(n) = \Theta(n^2) \ . \tag{2}$$

The average runtime is also quadratic, making insertion sort a bad choice for sorting large lists, however it is one of the best and quickest alternatives when it comes to sorting smaller lists.

We are going to compare insertion sort to several other comparison based sorting algorithms, such as bubble sort, quick sort and merge sort.

2.2 Algorithm 2 - Bubble sort

Listing 2 Bubble sort algorithm from **?**, Ch. 2.1.

```
BUBBLE SORT(A)
1  swapped = true
```

Pseudocode for the bubble sort is shown in listing 2.

2.3 Algorithm 3 - Merge sort

Listing 3 Merge sort algorithm from **?**, Ch. 2.1.

```
MERGE SORT(A)
```

Pseudocode for the merge sort algorithm is shown in listing 3.

Listing 4 Quick sort algorithm from ?, Ch. 2.1.

QUICK SORT(A)

Table 1: Versions of files used for this report; GitLab repository https://x.y.z.

File	Git hash
`utility.py`	`8ec07210f`
`src`	`396d8a309`
`plot_creation.ipynb`	`8ec07210f`
`benchmark_results.csv`	`88c28d55c`

2.4 Algorithm 4 - Quick sort

Pseudocode for the quick sort algorithm is shown in listing 4.

2.5 Algorithm 5 - Merge sort combined

Listing 5 Merge sort combined algorithm from ?, Ch. 2.1.

MERGE SORT COMBINED(A)

Pseudocode for the merge sort combined algorithm is shown in listing 5.

3 METHODS

Short description of what we have done so far and how:

- Our test data is generated using the class function ArrayGenerator found in our utility file.
- First test data is generated, then our benchmark function times how long each algorithm uses to sort given lists with given lengths.
- The timer function times each test a given number of repetitions and returns all the results (so that they can be saved and used later), as well as showing the mean.
- Mac OS and Windows 10, Python version 3.8.3 and 3.29?
- Git hashes are provided in table 1.

Listing 6 Expert from benchmark code.

```
for algorithm in kwargs['function_list']:
     array_copy = copy(kwargs['array'])
     record[algorithm.__name__] = []
     for _ in range(iters):
         start_time = time.perf_counter()
         algorithm(array_copy) # Runs algo
         end_time = time.perf_counter()
   times.append(bench(func, n))
```

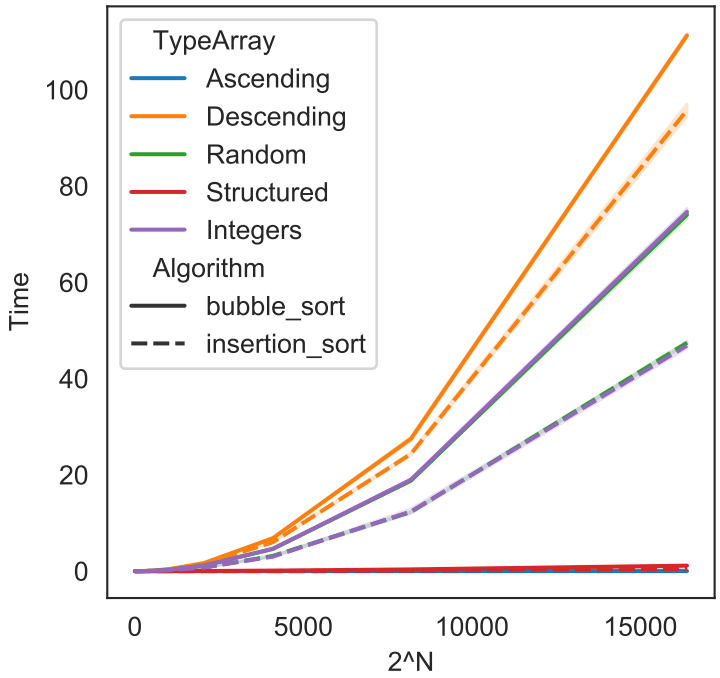

Figure 1: Benchmark results for insertion sort and bubble sort.

4 RESULTS

Until now have found that insertion sort and bubble sort which are quadratic in time complexity combined with other methods like for example merge sort, drastically reduce sorting time by reducing the callstack and memory complexity. Shown in results, by optimizing the difference beneath merge sort.

5 DISCUSSION

In this section, you should summarize your results and compare them to expectations from theory presented in Sec. 2.

ACKNOWLEDGMENTS

We are grateful to …for ….