Complexity Analysis of Sorting Algorithms

Ricardo Frumento

University of South Florida
Tampa, FL
Fall 2021

Contents

1 Introduction

For this project four sorting algorithms are used to compare the empirical behaviour with the theoretical run time. The algorithms are SelectionSort, InsertionSort, MergeSort, and QuickSort. They are shown below.

```
Data: array data
Data: int size
Result: array sorted
SelectionSort(data, size):
    for i = 1 to size do
        Let m be the location of the min value in the array data[1..n];
        Swap data[i] and data[m]
    end
return data;
```

```
Data: array data
Data: int size
Result: array data
InsertionSort(data, size):
    for i = 1 to size do
        j = i;
        while j > 0  &&  data[j − 1] > data[j] do
            Swap data[j] and data[j − 1];
            j = j − 1;
        end
    end
```

```
Data: array data
Data: int size
Result: array sorted
MergeSort(data, size):
    if size ≤ 1 then
        return data
    end
    mid = floor((size + 1)/2);
    left = MergeSort(data[1..mid], mid);
    right = MergeSort(data[mid + 1..size], size − mid);
    sorted = array(size);
    l = r = 1;
    while l < mid && r ≤ do
        if left[l] < right[r] then
            sorted[l + r − 1] = left[l];
            l = l + 1;
        else
            sorted[l + r − 1] = right[r];
            r = r + 1;
        end
    end
    sorted[l + r − 1..mid + r − 1] = left[l..mid];
    sorted[mid + r..n] = right[r..n − mid];
    return sorted
```

```
Data: array data
Data: int start
Data: int end
Result: array sorted
QuickSort(data, start, end):
    if start > end then
        partition = partition(data, start, end);
        QuickSort(data, start, partition − 1;
        QuickSort(data, partition + 1, end
    end
return data;
```

The theoretical run time for all the sorting algorithms is presented in the table below

	Best-case	Average-case	Worst-case
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
InsertionSort	$\Omega(n)$	$\Omega(n^2)$	$O(n^2)$
MergeSort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
QuickSort	$\Omega(n \lg n)$	$\Theta(n \lg n)$	$O(n^2)$

Table 1: Theoretical run time for the four sorting algorithms

The sorting algorithms were run several times and the empirical behaviour is shown in the next section. There are a total of twelve cases and for half of the cases were recorded the smallest array that takes 30 milliseconds or more per run to sort, the time to sort the smallest array that takes 30 milliseconds or more, and for the other half the largest array sorted, and the time required to sorted the largest array sorted.

2 Data

Algorithm	Input Type	n_{min}	t_{min}(ms)	n_{max}	t_{max}(ms)
SelectionSort	Constant	$46 \cdot 10^2$	30	10^6	107690
	Sorted	$46 \cdot 10^2$	30	$5 \cdot 10^5$	268420
	Random	$46 \cdot 10^2$	30	$5 \cdot 10^5$	266832
InsertionSort	Constant	10^7	30	10^9	3240
	Sorted	10^7	27	10^9	3542
	Random	$61 \cdot 10^2$	30	10^6	627013
MergeSort	Constant	$47 \cdot 10^4$	30	10^9	82437
	Sorted	$48 \cdot 10^4$	30	10^9	87445
	Random	$22 \cdot 10^4$	30	10^8	16632
QuickSort	Constant	$46 \cdot 10^2$	30	$2 \cdot 10^5$	42682
	Sorted	$35 \cdot 10^4$	29	10^9	106750
	Random	$2 \cdot 10^5$	29	$5 \cdot 10^8$	98347

Table 2: Data for the 12 tests executed

3 Analysis

The analysis will be made comparing the empirical values using three different equations

$$f_1(n) = n$$

$$f_2(n) = n \ln n$$

$$f_3(n) = n^2$$

Now it is possible to find the ratios using the relations

$$f_i(n_{max})/f_i(n_{min})$$

where i can be 1, 2, or 3. The results are shown in the table below

Algorithm	Input Type	$\frac{t_{max}}{t_{min}}$	f_1	f_2	f_3	Behaviour
SelectionSort	Constant	35897	217	356	47259	$O(n^2)$
	Sorted	8947	109	169	11815	$O(n^2)$
	Random	8894	109	169	11815	$O(n^2)$
InsertionSort	Constant	108	100	129	10000	$O(n)$
	Sorted	131	100	129	10000	$O(n \lg n)$
	Random	20900	164	260	26874	$O(n^2)$
MergeSort	Constant	2748	3376	3376	4526935	$O(n \lg n)$
	Sorted	2915	2083	3300	4340278	$O(n \lg n)$
	Random	554	455	681	206612	$O(n \lg n)$
QuickSort	Constant	1423	63	63	1890	$O(n^2)$
	Sorted	3681	2857	4638	8163265	$O(n \lg n)$
	Random	3391	2500	4102	6250000	$O(n \lg n)$

Table 3: Experimental ratio and expected ratios

It is fairly easy to see when the experimental result aligns with the quadratic growth, the number is very different from the other results. A curious result presented in this report is that the empirical quadratic growth is 75% of the expected for the arrays used.

A problem appeared when the growth was not quadratic. Sometimes the empirical value was in between the expected values for linear and $n \lg n$ which makes it harder to determine which one corresponds to the experimental value. To solve this another ratio was obtained, the ratio between t_{max}/t_{min} and $f_1(n_{max})/f_{min}$ gives a better understanding than the absolute difference between them.

$$ratio = \frac{t_{max}/t_{min}}{f_1(n_{max})/f_{min}}$$

With this relation, the following table was constructed with the approximated ratios found using the different functions as denominator.

All the values with an star near are closer to one so that behaviour will dominate if n_{max} keeps growing.

Observing that the empirical values all relate to the theoretical ones except for when InsertionSort was used with a sorted array. This was not expected because for InsertionSort a sorted array is the best case possible because the while loop inside the algorithm never runs. One explanation for this result can be that, as the best case is a sorted array, the smallest array that takes 30 ms is very big, in this case it has 10^7 elements.

f_1	f_2	f_3
165	101	0.76*
82	53	0.76*
82	53	0.75*
1.08*	0.84	0.01
1.31	1.02*	0.01
127	80	0.78*
1.29	0.81*	0
1.4	0.88*	0
1.22	0.81*	0
33	23	0.75*
1.29	0.79*	0
1.36	0.83*	0

Table 4: Ratio between experimental results and expected results

4 Conclusion

After comparing the values and analyzing the results it is possible to conclude that the sorting algorithms have very similar empirical behaviours than the ones expected. Possible error sources are the size of the arrays, there is a limit to test in personal computers and it might be hard to differentiate between n and $n \lg n$ when there is no sufficient distance between the smallest array tested to the biggest one.