GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

REVIEW ON SORTING ALGORITHMS

Computational Thinking with Algorithms

Lecturer: Dominic Carr

MAY 3, 2021

HIGHER DIPLOMA IN SCIENCE – COMPUTING (DATA ANALYTICS)
Olga Rozhdestvina
G00387844

I. Introduction

Sorting is a process of organising a list of elements in a particular order which makes handling the elements more efficient than handling randomize elements (S. Paira et al., 2014). Since it increases efficiency of each of the subsequent operations, sorting is frequently used as an intermediate step by many programs. This makes sorting a fundamental operation in computer science. Along with the rapid informational growth in our world this led to increase in the development of sorting algorithms.

Sorting algorithm is an algorithm that takes an array as input and outputs a permutation of that array that is sorted (Sorting Algorithms, n.d.). Enhancing the existing sorting algorithms or producing new algorithms can greatly optimize other algorithms. Thus, a large number of algorithms has been developed to improve sorting, each approaching the reordering of elements differently in order to increase the performance and efficiency of the practical applications (K. S. Al-Kharabsheh et al., 2013).

When comparing between various sorting algorithms, there are several factors that must be taken in consideration.

1. Time complexity.
   The time complexity of an algorithm determined the amount of time that can be taken by an algorithm to run [3][7][27]. different from sorting algorithm to another according to the size of data that we want to reorder, some sorting algorithm inefficient and too slow. The time complexity of an algorithm is generally written in form big O(n) notation, where the O represents the complexity of the algorithm and a value n represent the number of elementary operations performed by the algorithm [8].

2. Space complexity

3. Stability
   means; algorithm keeps elements with equal values in the same relative order in the output as they were in the input. [2][3][9]. Some sorting algorithms are stable by its nature such as insertion sort, merge sort, bubble sort, while some sorting algorithms are not, such as quick sort, any given sorting algorithm which is not stable can be modified to be stable [3].

performance, in-place sorting, comparator functions, comparison-based and non-comparison-based sorts, etc.

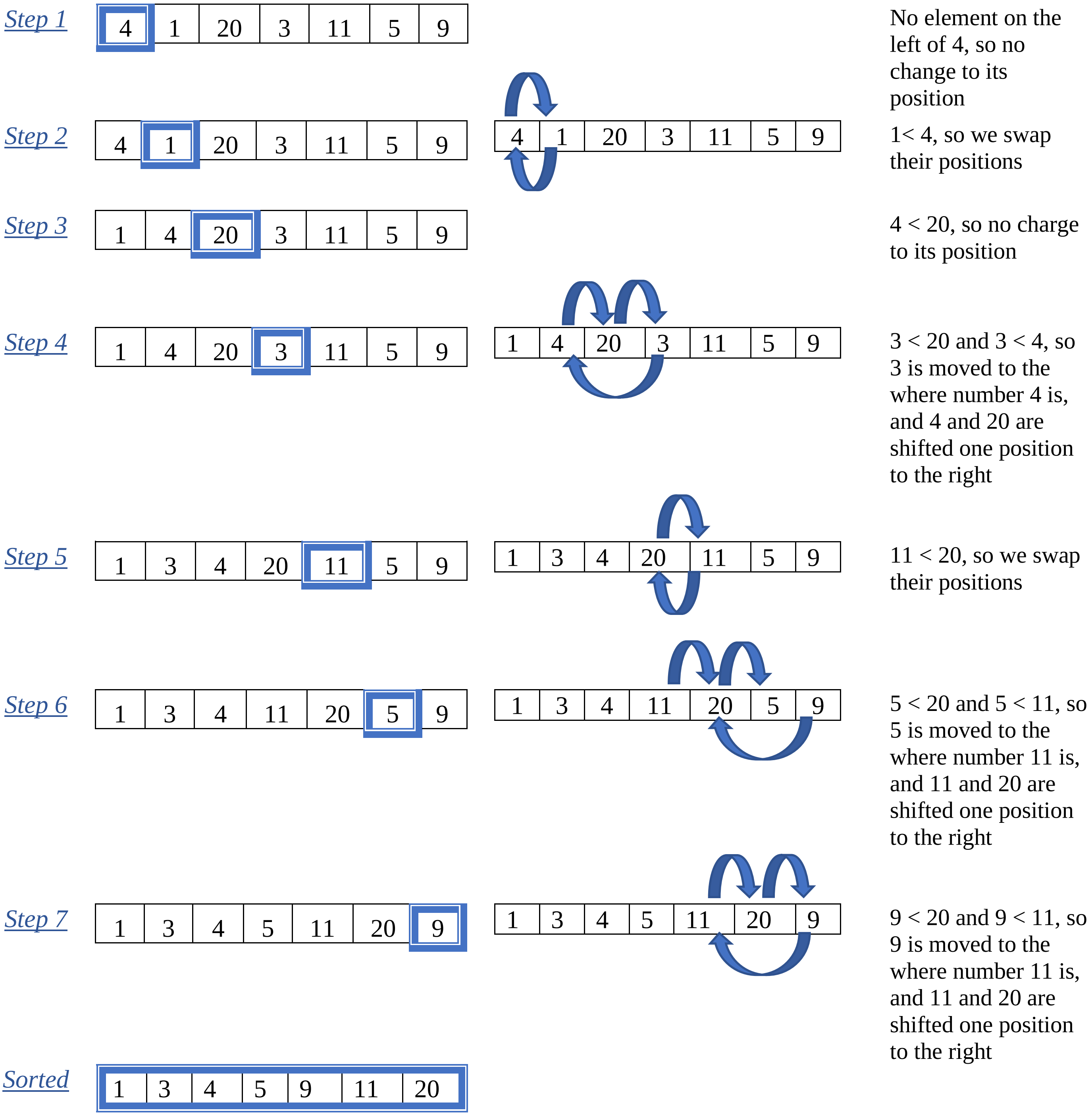
II. Five Sorting Algorithms

Introduce each of your chosen algorithms in turn, discuss their space and time complexity, and explain how each algorithm works using your own bespoke diagrams and different example input instances.

In this report I examined five sorting algorithms: Insertion Sort, Quicksort, Heap Sort, Bucket Sort and Introsort.

1. Insertion Sort

Consider below.



2. Quicksort
3. Heap Sort
4. Bucket Sort
5. Introsort

III. Implementation & Benchmarking

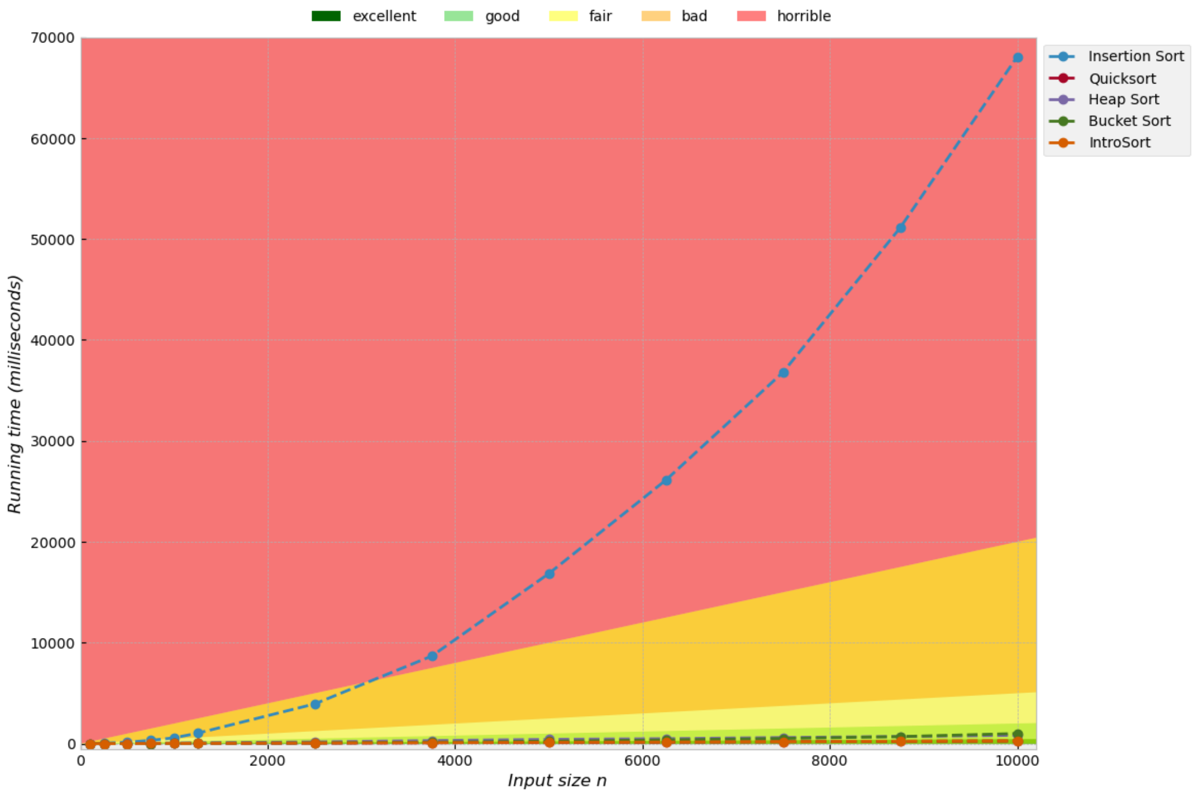
This section will describe the process followed when implementing the application above, and will present the results of your benchmarking. Discuss how the measured performance of the algorithms differed – were the results similar to what you would expect, given the time complexity of each chosen algorithm?

All five sorting algorithms (Insertion Sort, Quicksort, Heap Sort, Bucket Sort and Introsort) were implemented in Python and tested for the random sequence input of length from 100 to 10000. All five sorting algorithms were executed on machine Operating System having Intel(R) Core(TM) i7-7700HQ CPU @ 2.80 GHz and installed memory (RAM) 8.00 GB. The Plot of length of input and CPU time taken (in milliseconds) is shown in Figure 1. The result shows that for small input the performance for the five algorithms is almost nearest, but for the large input Quicksort and Introsort are the fastest and the Insertion sort the slowest.

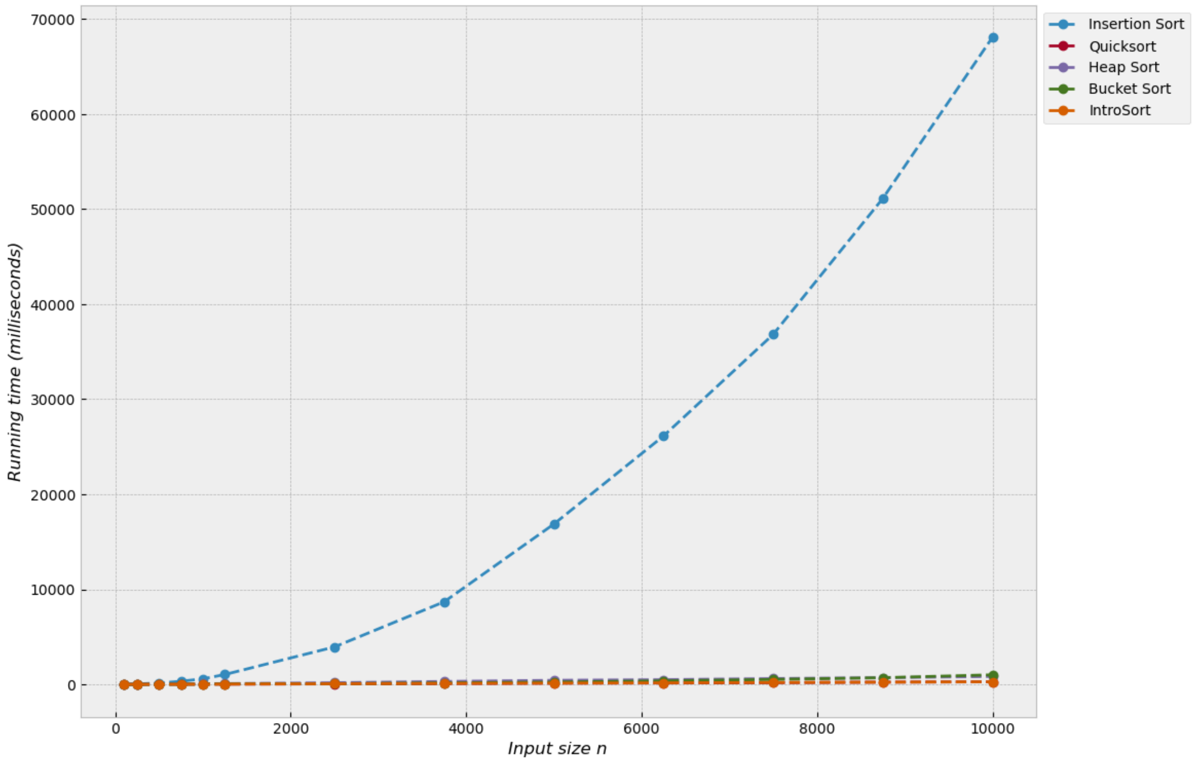
Sizes	***100***	***250***	***500***	***750***	***1000***	***1250***	***2500***	***3750***	***5000***	***6250***	***7500***	***8750***	***10000***
Insertion Sort	8.615	44.187	145.094	330.76	584.569	1038.718	3940.958	8683.307	16853.674	26118.586	36798.18	51102.482	68049.178
Quicksort	2.132	3.994	8.788	14.288	18.977	23.661	52.62	91.082	131.531	148.987	185.526	240.117	270.273
Heap Sort	3.938	14.024	29.678	49.135	71.095	83.327	174.13	309.035	428.624	495.514	612.139	708.832	836.248
Bucket Sort	1.598	4.179	10.328	16.611	22.959	47.141	76.723	150.642	242.875	371.101	519.634	703.357	1007.795
Introsort	1.674	5.043	11.098	20.134	23.068	33.364	67.835	103.779	136.952	166.592	203.662	233.689	279.067

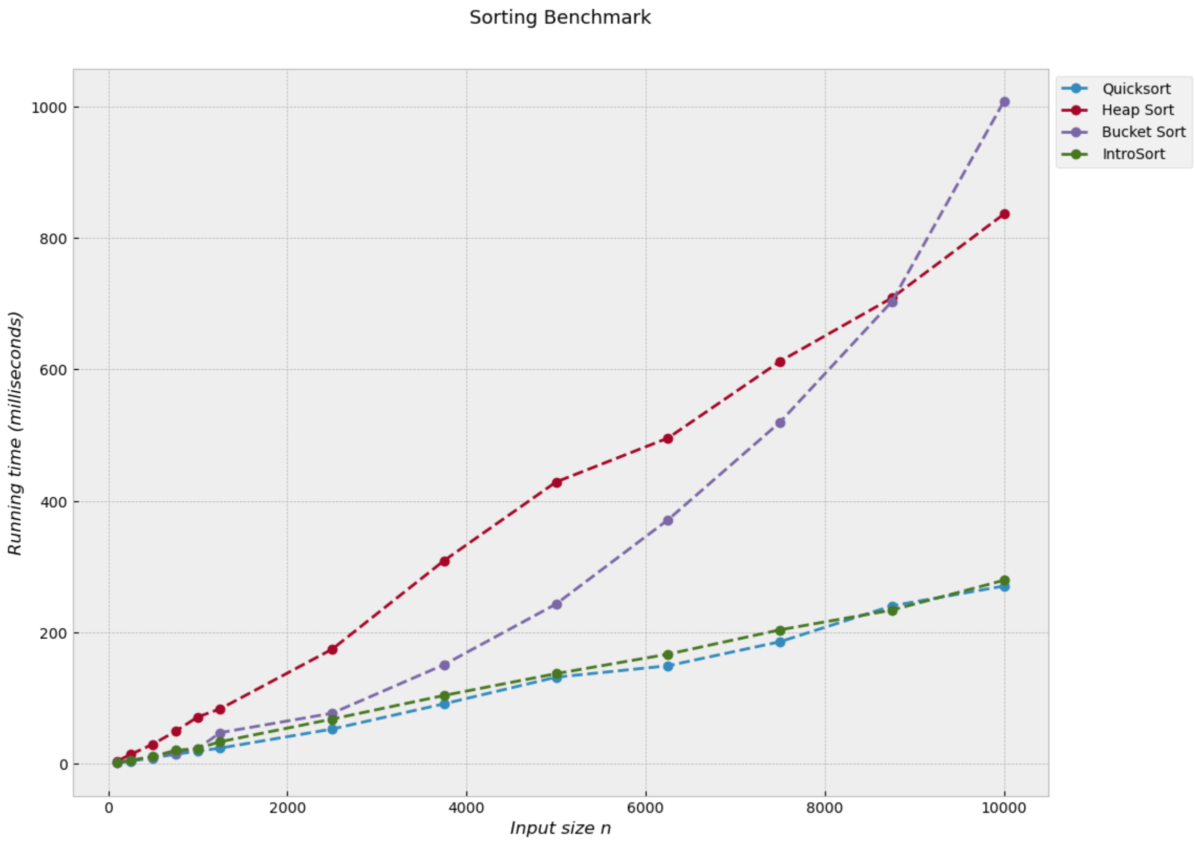
Figure 1 Running time benchmark (the average of 10 repeated runs)

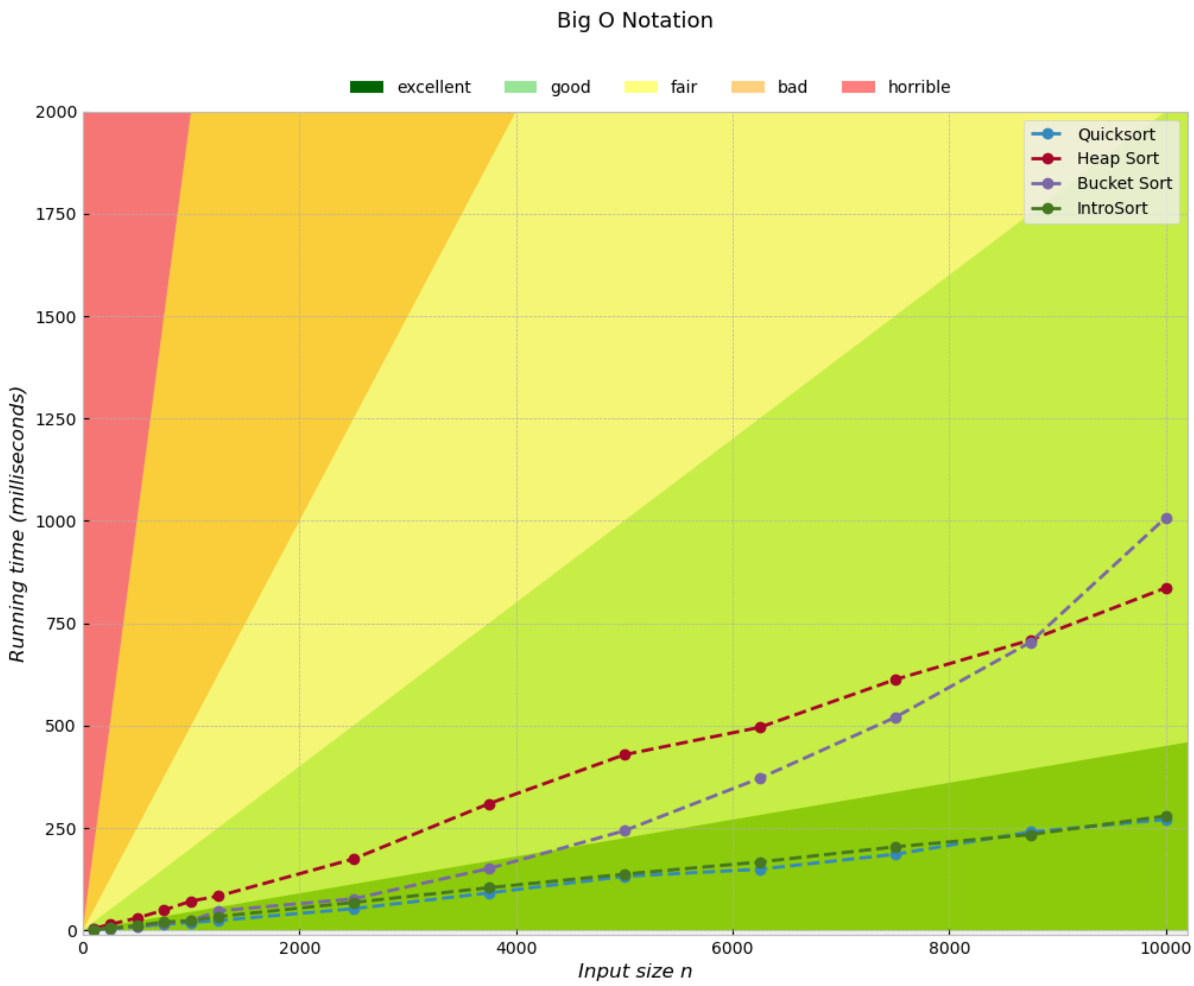
Big O Notation



Sorting Benchmark




Sorting Benchmark
Quicksort
Heap Sort
Bucket Sort
IntroSort
Running time (milliseconds)
Input size n
0
200
400
600
800
1000
0
2000
4000
6000
8000
10000


Big O Notation
excellent
good
fair
bad
horrible
Quicksort
Heap Sort
Bucket Sort
IntroSort
Running time (milliseconds)
Input size n
0
250
500
750
1000
1250
1500
1750
2000
0
2000
4000
6000
8000
10000

IV. References

1. S. Paira, S. Chnadra, A. Sk Safikul, D.S. Partha (2014) 'A Review Report on Divide and Conquer Sorting Algorithm'. Available at: https://www.researchgate.net/publication/276847633_A_Review_Report_on_Divide_and_Conquer_Sorting_Algorithm (Accessed: May 3, 2021)
2. 'Sorting Algorithms'. *Brilliant.org*. Available at: https://brilliant.org/wiki/sorting-algorithms/ (Accessed: May 3, 2021)
3. K. S. Al-Kharabsheh, I. M. AlTurani, A. M. I. AlTurani, N. I. Zanoon (2013) 'Review on Sorting Algorithms A Comparative Study'. Available at: https://www.researchgate.net/publication/259911982_Review_on_Sorting_Algorithms_A_Comparative_Study (Accessed: May 3, 2021)