INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO LICENCIATURA EM ENGENHARIA INFORMÁTICA

MDISC_1DE_G021

Team 021_Class1DE

1211131_Pedro Pereira
1211151_Alexandre Geração
1211089_José Gouveia
1211128_Tiago Oliveira

Teachers/Advisors

Alexandra Gavina (ALG)
Ana Moura (AIM)

Course Unit

Matematica Discreta

June 2022

MDISC 1

1.1 Introduction

The application should implement two sorting algorithms, to be chosen of manually by the center coordinator, that are capable of organizing, by arrival time or leaving time, the data that is imported to the system from a legacy system.

The first sorting algorithm that is implemented is called Bubble Sort Algorithm. It compares two adjacent elements of an array, and it swaps them if they are out of the wanted order. This process is then repeated until it is possible to run through the entire array and not find two elements that need to be swapped.

The implementation of the algorithm was done following the pseudocode below:

```
bubbleSort(array[1] … array[n] : date)
        for (i := 1 to n - 1)
                for (j := 1 to n - i )
                        if (array[j +1] isBefore array[j]) then
                        swapper = array[j]
                        array[j] = array[j +1]
                        array[j +1] = swapper
```

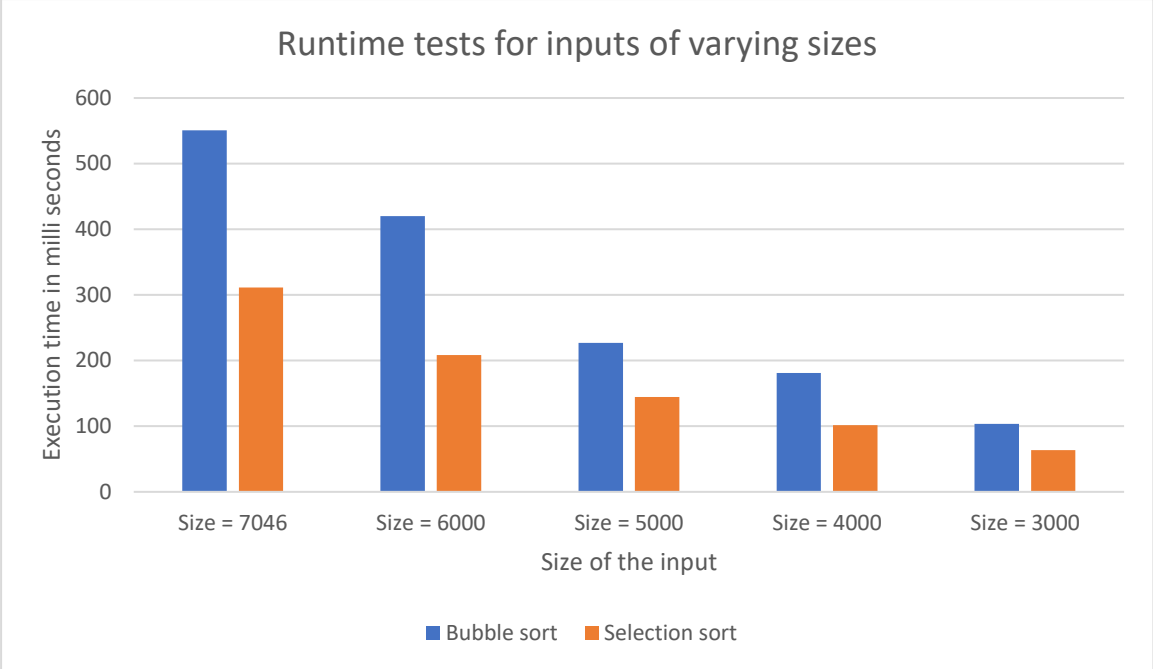
The second sorting algorithm that is implemented has the name of Selection Sort Algorithm. It selects the smallest element from the unsorted array in each iteration and places it at the beginning of the unsorted array.

The implementation of the algorithm was done following the pseudocode below:

```
selectionSort(array[1] … array[n] : date)
        for (i := 1 to n - 1)
                index = i
                for (j := i + 1 to n)
                        if (array[j] isBefore array[index]) then
                        index = j
                swapper = array[index]
                array[index] = array[i]
                array[i] = swapper
```

Each algorithm has four variations that only differ in the in order to satisfy the client's requirements, once the

1.2 Runtime tests for inputs of varying sizes



1.3 Worst-case time complexity analysis

To study the implemented algorithms, we will use the big O notation, one of the most used tools to estimate the complexity of an algorithm.

We say that $f(x)$ is $O(g(x))$ if

$$\exists c \in R^+ \; \exists k \in R^+ \; \forall x (x > k \Longrightarrow |f(x)| \leq c|g(x)|)$$

If $f(x)$ *is* $O(g(x))$ we say that $g(x)$ is the asymptotic limit of $f(x)$. This notation is used to estimate the number of operations that an algorithm will make as the size of the input increases, establishing a superior limit of that growing, this ass an advantage as we don't need to consider the multiplicity constants or small contributions (for i.e., $f(x) = ax^2$, $(f(x) = x^2 + a)$), so if we have an algorithm which makes $7n^2 + 4n$ operations (n is the number of the inputs) it will make less operations that another algorithm which makes $n^3 + 4$ operations for $n \geq 8$, all because the first one is $O(n^2)$ and the second one is $O(n^3)$, but why it is $O(n^2)$ and $O(n^3)$ respectively, if we use the following theorem:

Theorem 1:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + \; a_0 \; with \; a_i \; \in R, \forall \; i, f(x) \; is \; O(x)$$

We can conclude that $7n^2 + 4n$ is $O(n^2)$ and $n^3 + 4$ is $O(n^3)$.

How can we use this to study the temporal complexity of an algorithm? Well, the complexity of an algorithm is expressed as the number of operations (primitive operations) that it effectuates. What are the primitive operations that an algorithm can make?

- Read a value from a list

- Attribution of a value
- Comparison between two values
- Arithmetic operations
- Call a method
- Return of a method

Following its pseudocode, we will study the worst-case scenario for the Bubble Sort Algorithm:

Line	Algorithm	Complexity
1st	nA + nC	$O(n)$[1]
2nd	$\frac{1}{2}(n^2 - n)$(A + C)	$O(n^2)$
3rd	$(\frac{n^2}{2} + \frac{n}{2} - 1)$C	$O(n^2)$
4th	$\frac{n^2}{2} + \frac{n}{2} - 1$A	$O(n^2)$
5th	$\frac{n^2}{2} + \frac{n}{2} - 1$A	$O(n^2)$
6th	$\frac{n^2}{2} + \frac{n}{2} - 1$A	$O(n^2)$

- A is for attribution
- C is for comparison

By using the following theorem,

Theorem 2:

$$lets\ consider\ f_1(x)\ and\ f_2(x), funtions\ as\ f_1(x)\ is\ O\big(g(x)\big)\ and\ f_2(x)\ is\ O\big(g(x)\big)$$
$$\therefore\ f_1(x) + f_2(x)\ is\ O(\max\{|g_1(x)|, |g_2(x)|\})$$

and adding all the operations executed by the last algorithm,

$$O(n) + O(n^2) + O(n^2) + O(n^2) + O(n^2) + O(n^2) = O(n^2)$$

Then we can conclude that the worst-case scenario of the algorithm will make n^2 operations in order to give a result.

Again, following its pseudocode, we will study the worst-case scenario for the Selection Sort Algorithm:

Line	Algorithm	Complexity
1st	nA + nC	$O(n)$[1]
2nd	(n – 1)A	$O(n)$
3rd	$\frac{1}{2}(n^2 - n)$(A + C)	$O(n^2)$
4th	$\frac{n^2}{2} + \frac{n}{2} - 1$A	$O(n^2)$
5th	$\frac{n^2}{2} + \frac{n}{2} - 1$A	$O(n^2)$

6th	(n – 1)A	$O(n)$
7th	(n – 1)A	$O(n)$
8th	(n – 1)A	$O(n)$

By using the following theorem,

Theorem 2:

$$lets\ consider\ f_1(x)\ and\ f_2(x), funtions\ as\ f_1(x)\ is\ O\big(g(x)\big)\ and\ f_2(x)\ is\ O\big(g(x)\big)$$
$$\therefore\ f_1(x) + f_2(x)\ is\ O(\max\{|g_1(x)|, |g_2(x)|\})$$

and adding all the operations executed by the last algorithm,

$$O(n) + O(n^2) + O(n^2) + O(n^2) + O(n^2) + O(n^2) = O(n^2)$$

Then we can conclude that the worst-case scenario of the algorithm will make n^2 operations in order to give a result.

MDISC 2

1.1. Introduction

The application should be able to analyze the performance of a center, by implementing a brute -force algorithm. For a specific day, and time intervals in m minutes chosen by the coordinator of the center, the program creates a list of length 720/m were the i-th value is the difference between the number of new clients arriving and the number of clients leaving the center in that i-th interval

The application should implement a brute-force algorithm to determine the contiguous sub list with maximum sum

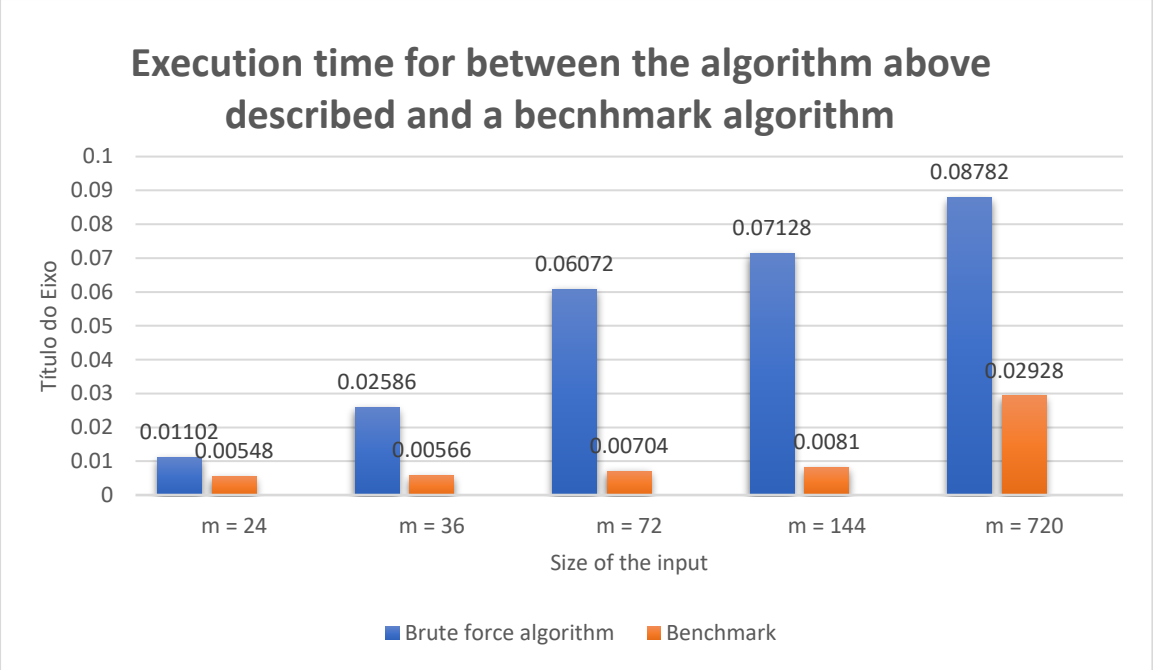
Our implementation of the brute force algorithm has the following pseudocode.

```
bruteForce(difference[1] … difference[n] : integer)
        maxSum := 0
        start := 0
        end := 0
        for (positionA := 1 to n – 1)
                sum := 0
                for (positionB := positionA to n)
                        sum = sum + difference[positionB]
                        if (sum > maxSum)
                                maxSum = sum
                                start = positionA
                                end = positionB
        pos = 0
        for (m := start to end)
                sublist[pos] := difference[m]
                pos = pos + 1
```

return sublist

1.2 Runtime tests for inputs of varying sizes

Now a time comparison between the algorithm that we implemented, and a benchmark algorithm provided for this purpose.



1.3 Worst-case time complexity analysis

To study the implemented algorithm, we will use the big O notation that was already explained in the report, in the part A.

Let's consider the previous pseudocode to see what operations are made in each line.

Line	Algorithm	Complexity
1st	1A	$O(1)$
2nd	1A	$O(1)$
3rd	1A	$O(1)$
4th	nA + nC	$O(n)$
5th	$(n-1)$A	$O(n)$
6th	$\frac{1}{2}(n^2+n)$(A + C)	$O(n^2)$
7th	$(\frac{n^2}{2}+\frac{n}{2}-1)$ (0 + A)	$O(n^2)$
8th	$(\frac{n^2}{2}+\frac{n}{2}-1)$C	$O(n^2)$
9th	$(\frac{n^2}{2}+\frac{n}{2}-1)A$	$O(n^2)$

10th	$(\frac{n^2}{2}+\frac{n}{2}-1)A$	$O(n^2)$
11th	$(\frac{n^2}{2}+\frac{n}{2}-1)A$	$O(n^2)$
12th	1A	$O(1)$
13th	nA + nC	$O(n)$
14th	$(n-1)$A	$O(n)$
15th	$(n-1)$ O + A	$O(n)$
16th	1R	$O(1)$

- A is for attribution
- C is for comparison
- R is for a return
- O is for operation

So, if we add all operations executed by the last algorithm, and apply the two explained theorems we will end up with:

$$O(1)+O(1)+O(1)+O(n)+O(n)+O(n^2)+O(n^2)+O(n^2)+O(n^2)+O(n^2)+O(n^2)+O(1)+O(n)+O(n)+O(1)=O(n^2)$$

In conclusion in the worst-case scenario the algorithm will make n^2 operations in order to give a result.