USER MANUAL

Instituto Superior de Engenharia do Porto

Project developed by:

G56_Os_Rotundas

1221219 Diogo Araújo

1220779 Simão Lopes

1220780 Tiago Alves

1221003 Tiago Santos

Annex B. MDISC

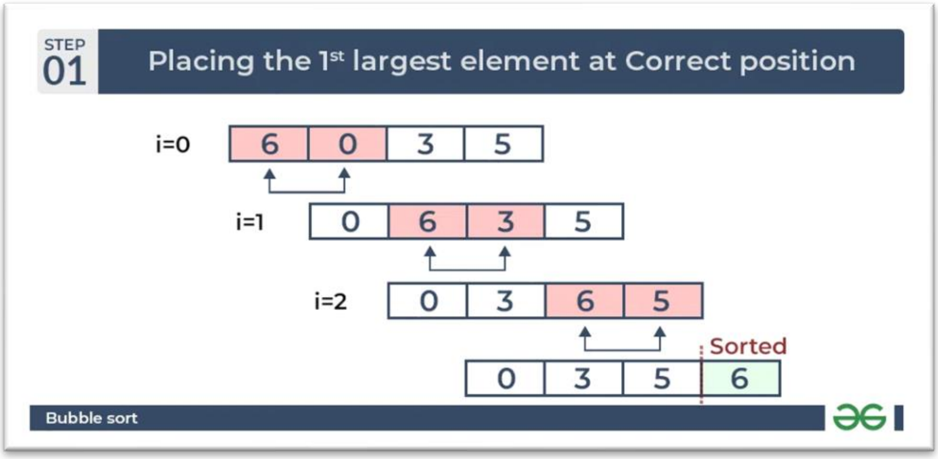
Sorting Algorithms

Introduction

The problem presented, “As a network manager, I want to list all deals made”, required the implementation of two algorithms that must sort the list containing the range of properties of the trades in ascending or descending order. The algorithms chosen were Bubble Sort, which compares adjacent elements to sort sequentially, and Selection Sort, which iteratively selects the smallest element and swaps it to the correct position. This is followed by an analysis of the study of the worst-case time complexity of these algorithms.

Bubble Sort

Bubble Sort compares the first element with the second and if the first is greater, it will swap them otherwise they stay in the same position.



Moves to the next pair comparing and swapping if they suit the condition referred before, after comparing every element, the largest element will be at the end of the list. This process will be repeated various times and at the end a list sorted will be returned.

procedure bubble sort (list [0], list [1], ..., list[n]: integers)

Iterate through the array

1. for i := 0 to n - 1

    Iterate through the unsorted part of the array

2. for j := 0 to n - i – 1

    Swaps the elements if they fit in the condition

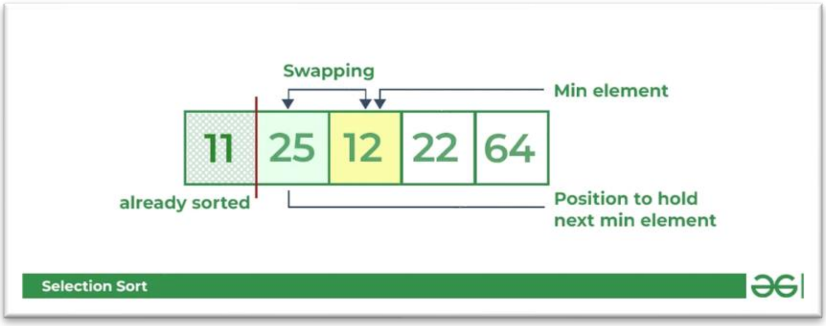
3. if list[j] > list[j+1] then swap list[j] and list[j+1]

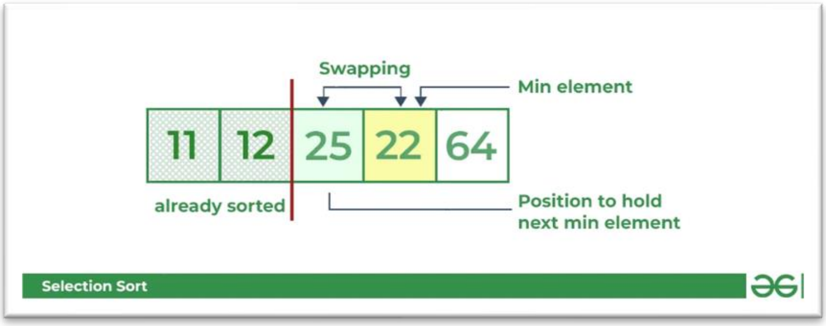
returns the list sorted

4. return list

Selection Sort

The Selection algorithm starts by finding the smallest element of the list and swaps it with the first element. Again, finds the smallest number and swaps it with the second. Here is an example:





This process will be repeated repeatedly passing by every element of the array and at the end it will be sorted.

procedure selectionSort(list[1], list[2], ..., list[n]:integers)

Iterate through the array

1. for i := 0 to len - 1
2. index := i

   Find the index of the minimum element in the unsorted portion of the array.

3. for j := i + 1 to len - 1

   Swap the current element with the minimum element.

4. if arr[j] < arr[index] then index := j

Return the list sorted

5. return list

Worst-case time complexity analysis

Bubble Sort Complexity

Line	Complexity
1	n A e n C → O(n)
2	(n^2-n) /2 A e (n^2-n) /2 C → O (n^2)
3	(n^2-n) /2 C → O (n^2)
4	1R → O (1)

After studying the complexity of every line of the Bubble Sort, we can conclude that the highest time complexity of this algorithm is O(n^2), indicating a quadratic time complexity, this means that the execution time grows quadratically with the input size.

Selection Sort Complexity

Line	Complexity
1	n A e n C → O(n)
2	n A e n C → O(n)
3	(n^2-n) /2 A e (n^2-n) /2 C → O (n^2)
4	(n^2-n) /2 A e (n^2-n) /2 C → O (n^2)
5	1R → O (1)

Upon analyzing the Selection Sort, it becomes evident that its time complexity reaches a maximum of O (n^2), signifying a quadratic relationship between execution time and input size. Like Bubble Sort, execution time will grow quadratically with the data size.

Balanced Partition Problem

Introduction

This time the problem found is:" As a network manager, I want to divide the set of all stores into two subsets so that the total number of properties of the stores between the two subsets is as small as possible." To solve this problem, a brute-force algorithm must be implemented, which consists of computing all partitions of the cardinal number two and then determining the sub lists that have the least discrepancy between the sums of their individual elements. These lists to be returned must specify the store, its corresponding number of properties and the difference between the sum of the number of properties and the execution time. Then, a table is presented with the data on the runtime tests executed, containing the main list, its subsets, a value "n" indicating the number of stores for the test, and the average time for each "n" calculated for five executions. Finally, the worst-case time complexity of the algorithm is demonstrated.

```
procedure bruteForce(arrNum[ 1 ], arrNum[ 2 ], ..., arrNum[ n ]:integers)

        list1 := integers[ n ]
        list2 := integers[ n ]
        index := divide(list)
        diff = getLists(arrNum, index, list1, list2)
    returns the inicial list, difference and both sublists
    return (arrNum, diff, list1, list2)

procedure getLists (arrNum, list1, list2, index)

    Iterate through the array
    for int i := 0 to n - 1:
        if  ((num right shift bits of set by one) Mod 2) = 1 then
            list1[ i ] := arr[ i ]
            list2[ i ] := 0
        else
            list1[ i ] := 0
            list2[ i ] := arr[ i ]
    return abs(sum(list1) - sum(list2))

procedure pow(x, y)

    int res := 1
    while y is not 0 do
        res := res * x
```

```
            y := y - 1
    return res
procedure divide(arr)

    int comb := pow(2, n - 1) - 1
    int index := -1
    int dif := -1
    Iterate through the array
    for int  i := 0 to comb - 1
       int temp := getDiff (arr, i + 1)
            if dif > temp or dif < 0 then
                    index := i + 1
                    dif := temp
    return index


procedure getDiff(arr, num):

    tmp1 := integers[ n ]
    tmp2 := integers[ n ]
    for i := 0, j := 0, k := 0 to n - 1:
            if  ((num right shift bits of set by one) Mod 2) = 1 then
                    tmp1[ j ] = arr[ i ]
            else
                    tmp0[ k ] = arr[ i ]
return abs(sum(tmp1) - sum(tmp2))

procedure abs(x):

    if x > 0 then return x
    else return -x

procedure sum(list):

    int res = 0
    for i := 0 to n -1
            res = res + i
    return res
```

Runtime tests for inputs of varying sizes

Input	Main List	LIST 1	LIST 2	Difference	Execution time (s)
3	[(9, North Pole), (28, Anchorage), (32, Phoenix)]	[(9, North Pole), (28, Anchorage), (0,)]	[(0,), (0,), (32, Phoenix)]	5	9,5084E-06
6	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo)]	[(0,), (0,), (32, Phoenix), (11, Cerritos), (0,), (0,)]	[(9, North Pole), (28, Anchorage), (0,), (0,), (5, Fremont), (2, El Segundo)]	1	5,12082E-05
9	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando)]	[(9, North Pole), (0,), (0,), (11, Cerritos), (5, Fremont), (0,), (69, Denver), (0,), (0,)]	[(0,), (28, Anchorage), (32, Phoenix), (0,), (0,), (2, El Segundo), (0,), (20, Lauderdale), (12, Orlando)]	0	0,000169009
12	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis)]	[(9, North Pole), (28, Anchorage), (0,), (11, Cerritos), (0,), (0,), (69, Denver), (0,), (0,), (0,), (0,), (0,)]	[(0,), (0,), (32, Phoenix), (0,), (5, Fremont), (2, El Segundo), (0,), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis)]	1	0,000403517

15	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte)]	[(9, North Pole), (0,), (32, Phoenix), (11, Cerritos), (5, Fremont), (0,), (69, Denver), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(0,), (28, Anchorage), (0,), (0,), (0,), (2, El Segundo), (0,), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte)]	1	0,0040143
18	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester)]	[(9, North Pole), (28, Anchorage), (32, Phoenix), (0,), (0,), (0,), (69, Denver), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(0,), (0,), (0,), (11, Cerritos), (5, Fremont), (2, El Segundo), (0,), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester)]	1	0,0186891

21	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston)]	[(9, North Pole), (0,), (32, Phoenix), (11, Cerritos), (0,), (0,), (69, Denver), (20, Lauderdale), (12, Orlando), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(0,), (28, Anchorage), (0,), (0,), (5, Fremont), (2, El Segundo), (0,), (0,), (0,), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston)]	0	0,088564317
24	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission)]	[(0,), (28, Anchorage), (32, Phoenix), (11, Cerritos), (0,), (0,), (69, Denver), (20, Lauderdale), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(9, North Pole), (0,), (0,), (0,), (5, Fremont), (2, El Segundo), (0,), (0,), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission)]	1	0,557341458

27	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission), (8, Columbus), (35, Portland), (37, Las Vegas 2)]	[(0,), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (0,), (69, Denver), (20, Lauderdale), (0,), (35, Chicago), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(9, North Pole), (0,), (0,), (0,), (0,), (2, El Segundo), (0,), (0,), (12, Orlando), (0,), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission), (8, Columbus), (35, Portland), (37, Las Vegas 2)]	1	4,48354745
30	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (2, Columbia), (5, Charlotte), (12, Las Vegas), (6, New York), (6, West Chester),	[(9, North Pole), (28, Anchorage), (32, Phoenix), (11, Cerritos), (5, Fremont), (2, El Segundo), (69, Denver), (20, Lauderdale), (12, Orlando), (35, Chicago), (3, Atlanta), (7, Indianapolis), (11, Boston), (0,), (5, Charlotte), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,)]	[(0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (0,), (2, Columbia), (0,), (12, Las Vegas), (6, New York), (6, West Chester), (6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission), (8, Columbus), (35, Portland), (37, Las Vegas 2), (15, Boston 2), (41,	1	36,75089345

	(6, Memphis), (13, Nashville), (12, Houston), (5, San Antonio), (2, Seattle), (8, Mission), (8, Columbus), (35, Portland), (37, Las Vegas 2), (15, Boston 2), (41, Tracy), (42, Fairdale)]		Tracy), (42, Fairdale)]		

In this table, from left to right, it is represented the input size used for each runtime test, the full list and the two sub lists that the algorithm determined, the difference between the sum of properties of the sub lists and the average time in seconds, defined by five executions.

Here is the calculation of the average times used in the previous table:

Average Times

Input, value of n	Times for each test(s)	Average Time (s)
3	8,17E-06 7,46E-06 9,50E-06 1,14E-05 1,10E-05	9,5084E-06
6	4,86E-05 5,37E-05 4,82E-05 6,88E-05 3,69E-05	5,12082E-05
9	2,73E-04 7,03E-05 2,50E-04 1,07E-04 1,45E-04	0,000169009
12	8,13E-04 1,91E-04 3,41E-04 3,92E-04	0,000403517

	2,80E-04	
15	0,00352475 0,002379417 0,00763875 0,004116833 0,00241175	0,0040143 0,
18	0,023272291 0,016447584 0,019182916 0,013563667 0,020979042	0,0186891
21	0,086563875 0,087697125 0,094489833 0,089959375 0,084111375	0,088564317
24	0,561946208 0,534822208 0,555776083 0,578093625 0,556069167	0,557341458
27	4,458059458 4,502668917 4,478140584 4,4711675 4,507700792	4,48354745
30	36,61970596 36,72753667 36,60346838 36,6833105 37,12044575	36,75089345

This graphic represents the time that the algorithm took to be executed in order of the values of “n” which is the number of stores, from three to thirty. A logarithmic scale was used to make the graph more perceptible because, with the final values being much larger than the rest, a line overlapping the x-axis was visible until n = 18 or n = 21.

Worst-case time complexity analysis

Line	Complexity
1	1A → O (1)
2	1A → O (1)
3	1A and 1 method call → O (1)
4	1A and 1 method call → O (1)
5	1R → O (1)
6	n A e n C → O(n)
7	n A e n C → O(n)
8	n A e n C → O(n)
9	n A e n C → O(n)
10	n A e n C → O(n)
11	n A e n C → O(n)

12	n A e n C → O(n)
13	1R → O (1)
14	1A → O (1)
15	n-1C → O (n)
16	n-1A → O (n)
17	n-1A → O (n)
18	1R → O (1)
19	1A and 1 method call → O (1)
20	1A → O (1)
21	1A → O (1)
22	2^n → O (2^n)
23	2^n → O (2^n)
24	2^n → O (2^n)
25	2^n → O (2^n)
26	2^n → O (2^n)
27	1R → O (1)
28	2^n → O (2^n)
29	2^n → O (2^n)
30	2^n → O (2^n)
31	2^n → O (2^n)
32	2^n → O (2^n)
33	2^n → O (2^n)
34	2^n → O (2^n)
35	2^n → O (2^n)
36	2^n → O (2^n)

37	2^n → O (2^n)
38	2^n → O (2^n)
39	2^n → O (2^n)
40	2^n → O (2^n)
41	2^n → O (2^n)

After analyzing table which studies the complexity of each line of the Brute Force algorithm and, with that, we can conclude that the complexity of the algorithm is O (2^n).