

Department of Computer Science

Second Semester, 2020/2021

COMP 2421

Project #4 (Sorting Algorithms)

### \*\*\*\*\*\*\*\*

### Student’s Name: Mahmoud Samara

### *Student’s No.:* *1191602*

### Section: 4

***Instructor:*** ***Dr. Radi Jarrar***

### \*\*\*\*\*\*\*\*

**TABLE OF CONTENTS**

**I: Introduction ……………………………………………………… 1**

**II: Sorting Algorithms …………………….……………………… 2**

1. Gnome Sort ………………………………………………………. 2

1.1 Information and facts about Gnome Sort ……………………. 2

1.2 Algorithm of Gnome Sort ….................................................... 2

1.3 The discussion of Algorithm of Gnome Sort ............................ 2

1.4 Time and space complexity of Gnome Sort …………………. 3

1.5 Explanation the running time of Gnome Sort …......................... 3

2. Stooge Sort ………………………………………………………… 4

2.1 Information and facts about Stooge Sort…................................. 4

2.2 Algorithm of Stooge Sort ……………………………………… 4

2.3 The discussion of Algorithm of Stooge Sort ............................ 4

2.4 Time and space complexity of Stooge Sort………………….. 5

2.5 Explanation the running time of Stooge Sort …………………. 5

3. Cocktail Sort …................................................................................. 6

3.1 Information and facts about Cocktail Sort ….............................. 6

3.2 Algorithm of Cocktail Sort …..................................................... 6

3.3 The discussion of Algorithm of Cocktail Sort ............................ 6

3.4 Time and space complexity of Cocktail Sort …………………… 7

3.5 Explanation the running time of Cocktail Sort …......................... 7

**III: Summary ……………………………………………………......... 8**

**IV: References ………………………………………………………... 9**

**I: INTRODUCTION**

A sorting algorithm is an algorithm that arranges items in a list in a specific order, the most common of which are numerical and lexicographical. The Sorting Algorithm is used to rearrange elements in a data structure or list using a comparison operator on the elements to determine the new order of the elements in the data structure. All sorting algorithms share the goal of outputting a sorted list, but the way that each algorithm goes about this task can vary. More formally, the output of any sorting algorithm must satisfy two conditions:

1. The output is in nondecreasing order (each element is no smaller than the previous element according to the desired total order). 2. The output is a permutation (a reordering, yet retaining all of the original elements) of the input.

In my report, I will explain and discuss three sorting algorithms by explaining how the algorithm work and give an example, running time in different situations and time complexity and space complexity in various cases. **[1]**

Page 1

**II: SORTING ALGORITHMS**

1. **Gnome Sorting:**
   1. **Information and facts about Gnome Sort:**

Gnome Sort is a simple sorting where the key idea is to swap adjacent elements (if not in order) to sort the entire list. It is similar to Insertion sort except the fact that in this case, we swap adjacent elements. The algorithm finds the first place where two adjacent elements are in the wrong order and swaps them. -------------------------------------------------------------------------------------------------

**1.2 Algorithm of Gnome Sort and the discussion of it:**

void gnome\_sort(int \*array, int size){

int i , tmp;

for(i=1; i<size; ){

if(array[i-1] <= array[i])

++i;

else{

tmp = array[i];

array[i] = array[i-1];

array[i-1] = tmp;

--i;

if(i == 0)

i = 1;

} }

}

// [2]

**1.3 The discussion of Algorithm of Gnome Sort (STEPS + EXAMPLE):**

**How it works (Steps):**

1. Go to right element if you're at the beginning of the array (from arr[0] to arr[1]).

2. Go one-step right if the current array element is greater or equal to the previous array element.

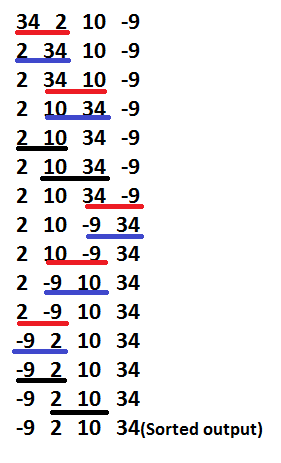
3. Switch these two elements and go one-step backwards if the new array element is smaller than the previous array element.

4. Repeat steps two and three before 'i' hits the array's end (i.e. 'n-1').

5. If the array's end is reached, the program will terminate and the array will be sorted.

Page 2

**Example of of Algorithm of Gnome Sort work:**



**Figure 1: Gnome Sort Example**

**1.4 Time and space complexity of Gnome Sort**

**►► Best Case** Complexity is: O (n).

**►► Average Case** Complexity: O (n^2).

**►► Worst Case** Complexity: O (n^2).

**►► Space complexity** is: O (1) auxiliary (no extra space).

►► Gnome Sort is: an in-place algorithm. ►► Gnome Sort is: stable algorithm so it does not require any extra storage. **[3]**

---------------------------------------------------------------------------------------------------

**1.5 Explanation the running time of Gnome Sort:**

**Explain the running time** of the **Gnome Sort** in the following situations if the input data array is:   
A) Sorted (ascending) 🡺 O (n).

B) Sorted (descending) 🡺 O (n^2).

C) Not sorted: 🡺O (n^2).

When the elements are random then the sorting is exactly similar to Bubble sort which has complexity of O (n^2) and it is the same thing when they are sorted descending in this case the worst case will appear and it will be O (n^2). Now if the elements are sorted or nearly sorted in ascending order then this sort skips the comparisons between the adjacent elements. This skipping of comparisons makes it tend towards O (n). **[3]**

Page 3

**2. Stooge Sorting:**

**2.1 Information and facts about Stooge Sort**

Stooge Sort is a recursive sorting algorithm. It is inefficient but interesting sorting algorithm and it is notable for its exceptionally bad [time complexity](https://en.wikipedia.org/wiki/Time_complexity) . It divides the array into two overlapping parts (2/3 each). The key idea is that sorting the overlapping part twice exchanges the elements between the other two sections accordingly. However, It is however more efficient than [**Slow sort**](https://en.wikipedia.org/wiki/Slowsort). The name comes from [The Three Stooges](https://en.wikipedia.org/wiki/The_Three_Stooges). **[4]**

----------------------------------------------------------------------------------------------------

**2.2Algorithm of Stooge Sort and the discussion of it:**

void stoogesort(int arr[], int i, int j)

{

int temp, k;

if (arr[i] > arr[j])

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

if ((i + 1) >= j)

return;

k = (int)((j - i + 1) / 3);

stoogesort(arr, i, j - k);

stoogesort(arr, i + k, j);

stoogesort(arr, i , j - k);

}

**2.3 The discussion of Algorithm of Gnome Sort (STEPS + EXAMPLE):**

**How it works (Steps):**

1. In the array if the value at the start is larger than the value at the end, swap them.
2. Recursively sort the first 2/3 part of the array.
3. Recursively sort the last 2/3 part of the array.
4. Again sort the first 2/3 part of the array.
5. The array becomes finally sorted. **[5]**

Page 4

**Example of Algorithm of Stooge Sort work:**

INPUT ARRAY: 2, 4, 5, 3, 1 🡪 OUTPUT ARRAY: 1, 2, 3, 4, 5

A) First and last element is compared and if last is greater than first then they are swapped. So swap 2 and 1. **(2 4 5 3 1 🡺 1 4 5 3 2)**

B) Now, recursively sort initial 2/3rd of the elements that is number (3).

**[1 4 5] 3 2 🡺 1 3 4 5 2**

C) Then, recursively sort last 2/3rd of the elements.

**1 3 [4 5 2] 🡺 1 3 2 4 5**

D) Again, sort the initial 2/3rd of the elements to confirm final data is sorted.

THE OUTPUT: (1 2 3 4 5) **[5]**

-------------------------------------------------------------------------------------------------

**2.4 Time and space complexity of Stooge Sort**

**►► Best Case** Complexity is: O (n log (3) /log (1.5)) = O (n2.709).

**►► Average Case** Complexity: O (n log (3) /log (1.5)) = O (n2.709).

**►► Worst Case** Complexity: O (n log (3) /log (1.5)) = O (n2.709).

**►► Space complexity** is: O (n).

►► Stooge Sort is: it is not in a place algorithm. ►► Stooge Sort is: unstable algorithm it is because the elements with identical values do not appear in the same order in the output array as they were in the input array. **[5]**

---------------------------------------------------------------------------------------------------

**2.5 Explanation the running time of Stooge Sort:**

**Explain the running time** of the **Stooge Sort** in the following situations if the input data array is:

A) Sorted (ascending) 🡺 O(1) ----> There will be no swap in a sorted ascending array since all of the numbers will already be at there places, but there will be some time complexity in comparing all of the numbers, which will be O(1) time complexity based on the steps of Stooge sort that I previously described.

B) Sorted (descending) 🡺 O (n log(3) /log(1.5)) = O(n2.709)----> All of the numbers in a descending array will be swapped. The code will sort the first 2/3 of the elements, then the last 2/3 of the elements, and finally the first 2/3 of the elements to validate, resulting in an O (n2.709) time complexity based on the Stooge sort steps that I previously discussed.

C) Not sorted: 🡺 O(n log(3) /log(1.5)) = O(n2.709) ----> The case for a normal array is similar, in that a normal array may be in any order, and we must always assume the worst case. So, based on the steps of Stooge sort that I explained previously, the time complexity for normal array would be the same as for descending array, resulting in an O(n2.709) time complexity.

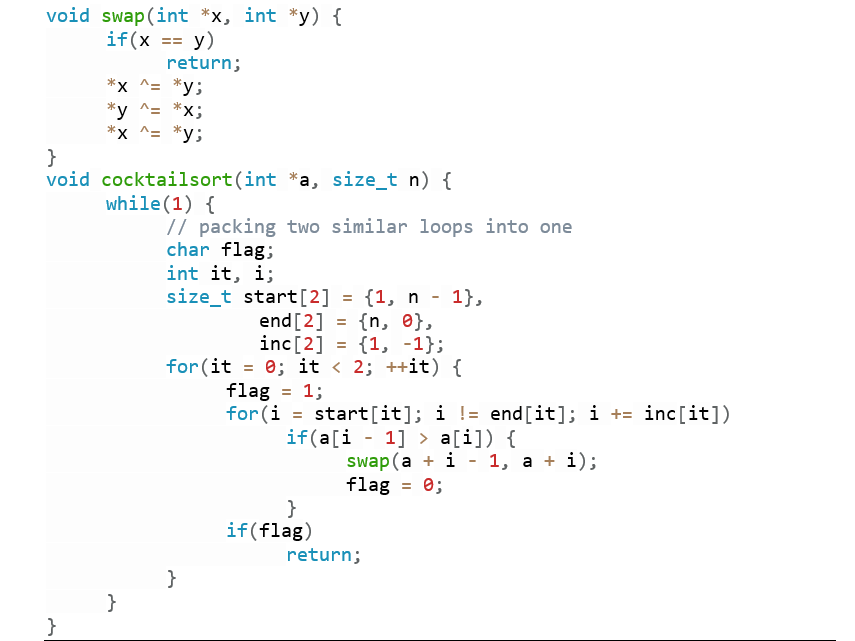
Page 5

**3. Cocktail Sorting:**

**3.1 Information and facts about Cocktail Sort**

Cocktail Sort,also known as bidirectional bubble sort(which can also refer to a variant of [selection sort](https://en.wikipedia.org/wiki/Selection_sort)).Cocktail sort traverses through a given array in both directions alternatively putting the largest and smallest elements in their positions in the first iteration, then the second largest and the second smallest in the second iteration and so on. **[6]**

--------------------------------------------------------------------------------------------------

**3.2 Algorithm of Cocktail Sort and the discussion of it:** 

**3.3 The discussion of Algorithm of Cocktail Sort (STEPS + EXAMPLE):**

**How it works (Steps):**

1. The first step, similar to the Bubble Sort, loops across the array from left to right. During the loop, neighboring items are compared, and values are exchanged if the value on the left is greater than the value on the right. The greatest number will be at the end of the array at the end of the first iteration.

2. The second stage loops through the array in the reverse direction, starting from the item immediately preceding the most recently sorted item and returning to the beginning. Similarly, adjacent items are compared and, if necessary, switched. **[7]**

Page 6

**Example of Cocktail Sort:**

INPUT ARRAY (8 2 3 1 9) 🡪 OUTPUT ARRAY : (1 2 3 8 9)

**Forward pass:** Compare  8 with 2; 8>2 → swap(8,2)   A={2,8,3,1,9}   Compare 8 with 3; 8>3 → swap(8,3)    A={2,3,8,1,9}   Compare 8 with 1; 8 > 1 → swap(8,1)   A = {2,3,1,8,9}    Compare 8 with 9; 8<9 → **do** not swap

**Backward pass:** Compare  8 with 1; 8> 1 → **do** not swap  A={2, 3, 1, 8, 9 }   Compare  1 with 3 ; 3>1 → swap(1,3)    A={2, 1, 3, 8, 9 }   Compare  1 with 2 ; 2> 1 → swap(1,2)   A={1, 2, 3, 8, 9}  **[8]** -------------------------------------------------------------------------------------------------

**3.4 Time and space complexity of Cocktail Sort**

**►► Best Case** Complexity is: O (n). It occurs when the array is almost sorted

**►► Average Case** Complexity: O (n^2).

**►► Worst Case** Complexity O (n^2).

**►► Space complexity** is: O (1).

►► Cocktail Sort is: an in-place algorithm. ►► Cocktail Sort is: stable algorithm so it does not require any extra storage.

---------------------------------------------------------------------------------------------------

**3.5 Explanation the running time of Cocktail Sort:**

**Explain the running time** of the **Stooge Sort** in the following situations if the input data array is:

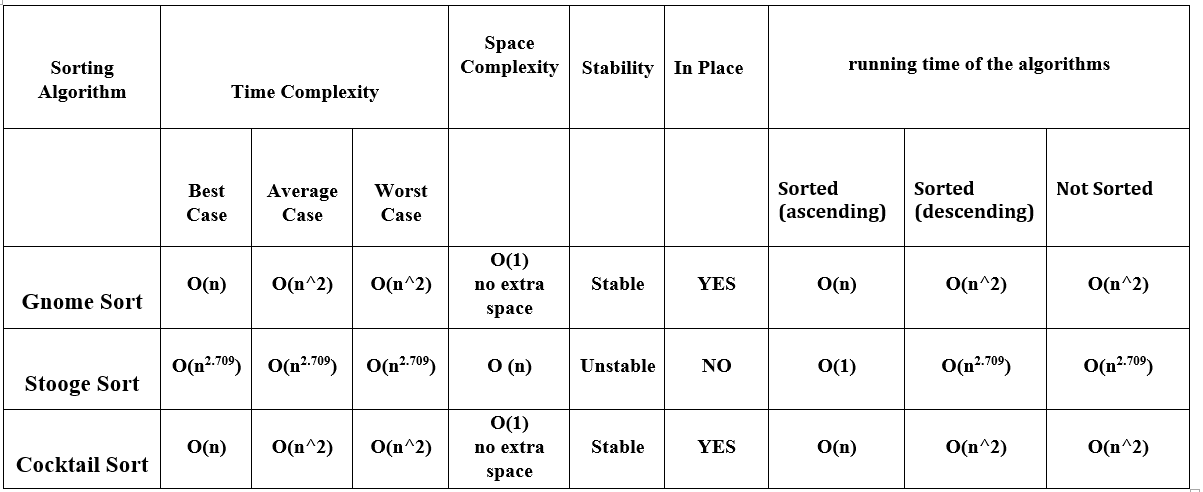
A) Sorted (ascending) 🡺 O (n) 🡪 In the case of the ascending, there will be no switching since the list has already been sorted. So, after the for loop, we simply check whether at least one swapping has been performed or not; if not, the array has already been sorted, and the time complexity will be of order O (n).

B) Sorted (descending) 🡺 O (n^2) 🡪 Since the largest element will go to the last index in the first forward pass. Also the smallest element will go to the first place in the first backward pass, the cumulative iteration will be of order (n ^ 2) / 2 in the worst case when using big O notation, but the complexity will be O (n^2).

C) Not sorted : 🡺 O (n^2) 🡪 Since the array is not sorted (original array).So the function will work on all elements of array so the largest element will go to the last index in the first forward pass. Also the smallest element will go to first place in the first backward pass. So it is O (n^2).

Page 7

**III: SUMMARY FOR THE THREE KINDS:**



From the previous explanation and the summary table, we can note many things, so we can conclude that:

1. The stooge sort is the slowest kind of all three kinds depending on the time complexity and it has something unique that its time complexity is constant in all time complexity cases. So it is not very used in the programs.
2. In the running time of the algorithms we can see that Stooge sort is the fastest when data in the array are sorted (ascending) since there will be no swap in a sorted ascending since all of the numbers will already be at there places. Moreover, both Gnome sort and Cocktail sort are the same time in the same situation.

Page 8

**IV: References:**

[1]: <https://en.wikipedia.org/wiki/Sorting_algorithm> (accessed on 22/5 at 11:00 AM).

[2]: <https://www.w3resource.com/c-programming-exercises/searching-and-sorting/c-search-and-sorting-exercise-10.php> (accessed on 22/5 at 11:15 AM).

[3]: <https://www.geeksforgeeks.org/gnome-sort-a-stupid-one> (accessed on 22/5 at 11:25 AM).

[4]: <https://en.wikipedia.org/wiki/Stooge_sort> (accessed on 22/5 at 12:05 PM).

[5]: <https://iq.opengenus.org/stooge-sort/> (accessed on 22/5 at 12:30 PM).

[6]: <https://en.wikipedia.org/wiki/Cocktail_shaker_sort> (accessed on 24/5 at 7:00 PM).

[7]: <https://www.geeksforgeeks.org/cocktail-sort> (accessed on 24/5 at 6:20 PM).

[8]: <https://www.javatpoint.com/cocktail-sort> (accessed on 24/5 at 8:30 PM).

Page 9