

DATA STRUCTURE (COMP242)

4th Research Project

Sorting Algorithms

* Teacher: DR.Mamoun Nawahdah
* Student: Shareef Nawabit 1142051

|  |
| --- |
| Contents: |
| 1. Bitonic Sort
 |
| 1. Comb Sort
 |
| 1. Cocktail Sort
 |
| 1. Cycle Sort
 |
| 1. Gnome Sort
 |

**Bitonic Sort**

* **Bitonic Sort as Merge Sort using array implementation.**
* **Main Idea:**

**If we have an array that has a bitonic sequence, and bitonic sequence is the half of array is in ascending order and the other half is descending order.**

* **Java Code:**

****

* **Time Complexity:**
* **Best Case Performance: O(log**(*n)*2**)**
* **Worst Case Performance: O(log**(*n)*2**)**
* **Average Case Performance: O(log**(*n)*2**)**
* **Space Complexity:**
* **Worst Case Space Complexity: O(n log**(*n)*2**)**
* **It is not In-place. Which means it uses extra memory. Stable.**
* **Example:**

**Let’s take [11, 13, 16, 35, 15, 4, 3, 1] as an example:**

**The first 4 elements is in ascending order and the rest is in descending order**

1. **Compare the two sets:**

**[11, 4, 3, 1, 15, 13, 16, 35]**

1. **Compare the two sets every one as two sets:**

 **[3, 1, 11, 4, 15, 13, 16, 35]**

1. **Compare the two sets every two element:**

**[1, 3, 4, 11, 13, 15, 16, 35]**

1. **Final result, SORTED.**

**Comb Sort**

* **Comb sort improves shell sort. Which means it is using array implementation.**
* **Main Idea:**

**Is that depends on delete the small values around end of the array, which make bubble sort slower.**

* **Java Code:**



* **Time Complexity:**
* **Best Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O**(n)
* **Worst Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O(***n*2**)**
* **Average Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**:  O**(*n*2)
* **Space Complexity:**[**Worst case space complexity**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O(1)**
* **It is in-place. Does not need extra memory.**
* **Example: Let’s take the following array of names as an example:
[10, 5, 8, 22, 3, 0]**

**The next shows steps of sorting the array using comb sort:**

1. **Gap = 3// [10, 5, 8, 22, 3, 0]**
2. **[10, 3, 8, 22, 5, 0]**
3. **[10, 3, 0, 22, 5, 8]**
4. **Gap = 2// [0, 3, 10, 22, 5, 8]**
5. **[0, 3, 10, 22, 5, 8]**
6. **[0, 3, 5, 22, 10, 8]**
7. **[0, 3, 5, 8, 10, 22]**
8. **Gap = 1 // [0, 3, 5, 8, 10, 22]**
9. **Sorted :)**

**Cocktail Sort**

* **Cocktail sort improves bubble sort. That means it is using**
* **Main Idea:**

**Is that this sort is using bubble sorting in each direction as it pass in the array.**

* **Java Code:**

****

* **Time Complexity:**
* **Best Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O(n)**
* **Worst Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O**(*n*2)
* **Average Case** [**performance**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**:  O**(*n*2)
* **Space Complexity:**[**Worst case space complexity**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O(1)**
* **It is in-place. Does not need extra memory.**
* **Example:
 Let’s take this set of numbers as (30, 20, 50, 40, 10) an example. Sorting it will take one pass of cocktail sort to become sorted, but if using an ascending**[**bubble sort**](http://en.wikipedia.org/wiki/Bubble_sort)**would take four passes. However one cocktail sort pass should be counted as two bubble sort passes. Which means that cocktail is faster than bubble sort. The following shows how it works with this example:**
1. **(20, 30, 50, 40, 10)**
2. **(10, 20, 30, 50, 40)**
3. **(10, 20, 30, 40, 50)**
4. **Sorted.**

**Cycle Sort**

* **Cycle sort is using array implementation.**
* **Main Idea: is that**[**permutation**](http://en.wikipedia.org/wiki/Permutation)**to be sorted can be factored into**[**cycles**](http://en.wikipedia.org/wiki/Cyclic_permutation)**, which is rotated to give a sorted result. Sort an array in place and return the number of writes.**
* **Java Code:** 
* **Time Complexity:**
* **Best Case Performance:** O(*n*2)
* **Worst Case Performance:** O(*n*2)
* **Average Case Performance:** O(*n*2)
* **Space Complexity:**[**Worst case space complexity**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**:**
* O(*n*) total
* O(*1*) auxiliary
* **It is in-place. And unstable.**
* **Example:**

**Let’s take [11, 2, 6, 12, 8] as an example:**

 **[11, 2, 6, 11, 12]**

**[11, 2, 8, 11, 12]**

**[11, 6, 8, 11, 12]**

**[2, 6, 8, 11, 12]**

**[2, 6, 8, 11, 12]**

**Writes: 5**

**Gnome Sort**

* **Gnome sort called stupid sort at first but changed to gnome sort, it is similar to insertion sort with some differences.**
* **Main Idea:**

**Compare every two elements and insert the element in its right place.**

* **Java Code:**

 ****

* **Time Complexity:**
* **Best Case Performance: O(n)**
* **Worst Case Performance: O(n^2)**
* **Average Case Performance: O(n^2)**
* **Space Complexity:**[**Worst case space complexity**](http://en.wikipedia.org/wiki/Best%2C_worst_and_average_case)**: O(1)**
* **It is in-place. Does not need extra memory.**
* **Example: take following array as an example [10, 5, 2, 9 ]. And to sort it see following:**
1. **[5, 10, 2, 9]**
2. **[5, 2, 10, 9]**
3. **[2, 5, 10, 9]**
4. **[2, 5, 9, 10]**

**Comparing All Method Sort:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Worst time complexity** | **Best time complexity** | **Average time complexity** | **Worst space complexity** |
| **Comb** | **O**(*n*2) | **O**(*n*) | **O**(*n*2) | **O**(*1*) |
| **Cocktail** | **O**(*n*2) | **O**(*n*) | **O**(*n*2) | **O**(*1*) |
| **Bitonic** | **O(log**(*n)*2**)** | **O(log**(*n)*2**)** | **O(log**(*n)*2**)** | **O(n log**(*n)*2**)** |
| **Gnome** | **O**(*n*2) | **Ω(n)** | **O(n^2)** | **O(1)** |
| **Cycle** | O(*n*2) | O(*n*2) | O(*n*2) | O(*n*) total |
| O(*1*) auxiliary |

* **The best time complexity is for Gnome and Cocktail Sorts.**
* **The best between them in the worst case is comb sort.**
* **As we know now the space complexity isn’t a case. It is very speed.**

**References:**

1. **Wikipedia / Algorithms Sorting / Comb Sort.**
2. **Wikipedia / Algorithms Sorting / Cocktail Sort.**
3. **Wikipedia / Algorithms Sorting / Gnome Sort.**
4. **Wikipedia / Algorithms Sorting / Cycle Sort.**
5. **Wikipedia / Algorithms Sorting / Bitonic Sort.**
6. **Sorting Algorithms Animation.**