Faculty of Engineering & Technology
Electrical & Computer Engineering Department

COMP2321

Research Report 2

Sorting Algorithms

Prepared by: Tareq Shannak

ID Number: 1181404

Instructor: Dr. Radi Jarar

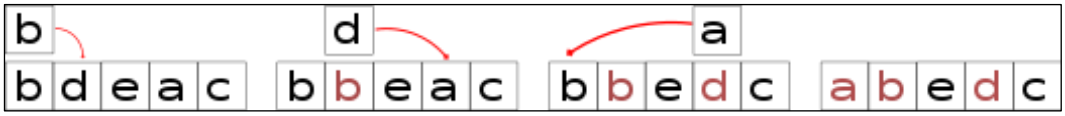
Section Number: 3

Date: 17/5/2020

Cycle Sort

Algorithm

This type of sort can give us a sorted list of distinct elements. For the first element in the list, count how many elements are less than this element, and this element must be in location equals to counter, do nothing if it is equals and check the next element, otherwise, replace the element to the suitable location, and check the element that we took its location as we do for the first one. The image below explains simply shifting the first element 'b'.



Another obvious example, let the list be {10, 5, 2, 3} , for the first element 10 there are 3 elements less than it, so we will move it to location 3 instead of element 3, {10, 5, 2, 10}. After that we will check element 3, it has 1 element less than it so it will moved to location 1, {10, 3, 2, 10} and element 5 will be checked. Element 5 has 2 elements less than it so its new location is 2, {10, 3, 5, 10} and element 2 will be checked. Element 2 will be in the first location because it is the least number. Finally, the list becomes sorted {2, 3, 5, 10}, but the code to be implemented will continue to check all numbers.

Time and Space Complexity

	Time	Auxiliary Space
Best Case	$O(n^2)$	O(1)
Average Case	$O(n^2)$	O(1)
Worst Case	$O(n^2)$	O(1)

When the list has been already sorted, the time will be taken is on checking every element if is it in the right location or not, otherwise time will consume is on checking and replacing if the element is not in the right location.

Cocktail Sort

Algorithm

The algorithm extends bubble sort operating in two directions, so each iteration of the algorithm is broken up into 2 stages:

First stage

Through the array, each two adjacent elements will be compared, if the left element is greater than the right one, they will be swapped. At the end, the greatest element will be in the las location in the array.

Second Stage

It will start in opposite direction through the array starting from the element recently stored, also it will compare each adjacent elements and swap if requires.

Let the array be {3, 4, 2, 1} as an example. First, we will begin with the first stage by comparing the first two elements {**3**, **4**, 2, 1}, element 3 is less than element 4 hence no swapping, {3, **4**, **2**, 1} element 4 is greater than element 2 hence swap {3, 2, 4, 1}, {3, 2, **4**, **1**} element 4 is greater than element 1 hence swap {3, 2, 1, 4}, we finished the first stage. In second stage, {3, **2**, **1**, 4} element 2 is greater than element 1 hence swap {3, 1, 2, 4}, {**3**, **1**, 2, 4} element 3 is greater than element 1 hence swap {1, 3, 2, 4}, after that we put the greatest element in last location and the smallest element in first location. Repeating first stage, {1, **3**, **2**, 4} element 3 is greater than element 2 hence swap {1, 2, 3, 4}. So the final array is ascending sort. In this case, this sort consumes three traversals, but in bubble sort it will consume four traversals.

Time and Space Complexity

	Time	Auxiliary Space
Best Case	O(n)	O(1)
Average Case	$O(n^2)$	O(1)
Worst Case	$O(n^2)$	O(1)

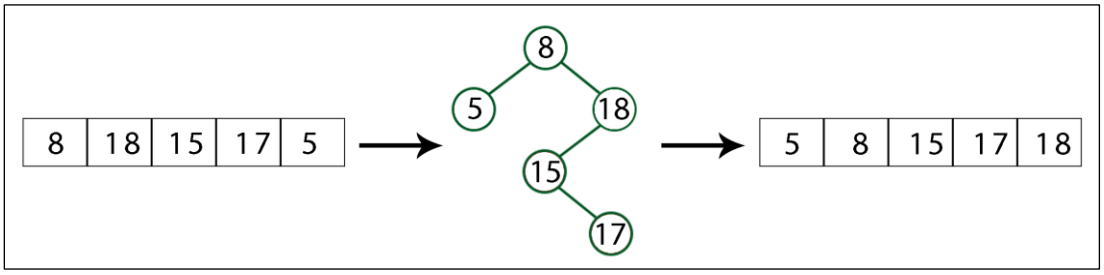
If the array is already sorted ascending, time complexity will be O(n). It will take more time if the array has more unsorting elements; so the descending sorted array is the worst case.

Tree Sort

Algorithm

This sort is based on Binary Search Tree (BST), we create a BST tree by inserting the elements of the array to be sorted into the tree. In any node in the tree, its left subtree's elements are less than the right ones, so when we do in-order traversal to insert back tree's elements into the array, the elements in the array must be sorted.

Example



Time and Space Complexity

	Time	Auxiliary Space
Best Case	O(n log n)	O(n)
Average Case	O(n log n)	O(n)
Worst Case	$O(n^2)$	O(n)

Time consumes O(log n) to add one element into the tree and O(n log n) to add n elements. If the tree is self-balancing binary search tree, then its worst case in time complexity will be O(n log n). If the array is already nearly or completely sorted (descending or ascending) it makes the worst case.

Summary

According to the time complexity, we can see that the Tree sort is the fastest when it is balanced, and the unbalanced tree has time complexity slower than the balanced one. Slower than that, we can say that Cocktail sort is faster than Cycle sort because it has time complexity in the best case O(n), but Cycle sort is $O(n^2)$, so we can consider Cycle sort as the slowest.

According to the space complexity, Tree sort has the greatest auxiliary space because it stores the elements into a temporary tree to sort them, so the extra space depends on the number of elements. On the other hand, Both Cycle sort and Cocktail sort has the least space because they need a certain temporary space to make the sorting without depending on number of elements.

Tree sort can be used when we need a fast sort without looking to memory storing or with little elements, but if we need to keep the memory safe with neglecting the speed, we can use either Cocktail sort or Cycle sort.

References

- COMP2321 Lecture Notes
- Dr. Radi Jarrar Slides
- https://www.geeksforgeeks.org/cycle-sort/
- https://en.wikipedia.org/wiki/Cycle_sort
- https://www.geeksforgeeks.org/cocktail-sort/
- https://en.wikipedia.org/wiki/Cocktail_shaker_sort
- https://www.geeksforgeeks.org/tree-sort/
- https://en.wikipedia.org/wiki/Tree_sort
- https://www.youtube.com/watch?v=n2MLjGeK7qA