Kotebe University of Education

Natural and Computational Science
Department Computer Science
Data Structure and Algorithem
Group Assigment

Group members:

Name	**ID**
1.Saleh Elias	CNCS/UR15419/12
2.Samuel Ayele	CNCS/UR15426/12
3 Nahom Girma	CNCS/UR15353/12
4.Atnabob Dessea	CNCS/UR14964/12
5.Siriyan Asfaw	CNCS/UR15454/12

Submission Date:-Dec,06,2021

Cocktail sort

- **Cocktail sort, also known as bidirectional bubble sort,[cocktail sort, shaker sort (which can also refer to a variant of selection sort), ripple sort, shuffle sort, or shuttle sort,**
- **It is an extension of the Bubble Sort. The algorithm extends bubble sort by operating in two directions. While it improves on bubble sort by more quickly moving items to the beginning of the list, it provides only marginal performance improvements.**

Algorithm-Cocktail sort

Each iteration of the algorithm is broken up into two stages:

1. The first stage loops through the array from left to right, just like the Bubble Sort. During the loop, adjacent items are compared and if value on the left is greater than the value on the right, then values are swapped. At the end of first iteration, largest number will reside at the end of the array.
2. The second stage loops through the array in opposite direction- starting from the item just before the most recently sorted item, and moving back to the start of the array. Here also, adjacent items are compared and are swapped if required.

Implementation-Cocktail Sort

```
#include <iostream>
using namespace std;>
using namespace std;

// Sorts arrar a[0..n-1] using Cocktail sort
void CocktailSort(int a[], int n)
{
    bool swapped = true;
    int start = 0;
    int end = n - 1;

    while (swapped) {
            // reset the swapped flag on entering
            // the loop, because it might be true from
            // a previous iteration.
            swapped = false;

            // loop from left to right same as
            // the bubble sort
            for (int i = start; i < end; ++i) {
                    if (a[i] > a[i + 1]) {
                            swap(a[i], a[i + 1]);
                            swapped = true;
                    }
            }

            // if nothing moved, then array is sorted.
            if (!swapped)
                    break;

            // otherwise, reset the swapped flag so that it
            // can be used in the next stage
            swapped = false;

            // move the end point back by one, because
            // item at the end is in its rightful spot
            --end;
```

```
                // from right to left, doing the
                // same comparison as in the previous stage
                for (int i = end - 1; i >= start; --i) {
                    if (a[i] > a[i + 1]) {
                        swap(a[i], a[i + 1]);
                        swapped = true;
                    }
                }
                // increase the starting point, because
                // the last stage would have moved the next
                // smallest number to its rightful spot.
                ++start;
        }
    }

    /* Prints the array */
    void printArray(int a[], int n)
    {
        for (int i = 0; i < n; i++)
                printf("%d ", a[i]);
        printf("\n");
    }


    int main()
    {
        int arr[] = { 5, 1, 4, 2, 8, 0, 2 };
        int n = sizeof(arr) / sizeof(arr[0]);
        CocktailSort(arr, n);
        printf("Sorted array :\n");
        printArray(arr, n);
        return 0;
    }
```

Output:

```
Sorted array :

0 1 2 2 4 5 8
```

Example-Cocktail sort:

Let us consider an example array (5 1 4 2 8 0 2)

First Forward Pass:
(**5 1** 4 2 8 0 2) ? (**1 5** 4 2 8 0 2), Swap since 5 > 1
(1 **5 4** 2 8 0 2) ? (1 **4 5** 2 8 0 2), Swap since 5 > 4
(1 4 **5 2** 8 0 2) ? (1 4 **2 5** 8 0 2), Swap since 5 > 2
(1 4 2 **5 8** 0 2) ? (1 4 2 **5 8** 0 2)
(1 4 2 5 **8 0** 2) ? (1 4 2 5 **0 8** 2), Swap since 8 > 0
(1 4 2 5 0 **8 2**) ? (1 4 2 5 0 **2 8**), Swap since 8 > 2
After first forward pass, greatest element of the array will be present at the last index of array.
First Backward Pass:
(1 4 2 5 **0 2** 8) ? (1 4 2 5 **0 2** 8)
(1 4 2 **5 0** 2 8) ? (1 4 2 **0 5** 2 8), Swap since 5 > 0
(1 4 **2 0** 5 2 8) ? (1 4 **0 2** 5 2 8), Swap since 2 > 0
(1 **4 0** 2 5 2 8) ? (1 **0 4** 2 5 2 8), Swap since 4 > 0
(**1 0** 4 2 5 2 8) ? (**0 1** 4 2 5 2 8), Swap since 1 > 0
After first backward pass, smallest element of the array will be present at the first index of the array.

Second Forward Pass:
(0 **1 4** 2 5 2 8) ? (0 **1 4** 2 5 2 8)
(0 1 **4 2** 5 2 8) ? (0 1 **2 4** 5 2 8), Swap since 4 > 2
(0 1 2 **4 5** 2 8) ? (0 1 2 **4 5** 2 8)
(0 1 2 4 **5 2** 8) ? (0 1 2 4 **2 5** 8), Swap since 5 > 2
Second Backward Pass:
(0 1 2 **4 2** 5 8) ? (0 1 2 **2 4** 5 8), Swap since 4 > 2

Now, the array is already sorted, but our algorithm doesn't know if it is completed. The algorithm needs to complete this whole pass without any **swap** to know it is sorted.

(0 1 **2 2** 4 5 8) ? (0 1 **2 2** 4 5 8)

(0 **1** **2** 2 4 5 8) ? (0 **1** **2** 2 4 5 8)

<u>Analysis-Cocktail sort</u>

Complexity

- Worst case time complexity: **Θ(n*n)**
- Average case time complexity: **Θ(n*n)**
- Best case time complexity: **Θ(n)**
- Space complexity: **Θ(1)**

Comparison with bubble sort

1. Time complexities are same, but Cocktail performs better than Bubble Sort. Typically cocktail sort is less than two times faster than bubble sort.
2. As the cocktail shaker sort goes bidirectional, the range of possible swaps, which is the range to be tested, will reduce per pass, thus reducing the overall running time slightly. For instance :- (2, 3, 4, 5, 1) Bubble sort requires four traversals of array for this example, while Cocktail sort requires only two traversals.