***Data Structures and AlgorithmsCOMP2421***



Computer Science Department.

Project 4

Sorting algorithms.

Student Name: Ibrahim Nobani

Student ID: 1190278

Instructor: DR. Radi Jarrar

Section: 4

Abstract:

In this report we are going to talk about 3 new and unique sorting algorithms, which are Tim sort, cocktail shaker sort and gnome sort.

1. Tim Sort

Tim sort is a pleasant sorting algorithm derived from two very well-known sorting algorithms, starting with Insertion sort and followed by merge sort.

It is also considered one of the fastest sorting algorithms along with quick sort, as well as being using as a default sorting algorithm in both Java and python.

Being a mix of two stable sorting algorithms, Tim sort can also be considered as a stable sorting algorithm.

-The array is divided into little blocks (sub array called Run) of size Run (Run will be defined alone as a multiple of 32 and so on.)

-Using insertion sort, each block will be sorted individually.

-Sorting is over when the array size is less than Run.

-If not, the process is repeated until we sort all the possible Runs.

-Finally, using Merge sort, the Runs are merged back into the array.

\*The algorithm [1]:

// iterative Timsort function

// let Run be defined as 32

void timSort(int arr[], int n)

{

    // start by Sorting the individual subarrays (blocks) of size RUN

    for (int i = 0; i < n; i+=RUN)

        insertionSort(arr, i, min((i+31), (n-1)));

//in case the array size is not a multiple of 32, we take the minimum value

    // now we start merging the blocks from size RUN (in this case 32).

    // merge process will raise exp from size 64, then 128, 256 and so on

    for (int size = RUN; size < n; size = 2\*size)

//in case array size < run we won do the merge process, just insertion above

    {

        // we now pick starting point of left sub array.so We can

        // merge arr[left..left+size-1] and arr[left+size, left+2\*size-1]

        // After every merge, we increase left by 2\*size

        for (int left = 0; left < n; left += 2\*size)

        {

            // we need the end point of left sub array(mid)

            // mid+1 is starting point of right sub array

// right is end point of the second sub array

            int mid = left + size - 1;

            int right = min((left + 2\*size - 1), (n-1));

            // now we merge arr[left.....mid] & arr[mid+1....right]

            merge(arr, left, mid, right);

        }

    }

}

Time and space complexity [2]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Case  (not Sorted) | Worst Case (Sorted descending) | Best Case  (Sorted Ascending) | Space complexity |
| TimSort | **O (n log n)** | **O (n log n)** | **O (n)** | **O (n)** |

For partially stored data, Tim Sort is very convenient, with the best case being totally sorted data.

Is Tim Sort in place or not? Well, it mainly depends on the implementations of the two sorting algorithms it relies on (Insertion and merge), but the most common implementation uses a not in place merge sort, thus we consider Tim Sort to be NOT TO BE IN PLACE.

2) Cocktail Shaker Sort

Cocktail sort is a sorting algorithm that is somewhat similar to bubble sort, it is also a stable sorting algorithm, the cocktail sort traverses the array in two ways (forward and backward), it has the same complexity as bubble sort, but can be twice as fast.

-At the start the element in the array is traversed in a forward direction, from left to right, the adjacent elements are compared and interchanged if the left element is bigger that the right one just like bubble, the largest number will reside at the end of the array.

-Followed by the first step the traverse now occurs backwards, and if the right element is less than the left one, same procedure occurs, they are interchanged and the smallest number will reside at the start of the array.

-The process is repeated till the array is sorted.

\*Algorithm [3]:

void CocktailSort(int a[], int n)

{

    bool swapped =true; //will be used to check if the array is sorted or not

    int start = 0, end = n - 1;

      while (swapped) {

        swapped = false; //reset the flag always for extra caution

        // forward traverse loop following bubble sort main concept

        for (int i = start; i < end; ++i) {

            if (a[i] > a[i + 1]) {

                swap(a[i], a[i + 1]);

                swapped = true;

            }

        }

        if (!swapped) //if nothing moved, then array is sorted, we return

            break;

        // else, reset swapped so that it can be used in the next stage

        swapped = false;

        --end;

//decrease it by one (largest is the last element already) and so on.

        // backward traverse following bubble sort concept.

        for (int i = end - 1; i >= start; --i) {

            if (a[i] > a[i + 1]) {

                swap(a[i], a[i + 1]);

                swapped = true;

            }

        }

        ++start;

//increase start because the smallest is the first element already and so on.

    }

}

Time and space complexity [4]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Case  (not Sorted) | Worst Case (Sorted descending) | Best Case  (Sorted Ascending) | Space complexity |
| cocktail sort | O (n^2) | O (n^2) | O (n) | O (1) |

Bubble sort time complexity is same as cocktail sort, with better performance for the latter, it is also less than two times faster than the bubble sort.

The cocktail sort is bidirectional, which means the possible swaps range, will reduce per pass, which leads to a slight reduce in the running time.

Cocktail sort handles and modifies elements directly in the array, so it’s conceded to be sort In Place algorithm.

1. Gnome Sort

Gnome sort is a stable sorting algorithm that works with one item at a time (like insertion sort), and sorts them in the right way with a series of swaps (similar to bubble sort).

Inspired by the garden gnome technique, imagine flower pots, basically you check a flower pot and the one before it, if their order is right, we move forward, else they are swapped and one we move backward. It is used everywhere a stable sort algorithm is not needed.

- If you are at the beginning of an array, head to the right element (from array[0] to array[1]).

-Take one element at a time, If the array element is equal or larger than the previous, move one step forwards (to the right).

- If the array element is less than the one before it, swap them and move one step backwards.

- Repeat the last 2 steps until it reaches the end of the array.

-If the end of the array is reached then stop and the array is sorted.

\*Algorithm [5]:

// gnome sort code

Void gnomeSort (int arr[], int n){

// index=0, which is the array element count.

int index=0;

While (index < n){ // index does not reach end of the array

if (index == 0)

// if the index is zero, which means we are still at the //start of the array (arr[0]), move one forward to arr[1]

index++;

if (arr[index] >= arr[index-1])

// if the element is equal or bigger than the previous, //move forward

index++;

else {// if not, swap both elements and move one step //backwards

swap(arr[index], arr[index-1]);

index--;

}

}

Return;

}

Time and space complexity [6]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Case  (not Sorted) | Worst Case (Sorted descending) | Best Case  (Sorted Ascending) | Space complexity |
| Gnome sort | O (n^2) | O (n^2) | O (n) | O (1) |

Gnome sort is not that efficient, the time and space complexity is similar to the bubble sort, Gnome Sort is an in-place sort that is does not require any extra storage.

Summary:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Case  (not Sorted) | Worst Case (Sorted descending) | Best Case  (Sorted Ascending) | Space complexity |
| TimSort | O (n log n) | O (n log n) | O (n) | O (n) |
| cocktail sort | O (n^2) | O (n^2) | O (n) | O (1) |
| Gnome sort | O (n^2) | O (n^2) | O (n) | O (1) |

As seen, Tim sort is far more superior than the other two, also being much faster than them. While both cocktail and gnome sort are similar to the bubble sort complexity.

.

References:

[1]<https://www.geeksforgeeks.org/timsort/>

[2] <https://iq.opengenus.org/tim-sort/>

[3]<https://www.geeksforgeeks.org/cocktail-sort/>

[4] <https://iq.opengenus.org/cocktail-shaker-sort/>

[5] <https://tutorialspoint.dev/algorithm/sorting-algorithms/gnome-sort-a-stupid-one>

[6] <https://iq.opengenus.org/gnome-sort/>

[7] <https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/>

-Ideas were inspired from these two videos:

<https://www.youtube.com/watch?v=OOBBI-kSChM>

<https://www.youtube.com/watch?v=kPRA0W1kECg>