Algorithms: Introduction to Sorting Algorithm Part-1

Topics To be Covered In Class

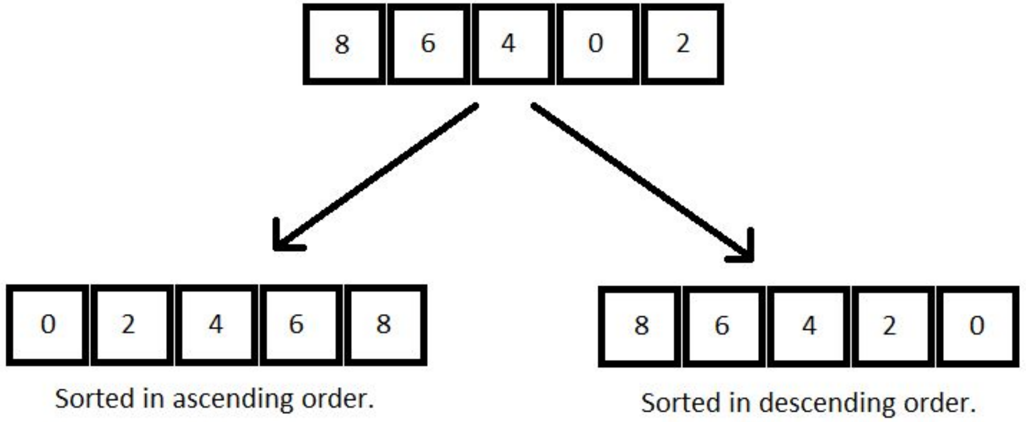
1. Sorting
2. Stability in Sorting
3. Bubble Sort
4. Problem on Bubble Sort
5. Selection Sort

What is Sorting?

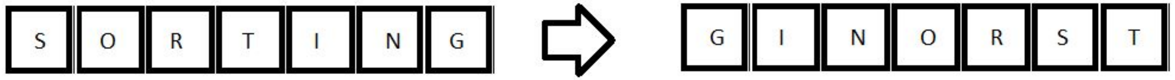
Quite often, we like to arrange things or data in a particular order.
Sometimes to improve the readability of that data, at others to search or extract some information quickly out of that data. For example, When we go to a travel site to book a hotel, the site gives us all these options of sorting the hotels by price from low to high or by star rating or by guest rating. Here sorting can help a lot.

Sorting is arranging the elements in a list or array in any desired permutation. The list should be homogeneous; that is, all the elements in the list should be of the same type.

For example:
In the below example, the elements of the given list hold numerical values and are rearranged based on the numerical order.



In the below example, the elements of the given list are holding alphabetical values so are rearranged based on the alphabetical order.



Types of Sorting Algorithms

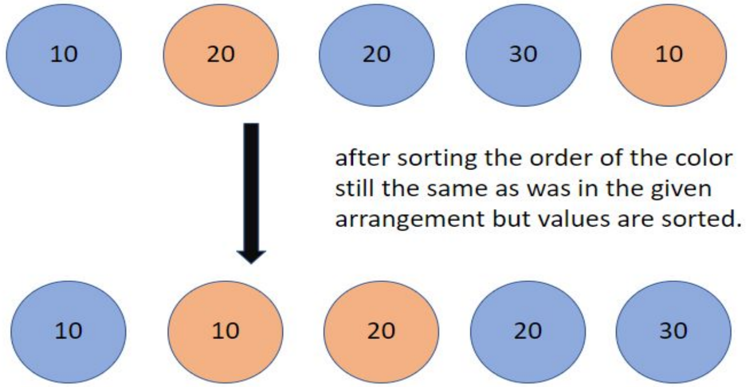
Sorting is basically classified into two types i.e.:

1. Comparison based sort
2. Counting based sort

And the selection of the sorting algorithm is based upon their stability, in-place property, space complexity, and comparison and swap times.

Stability in sorting algorithms

A sorting algorithm is stable when two objects with equal keys appear in the same order in output, as they appear in the input array. Stability is mainly important when we have key-value pairs with duplicate keys (like people's names as keys and their details as values). And we have to sort these objects by values.


10
20
20
30
10
after sorting the order of the color still the same as was in the given arrangement but values are sorted.
10
10
20
20
30

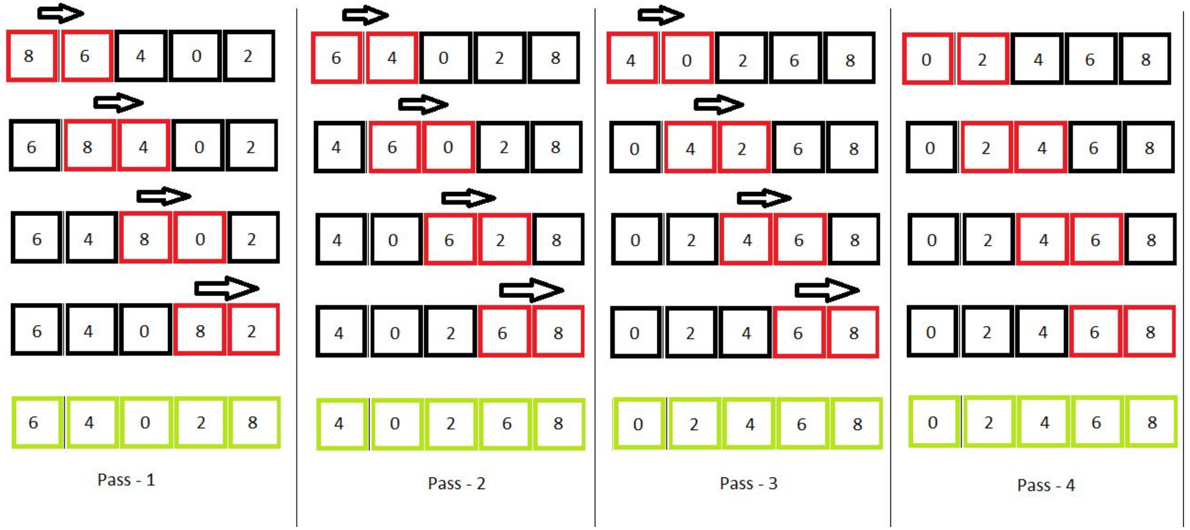
Any sorting algorithm which is not stable can be made stable by modifying the comparison operation to move the least element to the desired position in the array in each iteration.

Inplace property

While sorting, we have two options: to create a new array or list and place the elements from the given list to the new list in sorted order or change the positions of the elements within the list in ascending or descending order to get a sorted list.
In place, property helps avoid extra space and use the same memory space containing the input and producing the required output without using extra memory space.

Bubble Sort

Bubble sort got its name from the idea on how the larger bubble in water rises up to the surface, a similar approach is followed in this algorithm where the bigger elements are moved at the end of the list (while sorting the element in decreasing order)



Algorithm Working

The idea behind this algorithm is to check the order of the elements in the input array. It compares two adjacent elements and checks if they are in the correct order (ascending or descending). If they are not in order, the elements are swapped.

Let's consider an array of size 4. Array = **[5, 3, 1, 2]**

The algorithm checks for the 0th and 1st elements. If the 0th element is greater, then it swaps the two elements. If the 0th element is smaller, it doesn't take any action.
So, the above array will become **[3, 5, 1, 2]**

Then, the 1st element is compared to the second element. Since five is greater than 1, the two elements are again swapped.
The new array will be **[3, 1, 5, 2]**

Further, the 2nd element (5) is greater than the 3rd. Thus, it will be swapped again, and the array will become **[3, 1, 2, 5]**

This ends the first pass, and the array becomes **[3, 1, 2, 5]**. The largest element bubbles to the correct position in each iteration. Hence the algorithm is known as Bubble sort.

After the second pass, the array will be **[1, 2, 3, 5].** The second largest element is put in the correct position in this iteration.

No element will be swapped during the third pass as the array is already sorted. So, we will break from the loop.

Pseudo code
Code for reference - https://p.ip.fi/m-gT

Implementation
Code for reference - https://p.ip.fi/l3lJ

Complexity Analysis

Time Complexity
In the worst case, the input array will be sorted in a descending order. In this case, the first iteration will involve N-1 comparisons. In the second iteration, there will be N-2 comparisons. The total number of comparisons will be

Total = (N-1) + (N-2) + (N-3) + ... + 1

The above is an arithmetic series and can be expressed as Total = N*(N-1) /2, so Total = N^2/2 - N/2

In terms of Big-O notation, the total time complexity will be **O(N^2)** ignoring the constant factor of ½.

Space Complexity
We only use one temporary variable for swapping & a boolean to check if any swapping was done. Thus, the Space complexity is **O(1)**.

In-Place algorithm
In the case of Bubble sort, we don't use any extra space. We performed all the manipulations on the input array and sorted the final output. Hence, Bubble sort is an in-place algorithm.

Stability
Explain the concept of stability if the learners don't know about it. Following is a brief description of stability:-

A sorting algorithm is stable if it preserves the relative ordering of the elements after sorting. If there are two elements with the same value, they should appear in the same order as they appear in the input after sorting. Here is an example:-

Input Array = [3, 5, 7, 2, 7, 9, 10]

The above array has two elements with the value 7. For understanding, one has been highlighted in red & the other in blue. A stable sorting algorithm will produce the below output:-

Output Array = [2, 3, 5, 7, 7, 9, 10]

However, an unstable sorting algorithm wouldn’t preserve the order and will generate the below output:-

Output Array = [2, 3, 5, 7, 7, 9, 10]

Bubble sort is a comparison-based sorting algorithm & thus is a stable sorting algorithm.

You can explain why Bubble sort is a stable sorting algorithm. It doesn’t change the order of elements if the elements are equal. The element is swapped only if the adjacent element is greater or lesser (sorting the array in descending order) than the current element.

Application of Bubble Sort

1. In a TV application to sort the program based upon the audience viewing time.
2. To understand the foundation of sorting.

To find kth largest/smallest element

We can modify our bubble sort algorithm to detect the kth largest or smallest element in a sorted array.
To do so we just need to update the outer loop by making it run only k number of times and then take the kth last element of the array that we got after kth iteration.

So if in array [5,4,3,2,1,6,2] i have to find the 3rd largest element then

```
arr=[5,4,3,2,1,6,2];
function bubbleSort(inputArray,k) {
  for (var i = 0; i < k; i++) {
    var isSwapped = false
    // Last i elements are already sorted
    for (var j = 0; j < (inputArray.length - i - 1); j++) {
      // Check if the current element is greater than the next element
      if (inputArray[j] > inputArray[j + 1]) {
        // If the condition is true then swap them
        var temp = inputArray[j]
        inputArray[j] = inputArray[j + 1]
        inputArray[j + 1] = temp
        isSwapped = true
      }
    }
    if (!isSwapped)
      break
  }
  return inputArray
}
k=3
arr=  bubbleSort(arr,k);
console.log(arr[arr.length-k]);
```

```
Output >> 4
```

Time complexity for the same will be O(n*k)

Problem (20min)

Give an array of name of country, you are supposed to sort it in lexicographical order using the bubble sort

Input : [“India”,”Australia”,”China”,”Russia”,”Brazil”,”Japan”]
Output: [”Australia”,”Brazil”,”China”,“India”,”Japan”,”Russia”]

Hint: String comparison can be done using localeCompare()

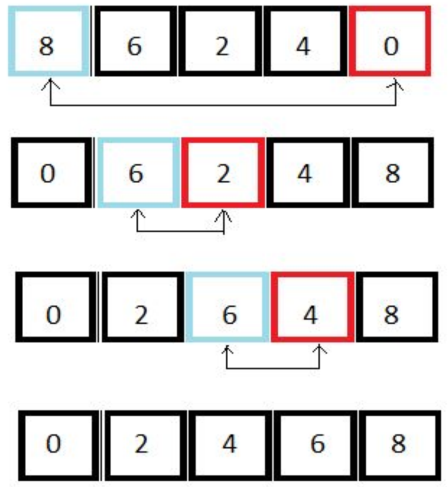
Solution

```
function bubbleSort(inputArray) {
   for (var i = 0; i < inputArray.length; i++) {
      var isSwapped = false
      // Last i elements are already sorted
      for (var j = 0; j < (inputArray.length - i - 1); j++) {
         // Check if the current element is greater than the next element
         if (inputArray[j].localeCompare(inputArray[j + 1])!=-1) {
            // If the condition is true then swap them
            var temp = inputArray[j]
            inputArray[j] = inputArray[j + 1]
            inputArray[j + 1] = temp
            isSwapped = true
         }
      }
      if (!isSwapped)
         break
   }
   return inputArray
}

// This is our unsorted array
var arr = ["India","Australia","China","Russia","Brazil","Japan"];
console.log(bubbleSort(arr));
```

Selection Sort

Visualization
Run through the visualization after explaining the algorithm.



Real-world Analogy

Let's say you are organizing a race and distributing chocolates at the end of the race. The chocolates are of different sizes, and right now, there is no order between them. During the prize distribution, the last kid will get the smallest chocolate, the second last will get the second-smallest chocolate, and the winner will get the largest chocolate.

Assume that prize distribution begins by calling the last kid first. Since the chocolates are not in any order, you will have to first find the chocolate with the minimum size and give it to the last kid. Then find the second smallest one and give it to the kid who came second last. You would continue this process until you assign the largest chocolate to the winner.

Algorithm
The algorithm consists of multiple steps. In the first step, you find the minimum element in the array. Then you swap the minimum element with the first element. The second minimum element is found and exchanged with the second element in the next step. This process is repeated, and the next smaller element is swapped until the whole array is sorted.

Since the algorithm repeatedly selects the next-smallest elements and puts them in their place, the algorithm is known as Selection sort.

This algorithm divides the array into two parts - a) Sorted left subarray b) Unsorted right subarray. The sorted left subarray is initially empty. The idea is to fill up the left subarray and remove all the elements from the unsorted right subarray. We perform the following steps to achieve this:-

1. Find the minimum element in the unsorted subarray.
2. Swap the minimum element with the leftmost element of the unsorted subarray.
3. The leftmost element becomes a part of the sorted left subarray. And the size of the unsorted subarray is reduced by 1.
4. This process continues until the size of the sorted left subarray becomes N.

Let's consider an array of size 4. Array = [5, 3, 1, 2]

Initially, the left sorted subarray will be empty, and the right unsorted subarray will be the original array.

Left sorted subarray = [], Right unsorted subarray = [5, 3, 1, 2]

In the first step, the minimum element found is 1. This element is exchanged with the leftmost element of the right subarray. The right subarray becomes [1, 3, 5, 2]. Further, the leftmost element is removed from the right subarray and added to the end of the left sorted subarray.

After step 1, the following is the state of the two subarrays:-

Left sorted subarray = [1], Right unsorted subarray = [3, 5, 2]

In the next step, the minimum element in the right subarray is 2. This is swapped with the leftmost element i.e 3. Thus, the right subarray becomes [2, 5, 3]. The leftmost element two is removed from the right subarray and added to the end of the left subarray.

After step 2, the following is the state of the two subarrays:-

Left sorted subarray = [1, 2], Right unsorted subarray = [5, 3]

3 is the minimum element in the right subarray in the third step and gets swapped with 5. The right subarray will become [3, 5]. Three then gets removed from the right subarray and is added to the left sorted subarray.

After step 3, the following is the state of the two subarrays:-

Left sorted subarray = [1, 2, 3], Right unsorted subarray = [5]

In the last step, 5 is the only element left in the right subarray and is removed and added to the left subarray.

Left sorted subarray = [1, 2, 3, 5], Right unsorted subarray = []

The left subarray now contains all the elements in sorted order.

Pseudo code

1. Code for reference - https://p.ip.fi/i42i

Implementation

1. Code for reference - https://p.ip.fi/t1r6

Complexity Analysis

Time Complexity

The algorithm consists of two loops. The second loop iterates on all the elements and finds the first minimum element. It takes N iterations in the first pass. The next pass scans N-1 elements to find the second minimum element. This process continues, and we can come up with the following expression to count the total number of iterations:-

Number of iterations = N + (N-1) + (N-2) + ……. + 1

This an arithmetic series and can be simplified and can be written as **Number of iterations = N * (N+1) / 2**

= N^2/2 + N/2

In terms of Big-O notation, the total time complexity will be **O(N^2)**, ignoring the constant factor of ½.

Time Complexity of the algorithm = **O(N^2)**

Space Complexity

In the implementation, we have only used two extra variables—one for swapping the next minimum element and the other to find the index of the minimum element. There is no extra space allocation. So, the space complexity of the algorithm is **O(1)**

The space complexity of the algorithm = **O(1)**

Stability

Selection sort is not a stable sorting algorithm. The algorithm finds the minimum element, and then swaps it with the element which is in its position. Thus, the relative ordering of the elements can change.

Input : 14 15 6 5 14 1

On running Selection sort on the above array, we will get the following output :-
Output : 1 5 6 14 14 15

A stable sorting algorithm would have produced 1 5 6 14 14 15

How to make selection sort stable ?
Swapping of the elements makes selection sort unstable as the order of the element changes. It can be made stable by placing the element in its correct position and then sliding the remaining elements to the right. Let's look at the above example :-

Input : 14 15 6 5 14 1
After the first iteration, the minimum element is 1. We need to place 1 at the first position, so we slide all the elements to the right. We get

Input : 1 14 15 6 5 14

The next minimum is 5, and its correct place is 2nd position or index=1. We follow the same procedure and slide the elements starting from index=1 till index=4

Input : 1 5 14 15 6 14

Continuing with the same process, we get the following in subsequent iterations

Input : 1 5 6 14 15 14

Input : 1 5 6 14 14 15

Application of Selection Sort

1. Sorting a sorted list is required.
2. The cost of exchanging is irrelevant.
3. It is necessary to check all of the elements.

Problem Statement

1. Implement the bubble sort algorithm to arrange a sequence of numbers from 1..100 in decreasing order. Analyze the time taken by the algorithm.
2. Perform the Time complexity analysis of bubble sort algorithm.

Upcoming Class Teaser

Next class is the continuation of this class, we will do some problem solving on selection sort and then cover another sorting algorithm i.e. insertion sort.

THANK YOU