Algorithms: Intro to Sorting Part-2

Relevel
by Unacademy

Class Topics:

- Comparison between bubble sort and selection sort
- Problem Solving using Selection Sort
- Insertion Sort
- Sorting in Javascript
- Problem Solving using Insertion Sort

Bubble Sort vs Selection Sort

Bubble Sorting	**Selection Sorting**
Performs the comparison of the adjacent elements and then do the necessary swapping	Performs the sorting by taking the minimum element from the unsorted array and then places it in the sorted array
Less Efficient compared to selection sort	More efficient compared to bubble sort
Slower	Faster
Performs exchanging of items	Perform selection of items

Problem-1

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

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

Hint: String comparison can be done using localeCompare()

Problem-1

Solution: https://jsfiddle.net/zt95cn3a/

```
function selectionSort(inputArray){
    var i, j, minimumIndex;
    for (i = 0; i < inputArray.length-1; i++)      {
         // Initialize the current element as minimumIndex
         minimumIndex = i;
         for (j = i + 1; j < inputArray.length; j++)         {
              if
(inputArray[j].localeCompare(inputArray[minimumIndex])==-1)
              {
                   minimumIndex = j;
              }           }
```

Problem-1

```
    // Swap the found minimum element with the current element
        if (minimumIndex != i)
        {
            var temp = inputArray[minimumIndex]
            inputArray[minimumIndex] = inputArray[i]
            inputArray[i] = temp
        }
    }
    return inputArray
}
// This is our unsorted array
var arr =
["India","Australia","China","Russia","Brazil","Japan"];
console.log(selectionSort(arr));
```

Problem-2

Given an object of employee name and there salary, find the second most paid employee of the company

Input :

arr=[{'name':'Ram','salary':100000},{'name':'Ramesh','salary':10000},{'name':'Rakesh','salary':500000},{'name':'Riya','salary':650000},{'name':'Rithik','salary':45000},{'name':'Ritesh','salary':230000}]

Output:

Rakesh

Problem-2

Solution: https://jsfiddle.net/9cuy4eng/1/

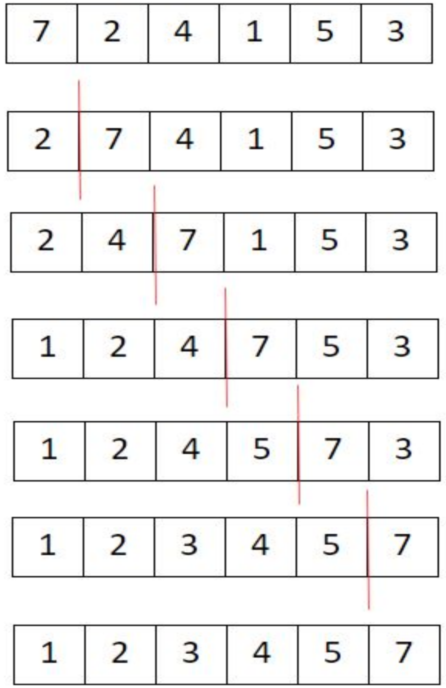
```
arr=[{'name':'Ram','salary':100000},{'name':'Ramesh','salary':1
0000},{'name':'Rakesh','salary':500000},{'name':'Riya','salary'
:650000},{'name':'Rithik','salary':45000},{'name':'Ritesh','sal
ary':230000}]

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++) {
```

Problem-2

```
 // Check if the current element is greater than the next
element
  if (inputArray[j]['salary'] > inputArray[j + 1]['salary']) {
                // 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=2
 arr=  bubbleSort(arr,k);
 console.log(arr)
 console.log(arr[arr.length-k]['name']);
```

Insertion Sort:



Insertion Sort:

Algorithm

Imagine that you are playing a game of cards. You are holding the cards in sorted order. Let's assume that the dealer hands you a new card. You will find the position of the new card and insert it in the correct position. The existing cards greater than the new card would be shifted right by one place.

The same logic is used in insertion sort. You loop over the Array starting from the first index. Each new position is analogous to a new card, & you need to insert it in the correct place in the sorted subarray to your left.

Insertion Sort:

Let’s take an example where the subarray from index 0 to index three is already sorted. We want to insert the element at index four into this sorted subarray. To insert an element at position four correctly, we repeatedly compare it to its left element. Let’s call the new element at position four as “**key**”. The “**key**” is compared with the element to its left. If the element is greater than the key, we shift it from one position to the right. If the element is lesser than the key, we stop as we have found the correct position. We then assign the key to the right of the current element.

Array = [1, 4, 5, 9, **3**]

We want to insert an element at index 4 in its correct position in the above example. The left subarray from index 0-3 is sorted in ascending order. Let’s create a temporary variable called key and assign Array [4] to it. The key denotes the element we want to insert in its correct place.

Insertion Sort:

Key = 3

We first compare the key with its immediate left element.

left = 3

A[left] = 9.

If A[left] > Key, then we shift this element to right by one place. This is done by copying the element to the next one. A[left+1] = A[left]

Insertion Sort:

Following will be the array after executing this logic :-

Array = [1, 4, 5, 9, **9**]

In the next step, we reduce the index left by 1.

left = 2
A[left] = 5

A[left] > key, so we shift this element one place to the right using A[left+1] = A[left].

Insertion Sort:

Following will be the array after executing this logic :-

Array = [1, 4, 5, **5**, **9**]

We now decrement Left by 1, and Left now becomes 2.

left = 1

A[left] = 4

Since A[left] > key, A[left] is copied to A[left+1]. The new array will be :-

Array = [1, 4, **4**, **5**, **9**]

Insertion Sort:

The variable left will reduce by 1 and will be equal to 1.

left = 0
A[left] = 1

Here A[left] < key, so we have found the correct position of the key. We need to stop here, since all the elements to the left will be less than the key.

Finally, we set the value of A[left+1] to key. The final sorted array will be :-

Array = [1, **3**, **4**, **5**, **9**]

Insertion Sort:

Pseudo code

Code reference - https://p.ip.fi/6Ixy

Implementation

Code for reference - https://p.ip.fi/7IgJ

Insertion Sort:

Complexity Analysis

Time Complexity

Let’s consider when the array is sorted in descending order. In this case, every new key will be less than all the elements in the sorted left subarray. This will require us to shift all the elements (in the left subarray) to the right by one position and then insert the key in the first place. This will have to be done for all the elements, from index one until n-1. (n is the size of the array).

Let’s find the number of operations done in each step. We start from the 1st index and compare the element at the 1st index with the element at the 0th index. The element at the 0th index will slide one position right, and the key will be placed at the 0th index. This will cost us one operation.

Insertion Sort:

In the next step, we will have to move the element at the 2nd index. Since this key is less than all the elements in the left subarray, we will have to move each element to its right and place the key at the 0th index. This will cost two operations.

The above process will repeat until we reach the last element. The last element is the smallest, and we will have to move all the n-1 elements to the right. This will take n-1 operations.

Total operations = 1 + 2 + 3 + ... + n-1

Total operations = (N-1)*(N)/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.

Insertion Sort:

Worst Case/ Average case Time Complexity of the algorithm = O(N^2)

The input array will be sorted in ascending order in the best case. As a result, every other element is in its correct position. When the code executes, it will not enter the second while loop since the key is greater than the element to its left. The outer loop will run (N-1) times, and hence the time complexity will be O(N-1).

Insertion Sort:

Best case time complexity = $\Omega(N)$

Space Complexity

In the above algorithm, we are only using extra space to store the key. Hence, the space complexity will be **O(1)**.

Stability

Insertion sort is a stable sorting algorithm. It will ensure that relative ordering of two elements with the same value is preserved. Following example can be given :-

Input = [15 12 31 **14 14** 17]

Output = [12 **14 14** 15 17 31]

Insertion Sort:

The **14** will be processed first and then **14** will be inserted to its right, as the insertion sort loop breaks when A[left] <= Key.

Extend this problem to find the k-th largest element in the array, where k >=1 and k <= size of the array.

Use cases

- Insertion sort becomes useful when the array elements are almost sorted. We need very few insertions in case the array is almost sorted

Eg: 14 2 5 6, Here except for 2, every element is in its correct position.

- Useful in online query as well.

Application of Insertion Sort

- Insertion sort is only efficient when there are smaller elements to sort in an array because its time complexity can reach O(N2).
- Insertion sort is an in-place approach, which means it doesn't take up any additional space.

Custom Comparator

This is a function that is used to compare the objects of a user defined class. For example:

If there exits two objects as :

obj1={ name='Ikaris' , age = 23}

obj2={ name='Jack' , age = 32}

And we need to compare is both are the same or not then we can create a custom comparison function which will return a boolean value

```
var comp = function(obj1,obj2){
//Any logic based upon need of problem statement
return  obj1===obj2;
}
```

Sorting in Javascript

In javascript sorting can be performed using the sort function. This methods takes the array elements and performs an inplace sorting.

Syntax: `<arr_name>.sort(compareFn)`

Where, arr_name is the name of array

- Sort: method name
- CompareFn: An optional argument that is an custom operator containing the definition on how to sort the elements of the array.

Since compared is optional it should be noted that when it is not specified then the array elements are changed into string and the comparison is based upon the characters Unicode point value.
Suppose we have an array arr=[10,20,30,4,2,1] and we want to sort it in ascending order by using a functionless sort method then

Sorting in Javascript

```
arr=[10,20,30,4,2,1];
function compare(a,b){
    if (b<a){
     return 1
     }     else{
        return 1;
     }
};
arr.sort();
console.log(arr);
```

Output >> [1, 10, 2, 20, 30, 4]

Sorting in Javascript

Since comparison was made at chaacter level 10 is placed after 1. Now to compare the array based upon the number we can create a custom comparator function as

```
arr=[10,20,30,4,2,1];
function compare(a,b){
    if (b<a){
     return 1
     }     else{
        return -1;
     }   };
arr.sort(compare);
console.log(arr);
```

Output >> [1, 2, 4, 10, 20, 30]

Thank You!