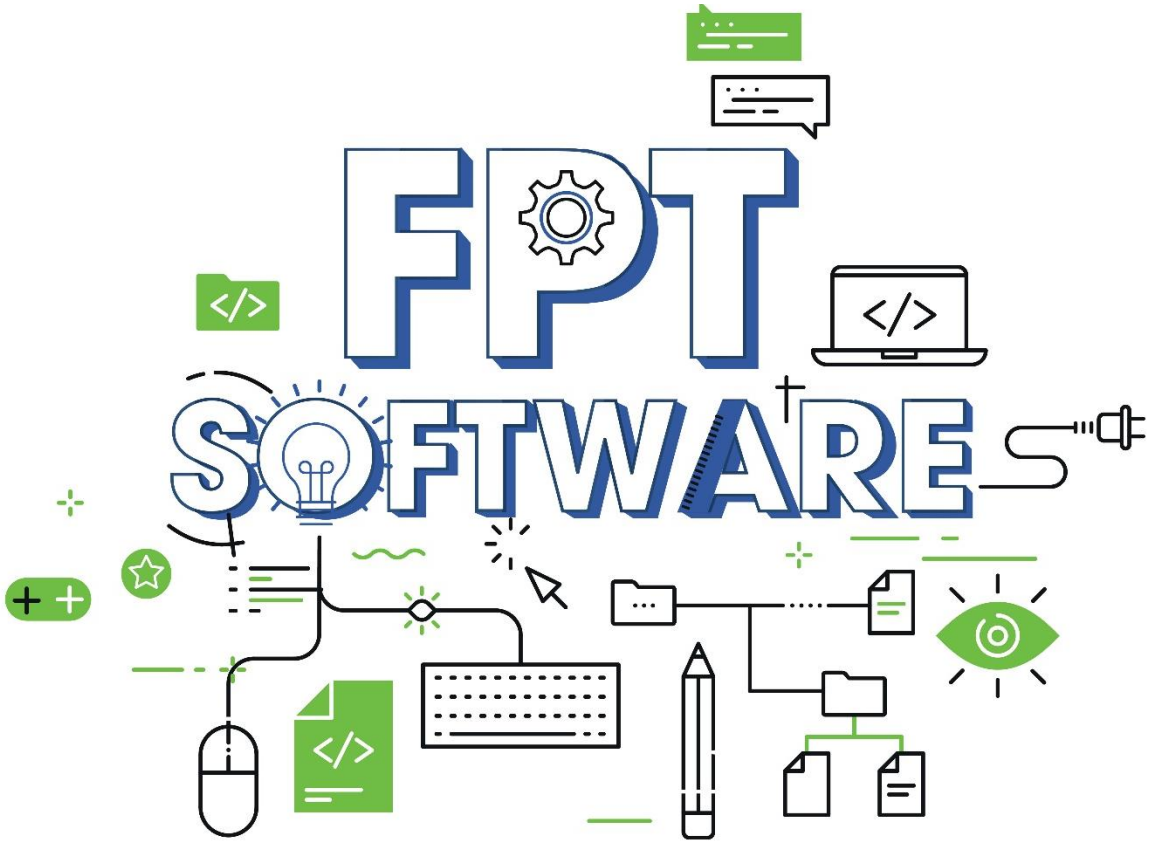
Basic Sorting Algorithms

Fsoft Academy



Lesson Objectives

- **Understand** the concept of sorting and its importance in computer science
- **Be familiar with different** sorting algorithms and their characteristics
- **Be able to analyze** the time and space complexity of different sorting algorithms
- **Be able to implement** different sorting algorithms in a programming language.

Agenda

1. Sorting Algorithms
2. Bubble Sort
3. Selection Sort
4. Q&A

Section 1

Sorting Algorithms

Introduction

A **Sorting Algorithm** is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

- In other words, a sorted array is an array that is in a particular order.
  - ✓ For example, [a,b,c,d][a,b,c,d] is sorted alphabetically, [1,2,3,4,5][1,2,3,4,5] is a list of integers sorted in increasing order, and [5,4,3,2,1][5,4,3,2,1] is a list of integers sorted in decreasing order.
- A sorting algorithm takes an array as input and outputs a permutation of that array that is sorted.

Introduction

- There are **two broad types of sorting algorithms**: integer sorts and comparison sorts.
- **Comparison Sorts**
  - ✓ Comparison sorts compare elements at each step of the algorithm to determine if one element should be to the left or right of another element.
  - ✓ Comparison sorts are usually more straightforward to implement than integer sorts, but comparison sorts are limited by a lower bound of O(nlogn), meaning that, on average, comparison sorts cannot be faster than O(nlogn).
  - ✓ A lower bound for an algorithm is the worst-case running time of the best possible algorithm for a given problem.

VD: insertion, selection, bubble, quick , heap, merge sort are comparison-based

Introduction

- **Integer Sorts**
  - ✓ **Integer sorts** are sometimes called **counting sorts** (though there is a specific integer sort algorithm called counting sort).
  - ✓ Integer sorts do not make comparisons, so they are not bounded by Ω(nlogn).
  - ✓ Integer sorts determine for each element x how many elements are less than x. If there are 14 elements that are less than x, then x will be placed in the 15th slot.
  - ✓ This information is used to place each element into the correct slot immediately—no need to rearrange lists..

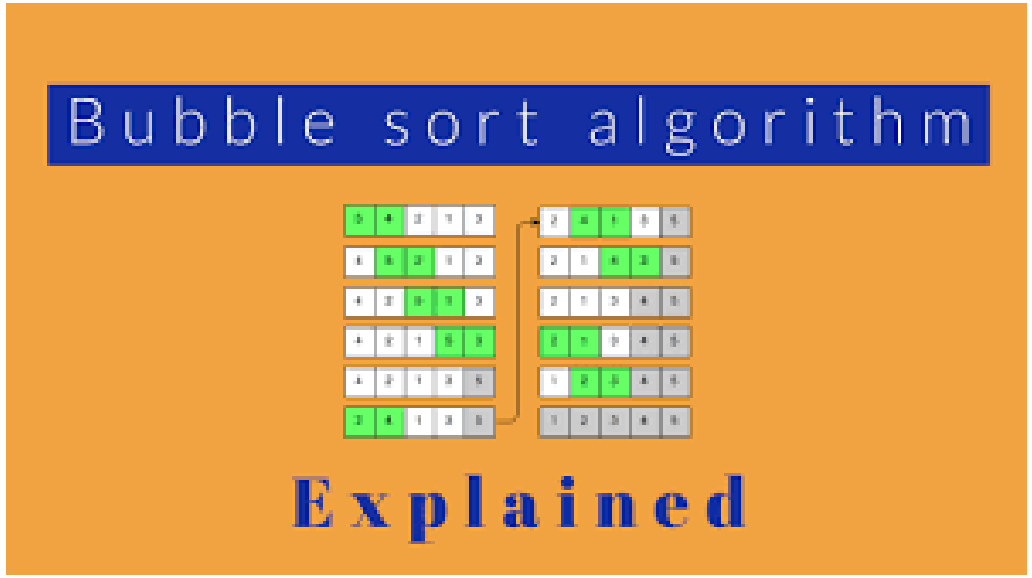
Counting Sort (O(n + k)) – Uses an auxiliary array to count occurrences.
Radix Sort (O(nk)) – Sorts numbers digit by digit.
Bucket Sort (O(n^2)) – Distributes elements into buckets and sorts them

Section 2

Bubble Sort Algorithm

Introduction

- **Bubble sort is a simple sorting algorithm.**
  - ✓This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.
  - ✓This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n is the number of items.

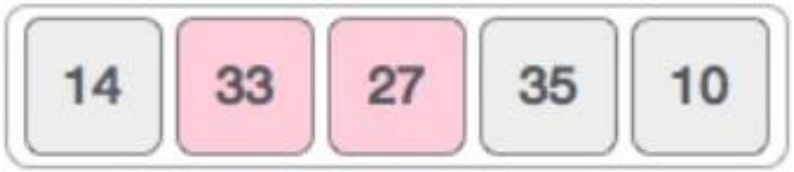


How Bubble Sort Work?

- Bubble sort starts with very first two elements, comparing them to check which one is greater.

- In this case, *value 33 is greater than 14*, so it is already in sorted locations. Next, *we compare 33 with 27*.

- We find that *27 is smaller than 33* and these two values must be swapped.



How Bubble Sort Work?

- The new array should look like this

- Next we *compare 33 and 35*. We find that both are in already sorted positions

- Then we move to the next two values, *35 and 10*.

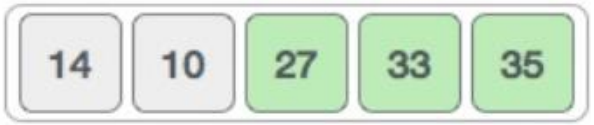
- We know then that *10 is smaller 35*. Hence they are not sorted.

How Bubble Sort Work?

- We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this

- To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this

- Notice that after each iteration, at least one value moves at the end.



- And when there's no swap required, bubble sorts learns that an array is completely sorted.

Algorithm

- We assume **list** is an array of **n** elements.
- We further assume that **swap** function swaps the values of the given array elements.

```
begin BubbleSort(list)

   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for

   return list

end BubbleSort
```

Pseudocode

- We observe in algorithm that **Bubble Sort compares each pair of array** element unless the whole array is completely sorted in an ascending order.
  - ✓This may cause a few complexity issues like what if the array needs no more swapping as all the elements are already ascending.
  - ✓To ease-out the issue, we use one flag variable swapped which will help us see if any swap has happened or not.
  - ✓If no swap has occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.

Pseudocode

```
procedure bubbleSort( list : array of items )

   loop = list.count;

   for i = 0 to loop-1 do:
      swapped = false

      for j = 0 to loop-1 do:


     /* compare the adjacent elements */
         if list[j] > list[j+1] then
            /* swap them */
            swap( list[j], list[j+1] )
            swapped = true
         end if

      end for
```

```
      /*if no number was swapped that means
       array is sorted now, break the loop.*/

       if(not swapped) then
          break
       end if

   end for

end procedure return list
```

Implementation

```java
// method to perform the bubble sort
public void bubbleSort(int[] array) {
  int size = array.length;
  // loop to access each array element
  for (int step = 0; step < array.length; ++step) {
    int swapped = 0;
    // check if swapping occurs
    for (int i = 0; i < size - step - 1; ++i) {
       // compare two array elements
       // change > to < to sort in descending order
       if (array[i] > array[i + 1]) {
          // swapping occurs if
          // the first element is greater than second
          int temp = array[i];
          array[i] = array[i + 1];
          array[i + 1] = temp;
          swapped = 1;
       }
    }
    // no swapping means the array is already sorted
    // so no need of further comparison
    if (swapped == 0) {
       break;
    }
  }
}
```

```java
@Test
public void TestBubbleSort() {
   int[] array = { 5, 2, 8, 3, 6, 23, 56, 33, 66, 25, 7 };

   System.out.println("Before sort:");
   for (int i : array) {
     System.out.print(i + " ");
   }

   BubbleSort bubbleSort = new BubbleSort();
   bubbleSort.bubbleSort(array);
   System.out.println();

   System.out.println("After sort:");
   for (int i : array) {
      System.out.print(i + " ");
   }
}
```

Output:

```
Before sort:
5 2 8 3 6 23 56 33 66 25 7
After sort:
2 3 5 6 7 8 23 25 33 56 66
```

Section 3

Selection Sort

Introduction

- **Selection sort** is a simple sorting algorithm.
- This sorting algorithm is an **in-place comparison-based** algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end.
  - ✓Initially, the sorted part is empty and the unsorted part is the entire list.
  - ✓The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array.
  - ✓This process continues moving unsorted array boundary by one element to the right.
- This algorithm is not suitable for large data sets as its average and worst case complexities are of O(n2), where n is the number of items.

How Selection Sort Work?

- Consider the following depicted array as an example.

- For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

- *So we replace 14 with 10*. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

- For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

How Selection Sort Work?

- After two iterations, two least values are positioned at the beginning in a sorted manner.

- The same process is applied to the rest of the items in the array. Following is a pictorial depiction of the entire sorting process

Algorithm for Selection Sort

- ***Step 1:*** *Array arr with N size*
- ***Step 2:*** *Initialise i=0*
- ***Step 3:*** *If(i<N-1) Check for any element arr[j] where j>i and arr[j]<arr[i] then Swap arr[i] and arr[j]*
- ***Step 4:*** *i=i+1 and Goto Step 3*
- ***Step 5:*** *Exit*

Pseudocode

```
procedure selection sort
   list  : array of items
   n     : size of list

   for i = 1 to n - 1
   /* set current element as minimum*/
      min = i

      /* check the element to be minimum */

      for j = i+1 to n
         if list[j] < list[min] then
            min = j;
         end if
      end for

      /* swap the minimum element with the current element*/
      if indexMin != i  then
         swap list[min] and list[i]
      end if
   end for

end procedure
```

Implementation in Java

```java
public void selectionSort(int arr[]) {
   int n = arr.length;

   // One by one move boundary of unsorted sub-array
   for (int i = 0; i < n - 1; i++) {
      // Find the minimum element in unsorted array
      int min_idx = i;
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[min_idx])
         min_idx = j;
      }

      // Swap the found minimum element with
      // the first element
      int temp = arr[min_idx];
      arr[min_idx] = arr[i];
      arr[i] = temp;
   }
}
```

```java
@Test
public void TestSelectionSort() {
   int[] array = { 5, 2, 8, 3, 6, 23, 56, 33, 66, 25, 7 };

   System.out.println("Before sort:");
   for (int i : array) {
      System.out.print(i + " ");
   }

   SelectionSort sSort = new SelectionSort();
   sSort.selectionSort(array);
   System.out.println();

   System.out.println("After sort:");
   for (int i : array) {
      System.out.print(i + " ");
   }
}
```

Output:

```
Before sort:
5 2 8 3 6 23 56 33 66 25 7
After sort:
2 3 5 6 7 8 23 25 33 56 66
```

Lesson Summary

- Sorting Algorithms
- Bubble Sort
- Selection Sort

THANK YOU!