INTRODUCTION TO SORTING ALGORITHMS

1. Definitions

- Sorting can be defined as any process of systematically arranging items of any kind.

- Sorting either means:

1. **Ordering**: arranging items in a sequence ordered by some criterion;
2. **Categorizing:** grouping items with similar properties and or behaviors.

In computer science, arranging in an ordered sequence is called "**sorting**". And it is a common operation in many applications.

The most common uses of sorting in computing :

- Lookup or searching : Finding a record in a data base
- Merging of sequences: Copy and pasting, file transfers .
- Data processing in a defined order: Request handling by **queue** and **priorities**

The opposite of sorting or rearranging a sequence of items is a way of arranging data in a random or meaningless order and is called **shuffling**.

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

2. Some Common Sorting Algorithms

- Selection Sort **(Our focus)**
- Bubble Sort **(Our focus)**
- Insertion Sort**(Our focus)**
- Merge Sort
- Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort

- Bucket Sor

3. Sorting Terminology

. In-place sorting

An in-place sorting algorithm uses constant extra space for producing the output (modifies the given array only). It sorts the list only by modifying the order of the elements within the list.
Known in-place sorting algorithms are :, **Insertion Sort** and **Selection Sort**s as they do not use any additional space for sorting the list and in the same way a typical implementation of **Merge Sort** and **counting sort are** not in-place, not in-place sorting algorithms.

. Internal and External Sorting

When all data that needs to be sorted cannot be placed in-memory at a time, the sorting is called **external sorting**. External Sorting is used for massive amount of data. Merge Sort and its variations are typically used for external sorting. Some external storage like hard-disk, CD,...are used for external storage, on the other hand
When all data is placed in-memory for processing, then the applied sorting algorithm is called **internal sorting**.

. Stable sorting

Stability is mainly important when we have key value pairs with duplicate keys possible (like people names as keys and their details as values). And we wish to sort these objects by keys,then the sorting algorithm is said to be **stable** if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted

Example : {

[“**domicique**”,”Maried-1.7metres,male”],

[“**domicique**”,”Mukeru-Maried-1.7metres-male”],

[“atanase”,”Maried-1.7metres=male”],

[“mugunga”,”Maried-1.7metres-male”]

}

If you can sort this algorithm in a way that the first element will continue to be before the second element then your sorting is stable :

4. **Things to sort**

A number of things can be sorted depending on the application area:

- **In farming:** you may sort your crops potatoes,bananas or your animals Pigs,cows,dogs
- **In the kitchen:** various types of cereal, dry pasta, beans,...
- In Business: money, notes,coins
- **information architecture (Our focus) :** Alphabetically — simply, in alphabetical order
- Chronologically — by time; for instance by when the document was created or what year was the report concerned
- Magnitude — some kind of ranking; for example from the lowest price to the highest, from the shortest nails to the longest

Sorting containers:

- Open space,Trash beans, baskets,**Data Structures (Arrays,Lists,nodes,….)**

Every day applications of sorting:

- In Cash processing
- School result proclamation
- While eating
- Toy clean-up

Usual Sorting criteria:

- color,shape , weight, size ,test,….

Array Sorting Example:

var theArray={**29**, 64, 73, 34, **20**};

Let us sort this array using an algorithm that divide the array into two sub arrays: sorted and unsorted then take the first element and swap it with the Smollett element of the unsorted array producing the sorted array.

Step 0. **29**, 64, 73, 34, **20.**
Step 1. 20, **64**, 73, 34, **29.**
Step 2. 20, 29, **73**, **34**, 64.
Step 3. 20, 29, 34, **73**, **64.**
Step 4. 20, 29, 34, 64, 73.

Note : It took us exactly 5 steps as the number of elements to complete the sorting of the entire collection

As you can see sorting is not only fun, but it's also building those important early mathematical and scientific reasoning skills.

Why Sorting Algorithms are Important

Since sorting can often reduce the complexity of a problem, it is an important algorithm in Computer Science. These algorithms have direct applications in searching , database management, divide and conquer methods, data structure efficient usage, and …

1. Bubble Sort Algorithm (**S**inking Sort**)**

Bubble Sort is a simple algorithm which is used to sort a given set of **n** elements provided in a form of an array with **n** number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

For example , If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on until when no more elements to compare.

If we have total **n** elements, then we need to repeat this process for n-1 times.

Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

This algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list, where for every complete iteration the largest or smallest element in the given array, **bubbles** up towards the last place or the highest index of the unsorted array, just like a water bubble rises up to the water surface.

Implementing Bubble Sort Algorithm

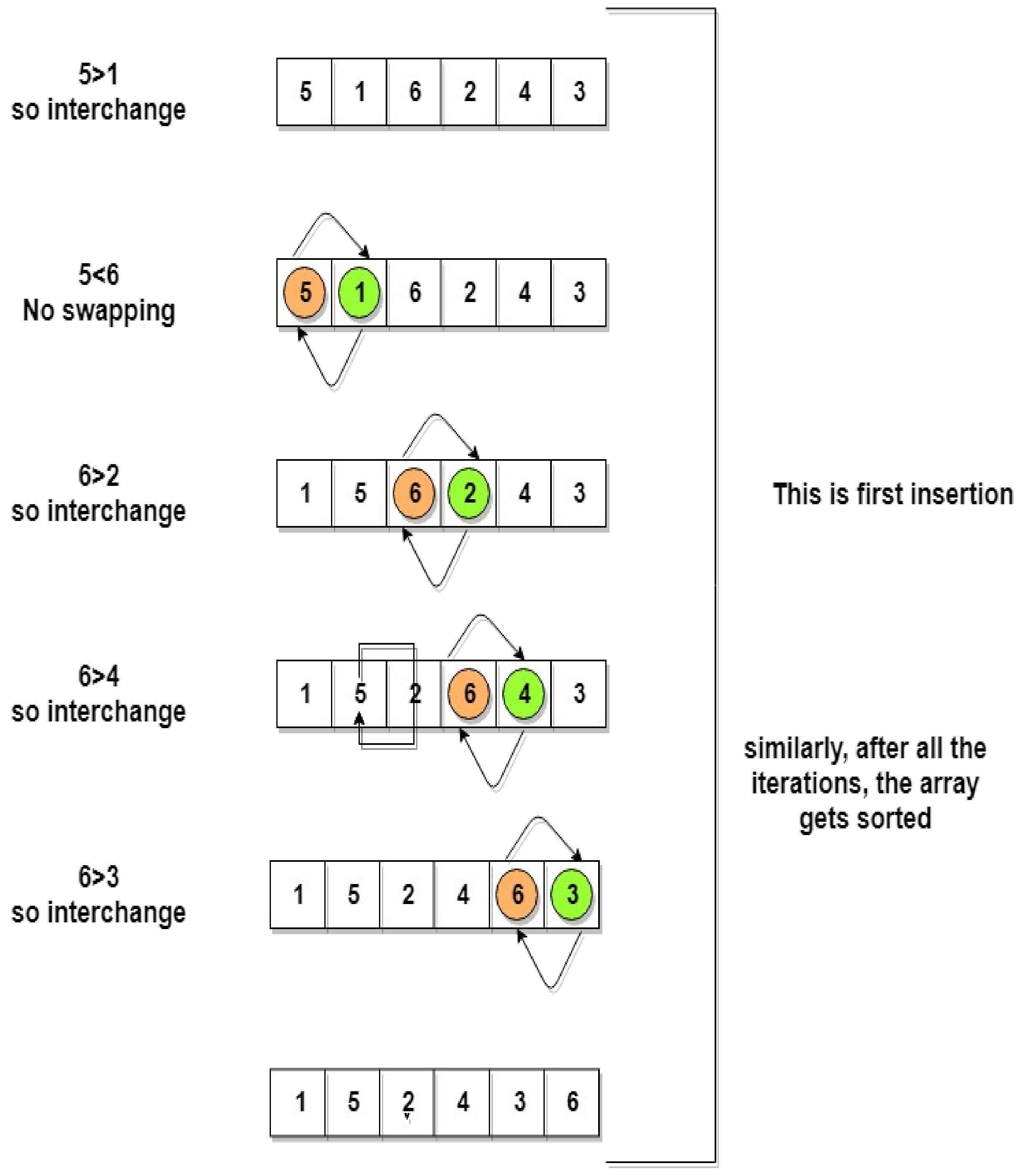
Following are the steps involved in bubble sort(for sorting a given array in ascending order):

1. Starting with the first element(index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. **Repeat Step 1**.

4. After each visit or each comparison of two adjustment elements, the cursor is moved to the next position

Let's consider an array with **6** values **{5, 1, 6, 2, 4, 3}**

Below, the representation of how the first iteration in bubble sort will look like to sort the given array .



So as we can see in the representation above, after the first iteration, 6 is placed at the last index, which is the correct position for it,in similar ways, for each iteration the remaining greatest element will be placed at last index of the remaining unsorted array; which means:” **after the second iteration, 5 will be at the second last index**” and continue until the last iteration the entire array will be sorted.

Pseudo codes that can work for any programming languages:

```
start Bubble Sorting the array
start looping using for loop in equal array elements iterations:
 do…  **for** all elements of array using index i
if array[i] > array[i+1]
  **swap** array[i] with array[i+1]
end if condition
end **for** loop
  return or print out the sorted array.
end of bubble Sorting task.
```

Note: In the algorithm above, we ignore what swapping technique is conducted, but we are sure that in the middle of the process some many swapping will happen.

Pseudo code for c like languages:

```
// Declare global variables.

Var  array[];
var n=array .length; // array .length();

   var  i, j, temp;

// start sorting, ie. Visiting each elements from the first to the last

   for(i = 0; i < n-1; i++)
   {
      for(j = 0; j < n-i-1; j++)
      {
          if( array[j] > array[j+1])
          {
             // swap the elements when condition match

             temp = array[j];

             array[j] = array[j+1];

             array[j+1] = temp;
          }
       }
   }

// done sorting.

 // printing out  the sorted array

   for( i = 0; i < n; i++)
   {
      print array[i] ;
   }


// done printing the sorted array
```

Note : What we can say, I that the above algorithm is not optimized as we may try to sort an already sorted array. Optimization will be dealt later in this course.

Assignement :

Configuring   C development environment environment:

Reference:
https://www.tutorialspoint.com/cprogramming/c_environment_setup.htm

Possible Compilers: GCC,Turbo c++,Dev ++