SORTING ALGORITHMS

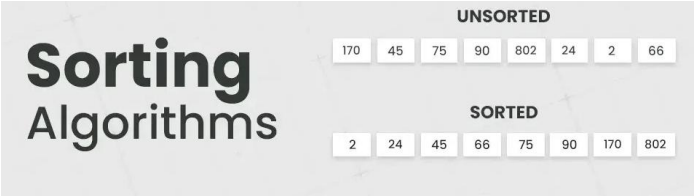
DCIT25 - Data Structures and Algorithms

Prepared by:
Mariella R. Leyba, MIT

Sorting Algorithm

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

For Example: The below list of characters is sorted in increasing order of their ASCII values. That is, the character with a lesser ASCII value will be placed first than the character with a higher ASCII value



What is Sorting

Sorting refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure. Sorting means reordering of all the elements either in ascending or in descending order.

Sorting Terminology

- **In-place Sorting:** An in-place sorting algorithm uses constant 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.

  Examples: Selection Sort, Bubble Sort Insertion Sort and Heap Sort.

- **Internal Sorting:** Internal Sorting is when all the data is placed in the main memory or internal memory. In internal sorting, the problem cannot take input beyond its size.

  Example: heap sort, bubble sort, selection sort, quick sort, shell sort, insertion sort.

Sorting Terminology

- **External Sorting :** External Sorting is when all the 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 the massive amount of data.

  Examples: Merge sort, Tag sort, Polyphase sort, Four tape sort, External radix sort, etc.

- **Stable sorting:** When two same data appear in the same order in sorted data without changing their position is called stable sort.

  Examples: Merge Sort, Insertion Sort, Bubble Sort.

- **Unstable sorting:** When two same data appear in the different order in sorted data it is called unstable sort.

  Examples: Quick Sort, Heap Sort, Shell Sort.

Characteristics of Sorting Algorithms

- **Time Complexity:** Time complexity, a measure of how long it takes to run an algorithm, is used to categorize sorting algorithms. The worst-case, average-case, and best-case performance of a sorting algorithm can be used to quantify the time complexity of the process.
- **Space Complexity:** Sorting algorithms also have space complexity, which is the amount of memory required to execute the algorithm.
- **Stability:** A sorting algorithm is said to be stable if the relative order of equal elements is preserved after sorting. This is important in certain applications where the original order of equal elements must be maintained.
- **In-Place Sorting:** An in-place sorting algorithm is one that does not require additional memory to sort the data. This is important when the available memory is limited or when the data cannot be moved.
- **Adaptivity:** An adaptive sorting algorithm is one that takes advantage of pre-existing order in the data to improve performance.

Applications of Sorting Algorithms

1. **Searching Algorithms:** Sorting is often a crucial step in search algorithms like binary search, Ternary Search, where the data needs to be sorted before searching for a specific element.
2. **Data management:** Sorting data makes it easier to search, retrieve, and analyze.
3. **Database optimization:** Sorting data in databases improves query performance.
4. **Machine learning:** Sorting is used to prepare data for training machine learning models.
5. **Data Analysis:** Sorting helps in identifying patterns, trends, and outliers in datasets. It plays a vital role in statistical analysis, financial modeling, and other data-driven fields.
6. **Operating Systems:** Sorting algorithms are used in operating systems for tasks like task scheduling, memory management, and file system organization.

1. Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

In Bubble Sort algorithm,

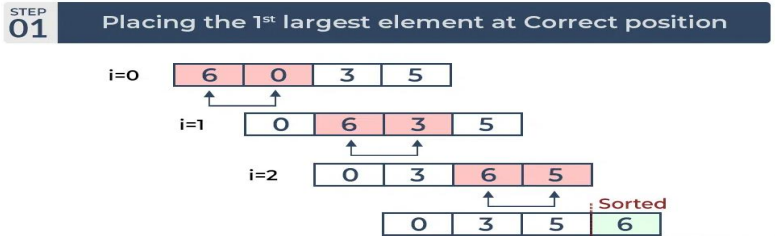
- Traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so on until the data is sorted.

Let us understand Bubble Sort

Input: arr[] = {6, 0, 3, 5}

First Pass:

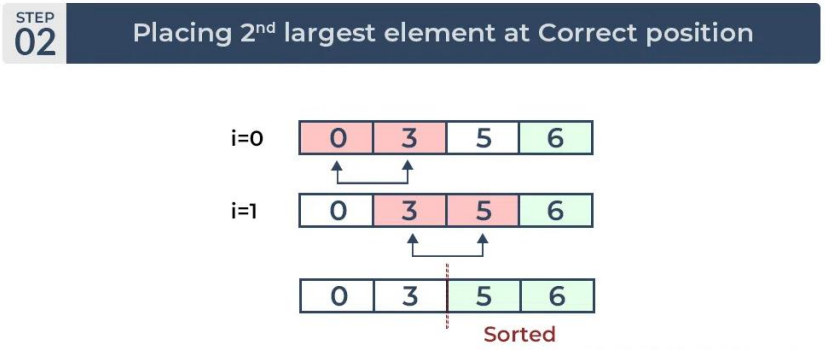
The largest element is placed in its correct position, i.e., the end of the array.



Let us understand Bubble Sort

Second Pass:

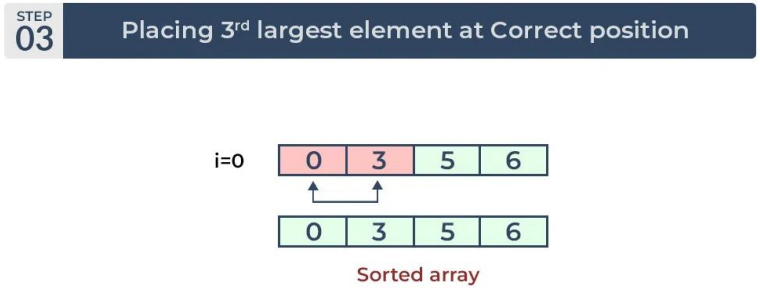
Place the second largest element at correct position



Let us understand Bubble Sort

Third Pass:

Place the remaining two elements at their correct positions.



Advantages of Bubble Sort:

- Bubble sort is easy to understand and implement.
- It does not require any additional memory space.
- It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the sorted output.

Disadvantages of Bubble Sort:

- Bubble sort has a time complexity of O(N2) which makes it very slow for large data sets.
- Bubble sort is a comparison-based sorting algorithm, which means that it requires a comparison operator to determine the relative order of elements in the input data set. It can limit the efficiency of the algorithm in certain cases.

2. Insertion Sort

- **Insertion sort** is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted output.
- Insertion sort is like sorting playing cards in your hands. You split the cards into two groups: the sorted cards and the unsorted cards. Then, you pick a card from the unsorted group and put it in the right place in the sorted group.

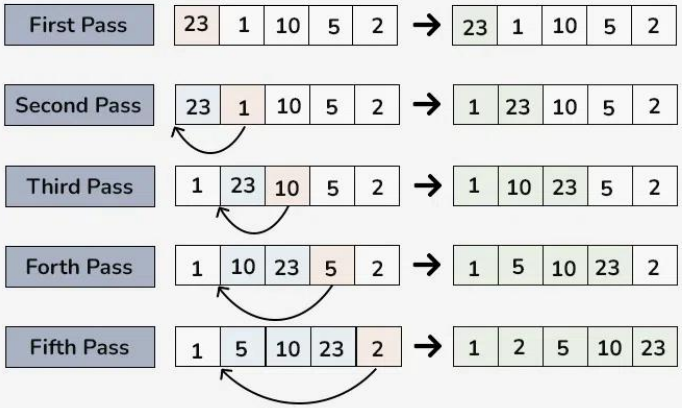
Algorithm

To achieve insertion sort, follow these steps:

1. We have to start with second element of the array as first element in the array is assumed to be sorted.
2. Compare second element with the first element and check if the second element is smaller then swap them.
3. Move to the third element and compare it with the second element, then the first element and swap as necessary to put it in the correct position among the first three elements.
4. Continue this process, comparing each element with the ones before it and swapping as needed to place it in the correct position among the sorted elements.
5. Repeat until the entire array is sorted.

Working of Insertion Sort Algorithm:

Consider an array having elements: {23, 1, 10, 5, 2}



3. Selection Sort

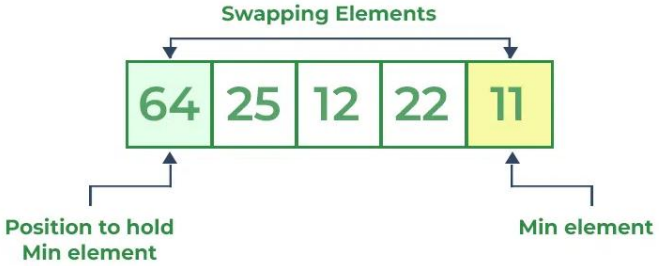
- Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.
- The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

How does Selection Sort Algorithm work?

Lets consider the following array as an example: arr[] = {64, 25, 12, 22, 11}

First pass:

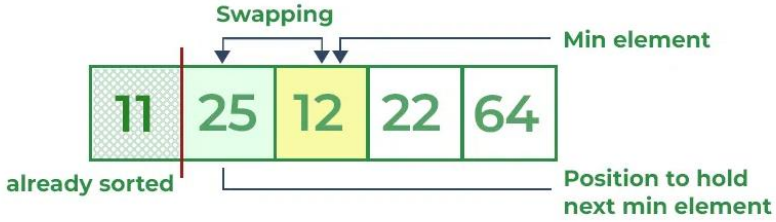
- For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.
- Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array



How does Selection Sort Algorithm work?

Second Pass:

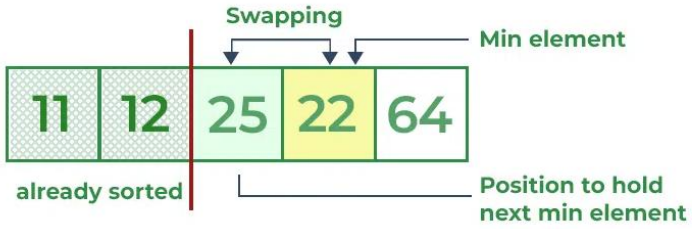
- For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.
- After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



How does Selection Sort Algorithm work?

Third Pass:

- Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.
- While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



How does Selection Sort Algorithm work?

Fourth pass:

- Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array
- As 25 is the 4th lowest value hence, it will place at the fourth position.

How does Selection Sort Algorithm work?

Fifth Pass:

- At last the largest value present in the array automatically get placed at the last position in the array
- The resulted array is the sorted array.

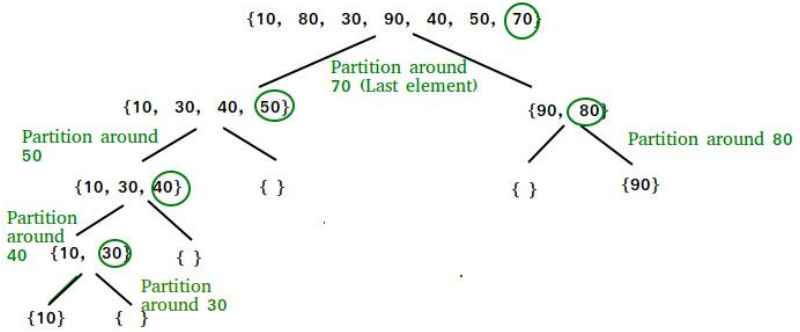
Sorted array

4. Quick Sort

Quick Sort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

How does QuickSort work?

- The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot.
- Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



Choice of Pivot

There are many different choices for picking pivots.

1. Always pick the first element as a pivot.
2. Always pick the last element as a pivot (implemented below)
3. Pick a random element as a pivot.
4. Pick the middle as the pivot.

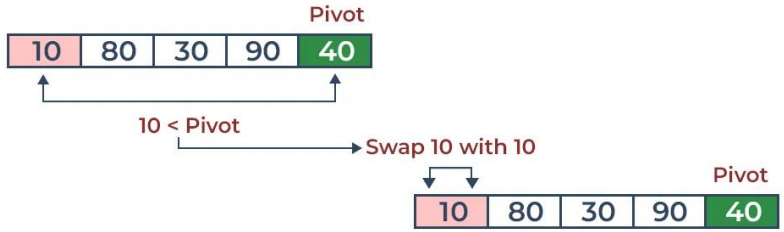
Partition Algorithm:

- The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.

Partition Algorithm:

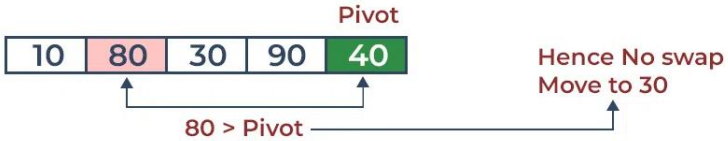
Consider: arr[] = {10, 80, 30, 90, 40}.

- Compare 10 with the pivot and as it is less than pivot arrange it accrodingly.

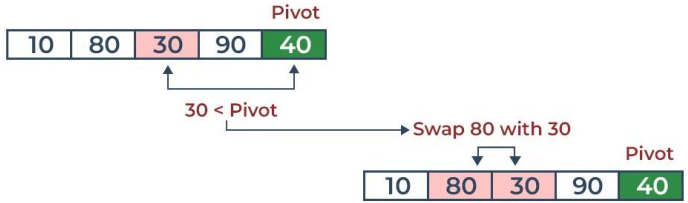


Partition Algorithm:

- Compare 80 with the pivot. It is greater than pivot.

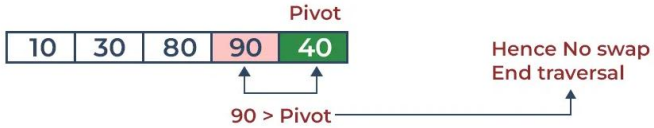


- Compare 30 with pivot. It is less than pivot so arrange it accordingly.

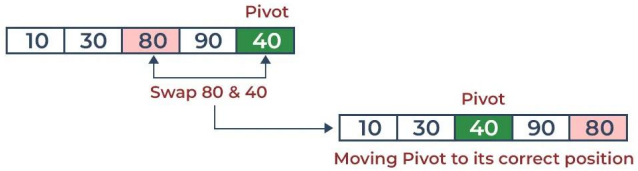


Partition Algorithm:

- Compare 90 with the pivot. It is greater than the pivot.

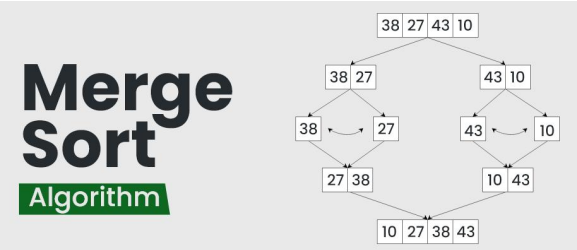


- Arrange the pivot in its correct position.



5. Merge Sort

- Merge sort is a sorting algorithm that follows the divide-and-conquer approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.
- In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



How does Merge Sort work?

- Merge sort is a popular sorting algorithm known for its efficiency and stability. It follows the divide-and-conquer approach to sort a given array of elements.

Here's a step-by-step explanation of how merge sort works:

1. Divide: Divide the list or array recursively into two halves until it can no more be divided.
2. Conquer: Each subarray is sorted individually using the merge sort algorithm.
3. Merge: The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.

Illustration of Merge Sort

Let's sort the array or list [38, 27, 43, 10] using Merge Sort

Let's look at the working of above example:

Divide:

[38, 27, 43, 10] is divided into [38, 27] and [43, 10].

[38, 27] is divided into [38] and [27].

[43, 10] is divided into [43] and [10].

Conquer:

[38] is already sorted.

[27] is already sorted.

[43] is already sorted.

[10] is already sorted.

Merge:

Merge [38] and [27] to get [27, 38].

Merge [43] and [10] to get [10,43].

Merge [27, 38] and [10,43] to get the final sorted list [10, 27, 38, 43]

Therefore, the sorted list is [10, 27, 38, 43].

THANK YOU!