Lesson:

Selection Sort

PW SKILLS

Pre-Requisites

- Loops
- Arrays

List of Concepts Involved

- Selection Sort Algorithm
- Example Explanation
- Selection Sort Code
- Selection Sort Time and Space Complexity
- Is selection sort stable?
- Applications of selection sort

As we are now familiar with the concept of Bubble sort from the previous lesson, we will expand our knowledge further to learn one more type of sorting viz. selection sort.

Topic : Selection Sort Algorithm

This **algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array.

- The subarray which is already sorted.
- The remaining subarray was unsorted

As the name suggests, in the Selection Sort algorithm we select the index having minimum element by traversing through the array and swap it with the current index, then increment the current index. This way we keep on traversing the rest of the array and finding the minimum index and swap it with the current index. This will result in forming a sorted array on the left side.

- Iterate through the array and consider the current element index.
- Now traverse the rest of the array elements and find the minimum element index.
- Swap it with the current element index and increment the current element index by one.
- Repeat until we reach the end of the array.

Working of Selection Sort

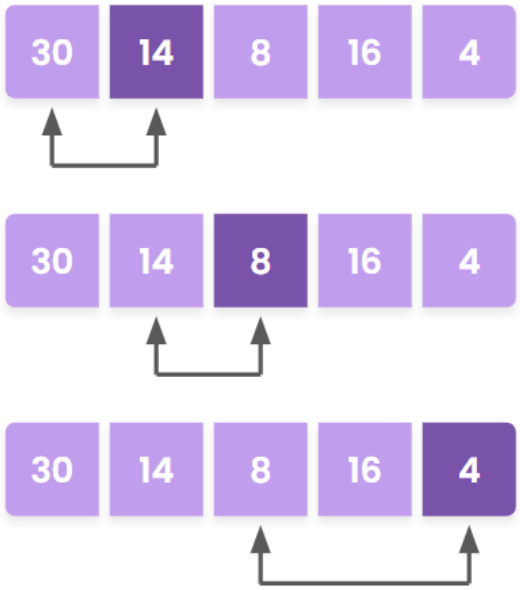
In selection sort we select an element for a given index, and let say if we are selecting an element for i'th index then it should have <=i element which are smaller or equal to it and remaining greater to it. This whole thing is difficult for any random index but for the first or last index we can do it easily as this will be the minimum or maximum valued index for them respectively. and once the minimum (or maximum) element is placed correctly then we can ignore it and do the same for remaining array of size n-1

1. Set the first element as `minimum`.

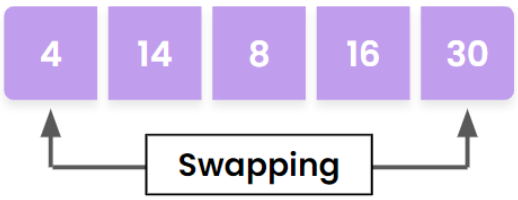
2. Compare `minimum` with the second element. If the second element is smaller than `minimum`, assign the second element as `minimum`.

   Compare `minimum` with the third element. Again, if the third element is smaller, then assign `minimum` to the third element otherwise do nothing.

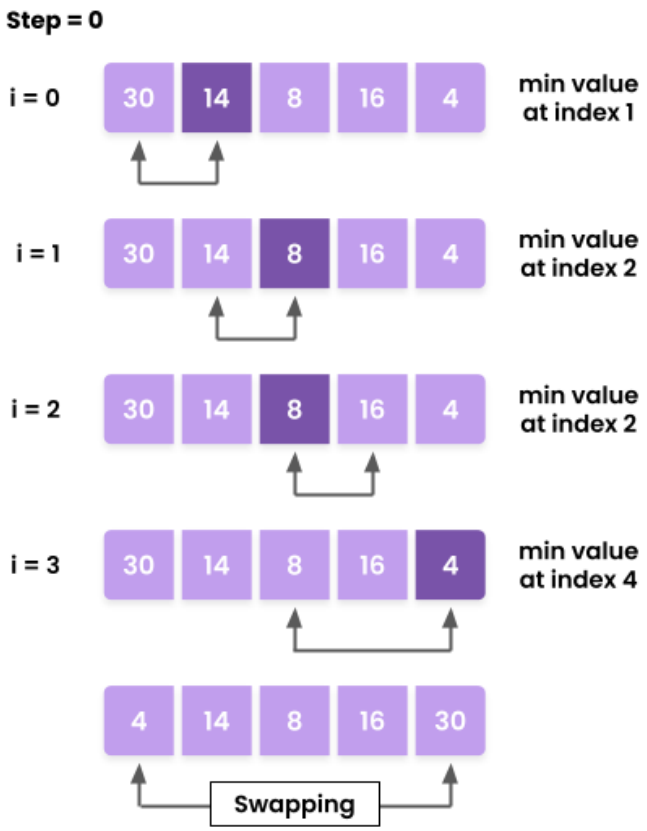
   The process goes on until the last element.



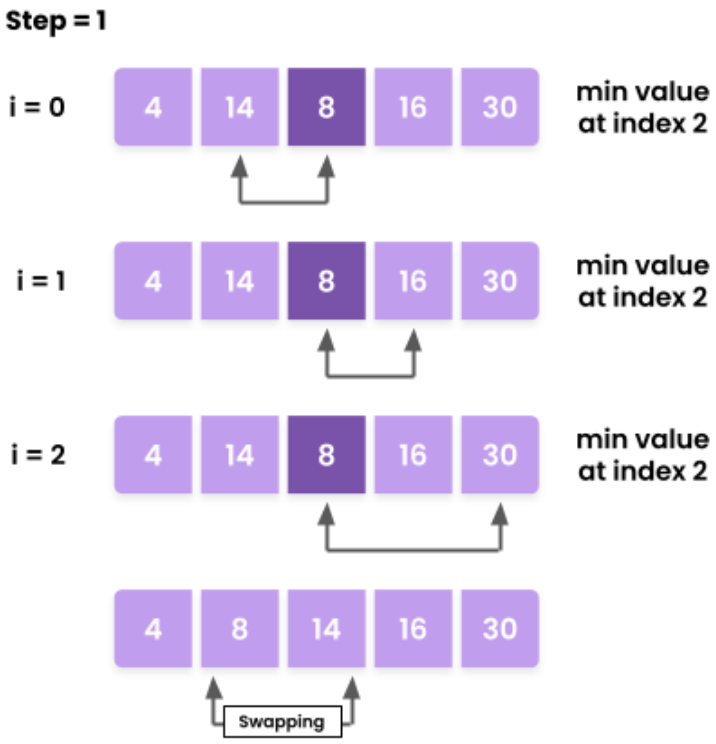
3. After each iteration, `minimum` is placed in the front of the unsorted list.



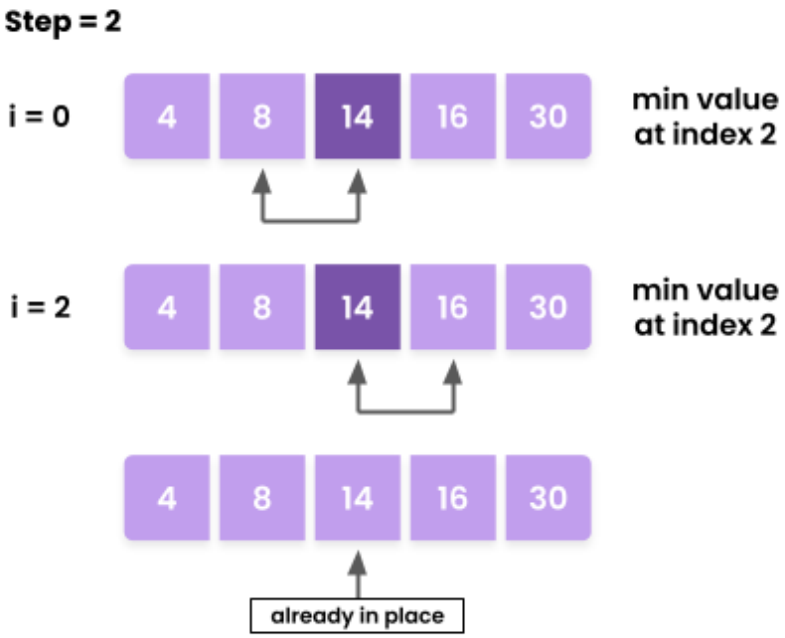
4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.



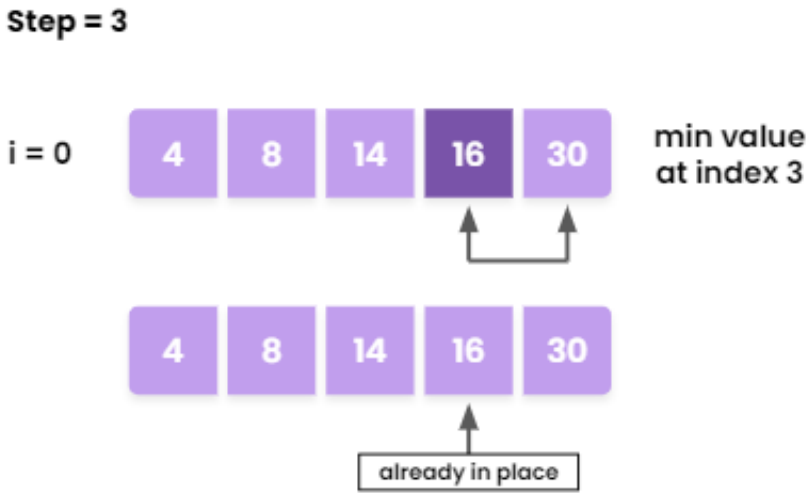
The first iteration



The second iteration



The third iteration

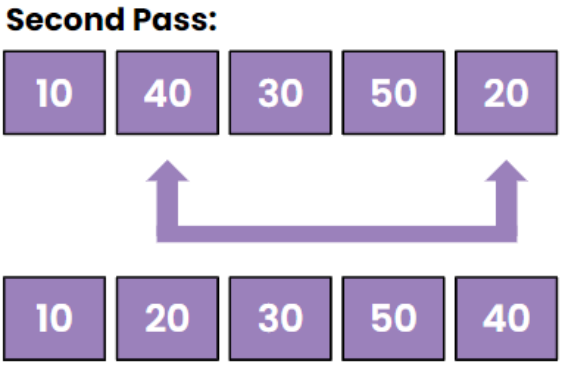


Example: Sort the given array using Selection Sort Algorithm.
Array = [10, 40, 30, 50, 20]

In the above example, as we iterate through the array from left to right, we keep finding the minimum index element and keep swapping it with the current element index and then increment the current index. Repeat the process until we reach the end of the array. This way we will have a sorted array at the end.
// taking 0 based indexing

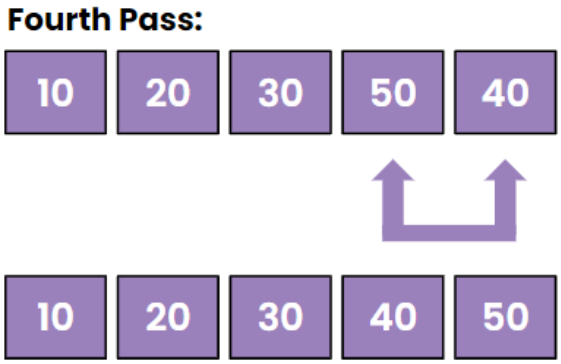
First Pass: Current index is 0 and current element is 10. Now just take min_index to be the current index and traverse through the rest of the array and update when we find a more minimal value. This way we have our minimum value index and swap it. In this case, the minimum value is 10 only so no need to swap anything and just increment the iterator by one.

Second Pass: Now the current element is 40. We find the minimum value index by traversing through the rest of the array and find that the minimum value in the rest of the array is 20 with index being 4. (taking 0-based indexing). Thus we swap the value at current index i.e. 40 with minimum index value i.e. 20. And move the iterator by one.
Array after second pass: [10, 20, 30, 50, 40]



Third Pass: The current element now becomes 30 and the minimum value is found by traversing the rest of the array and here we found that 30 is the minimum value only so no need to swap it with anything and simply increment the iterator by one.

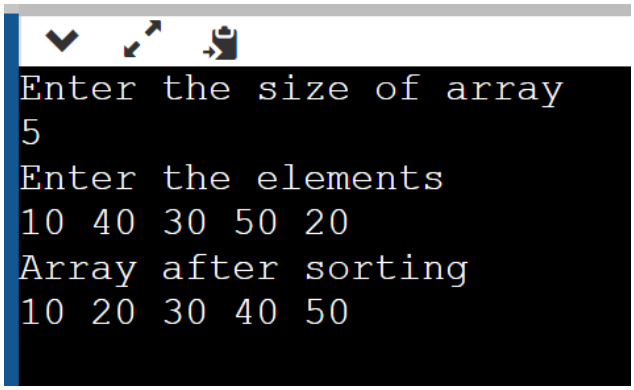
Fourth Pass: The current element now is 50. And we find the minimum value in rest of the array is 40 so we swap it with current value index and increment the iterator by one
Array after fourth pass: [10, 20, 30, 40, 50]



This way we have our sorted array at the end.

Topic : Selection Sort Code

https://pastebin.com/KXtXKLYC



Topic : Selection Sort Time Complexity

The time complexity of the following algorithm will be $O(n^2)$ as at each particular index from left to right as we are iterating in an array, we are finding the minimum value index by traversing through the rest of the array. Thus we require O(n) to find the minimum value index and we are doing this process n time so final time complexity will be **$O(n^2)$**.

Topic : Selection Sort Space Complexity

It does not require any extra space so space complexity is O(1). Thus it will be an in-place sorting algorithm.

Topic : Is selection sort stable?

As we know(Explained in previous lesson) A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appeared in the input unsorted array.

Selection sort works by finding the minimum element in the array and then inserting it in its correct position by swapping with the element which has minimum value. This is what makes it unstable.

Let's see an example :
arr[] ={Deer,Monkey,Deer,Camel}
In the first iteration of the outer for-loop, the algorithm determines that "Camel"is the minimal element and exchanges it with the blue "Deer" :
arr[] ={Camel,Monkey,Deer,Deer}
Then, it finds that the red Deer is the minimal item in the rest of the array and swaps it with "Monkey":
arr[] ={Camel,Deer,Monkey,Deer}
Finally, since Deer < Monkey in the usual lexicographic order, Selection Sort swaps the blue Deer with Monkey:
arr[] ={Camel,Deer,Deer,Monkey}

As you can see, arr doesn't maintain the relative order of the two strings. Since they are equal, the one that was initially before the other should come first in the output array. But, Selection Sort places the red Deer before the blue one even though their initial relative order was opposite.
So, we can conclude that **Selection Sort isn't stable**.

Topic : Selection Sort Applications

The selection sort is preferred when

- the cost of swapping is not an issue
- Number of swaps in the selection sort is of O(N) while O(N^2) in other brute force sorting algorithms.

Upcoming class Teasers:

- Insertion sort and its complexity