Selection Sort Algorithm

Selection sort is an algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

How Selection Sort Works?

1. Set the first element as `minimum`. Select 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

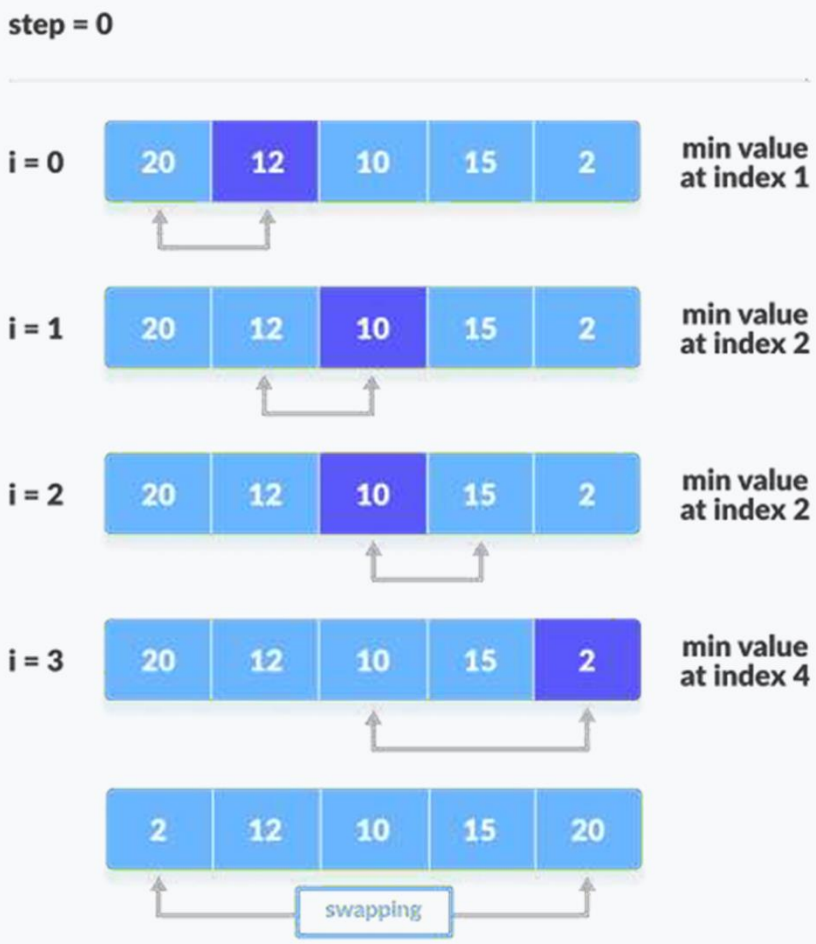
goes on until the last element.

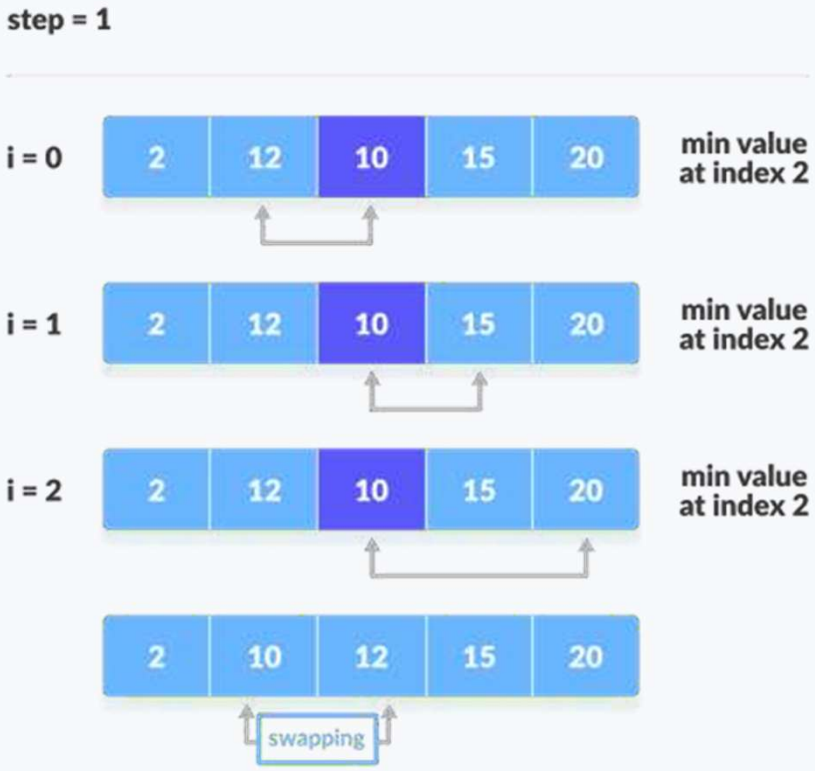
Compare minimum with the remaining

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

Swap the first with minimum

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




step = 1
i = 0
2
12
10
15
20
min value
at index 2
i = 1
2
12
10
15
20
min value
at index 2
i = 2
2
12
10
15
20
min value
at index 2
2
10
12
15
20
swapping


step = 2
i = 0
2
10
12
15
20
min value
at index 2
i = 2
2
10
12
15
20
min value
at index 2
2
10
12
15
20
already in place

Selection Sort Algorithm

```
selectionSort(array, size)  repeat (size - 1) times

set the first unsorted element as the minimum

  for each of the unsorted elements  if element < currentMinimum

    set element as new minimum

  swap minimum with first unsorted position  end selectionSort
```

Complexity

Cycle	Number of Comparison
1st	(n-1)
2nd	(n-2)
3rd	(n-3)

Number of `(n - 1) + (n - 2) + (n - 3) + ..... +`

Complexity $O(n^2)$

Also, we can analyze the complexity by simply observing the number of loops. There are 2 loops so the comp

Time Complexities:

- **Worst Case** $O(n^2)$

  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.

- **Best Case** $O(n^2)$

  It occurs when the array is already

- **Average Case** $O(n^2)$

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

  The time complexity of the selection sort is the same in all cases. At every step, you have to find the minimum element and put it in the right place. The minimum element is not known until the end of the array is

Space Complexity: `O(1` because an extra `tem` is

Selection Sort Applications

The selection sort is used when:

- a small list is to be sorted
- cost of swapping does not matter
- checking of all the elements is compulsory
- cost of writing to a memory matters like in flash memory (number of writes/swaps is `O(n)` as compared to `O(n²)` of bubble sort)

Bubble Sort Algorithm

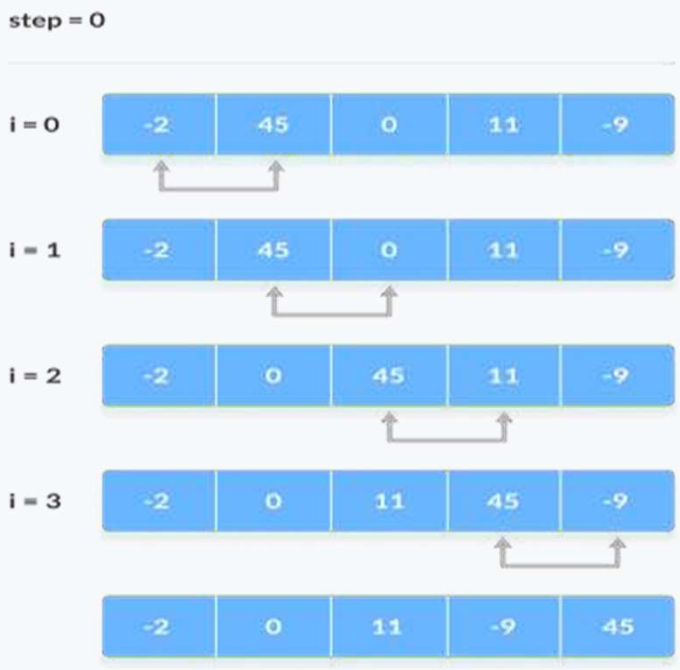
Bubble sort is an algorithm that compares the adjacent elements and swaps their positions if they are not in the intended order. The order can be ascending or descending.

How Bubble Sort Works?

1. Starting from the first index, compare the first and the second elements.If the first element is greater than the second element, they are swapped.

Now, compare the second and the third elements. Swap them if they are not in order.

The above process goes on until the last element.



Compare the adjacent elements

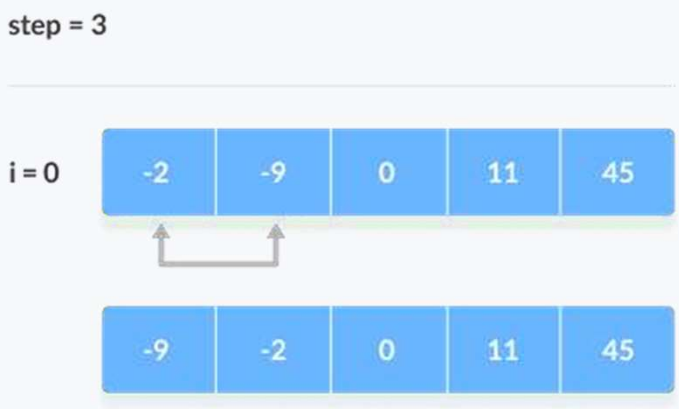
2. The same process goes on for the remaining iterations. After each iteration, the largest element among the unsorted elements is placed at the end.

In each iteration, the comparison takes place up to the last unsorted element.

The array is sorted when all the unsorted elements are placed at their correct

positions Compare

adjacent elements Compar e



the adjacent elements

Compare the adjacent elements

Bubble Sort Algorithm

```
  bubbleSort(array)

    for i <- 1 to indexOfLastUnsortedElement-1  if leftElement > rightElement

      swap leftElement and rightElement

end bubbleSort
```

Complexity

Bubble Sort is one of the simplest sorting algorithms. Two loops are implemented in the algorithm.

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
.....	.....

Number of `(n - 1) + (n - 2) + (n - 3) +.....+ 1 = n(n - 1)`

Complexity $O(n^2)$

Also, we can analyze the complexity by simply observing the number of

There are 2 loops so the complexity `n*n =` is

- **Worst Case Complexity:** $O(n^2)$
  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.
- **Best Case Complexity:** $O(n)$
  If the array is already sorted, then there is no need for sorting.
- **Average Case Complexity:** $O(n^2)$
  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

Space Complexity: `O(1` because an extra variable `tem` is used for

In the optimized algorithm, the variable `swappe` adds to the space complexity thu

Bubble Sort Applications

Bubble sort is used in the following cases where

1. the complexity of the code does not matter.
1. a short code is preferred.