Programiz

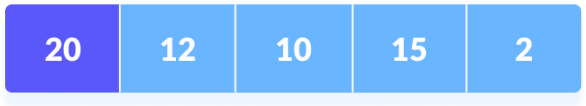
Selection Sort Algorithm

In this tutorial, you will learn how selection sort works. Also, you will find working examples of selection sort in C, C++, Java and Python.

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`.

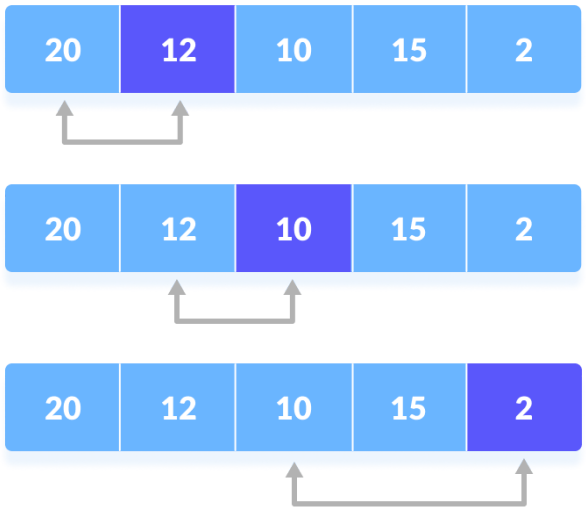


2. Compare `minimum` with the second element. If the second element is smaller than `minimum`, assign second element as `minimum`.

Contents

   Compare `minimum` with the third element. Again, if the third element is smaller, then assign `minimum` to the third

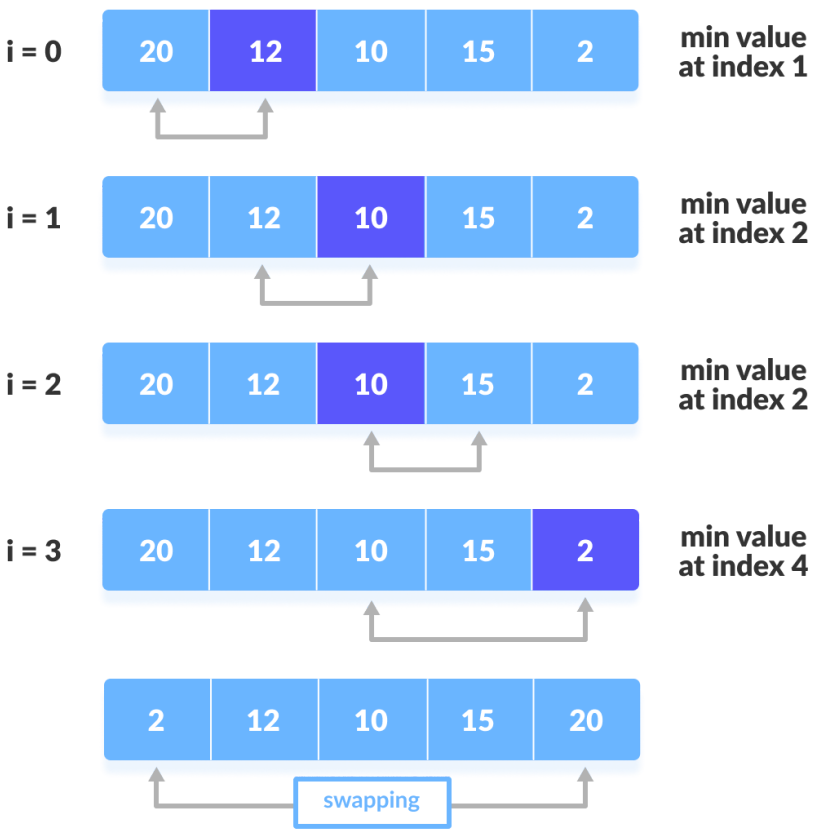
element otherwise do nothing. The process goes on until the last element.

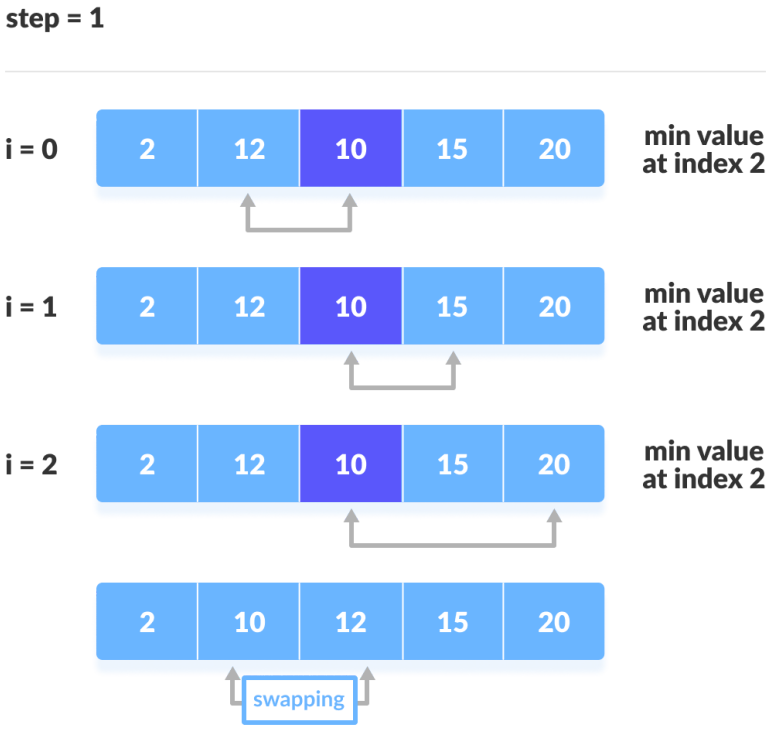


3. After each iteration, `minimum` is placed in the front of the unsorted list.

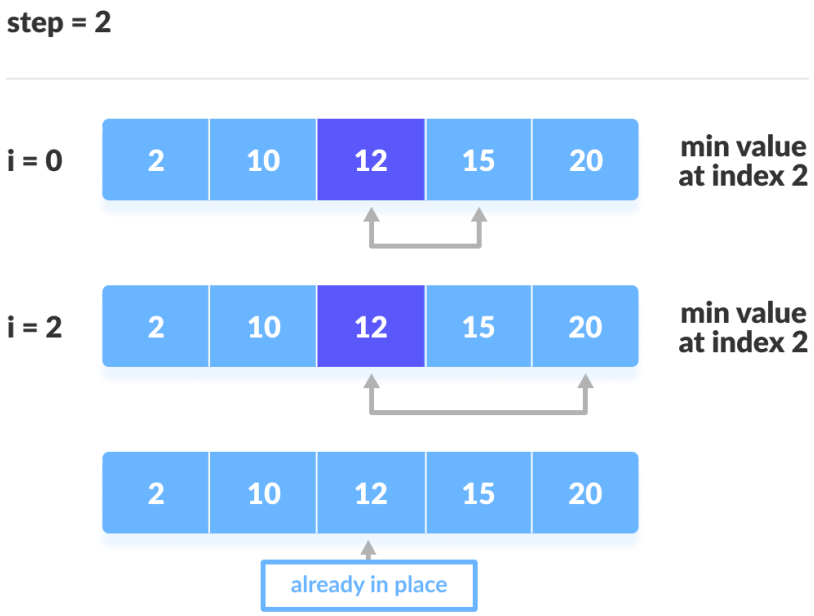
4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

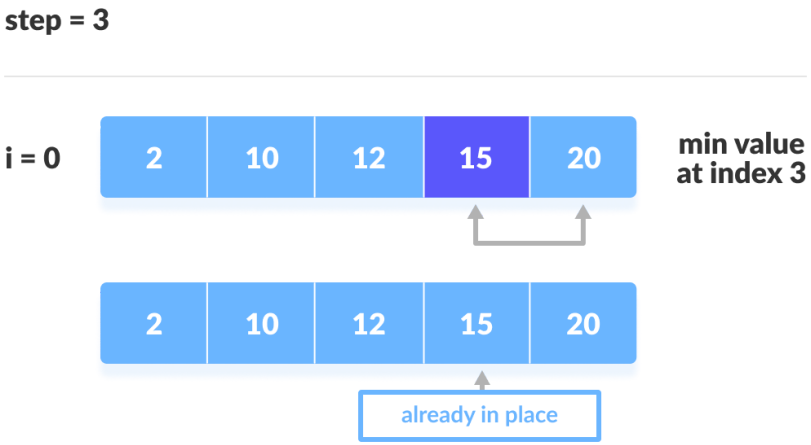
Contents


step = 0
i = 0
20 12 10 15 2
min value at index 1
i = 1
20 12 10 15 2
min value at index 2
i = 2
20 12 10 15 2
min value at index 2
i = 3
20 12 10 15 2
min value at index 4
2 12 10 15 20
swapping


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

Contents





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
```

Python, Java and C/C++ Examples

Contents

Python Java C C+

```python
# Selection sort in Python

def selectionSort(array, size):
    for step in range(size):
        min_idx = step
        for i in range(step + 1, size):

            # To sort in descending order, change > to < in this line.

            if array[i] < array[min_idx]:
                min_idx = i

        (array[step], array[min_idx]) = (array[min_idx], array[step])


data = [-2, 45, 0, 11, -9]
size = len(data)
selectionSort(data, size)
print('Sorted Array in Ascending Order:\n')
print(data)
```

Complexity

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

Contents

Cycle	Number of Comparison
...	...
last	1

Number of comparisons: `(n-1) + (n-2) + (n-3) +.....+ 1 = n(n-1)/2` nearly equals to n^2

Complexity = $O(n^2)$

Also, we can analyze the complexity by simply observing the number of loops. There are 2 loops so the complexity is $n*n = n^2$.

Time Complexities:

- **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:** $O(n^2)$

  It occurs when the the array is already sorted

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

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

The time complexity of selection sort is the same in Contents very step, you have to find the minimum element and put it in the right place. The minimum element until the end of the array is not reached.

Space Complexity:

Space complexity is `O(1)` because an extra variable `temp` is used.

Selection Sort Applications

The selection sort is used when:

- 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)

Data Structure & Algorithms

Bubble Sort Algorithm

Insertion Sort Algorithm

Contents

Contents

Contents

Asymptotic Analysis

Heap Data Structure

Priority Queue

Red-Black Tree

Insertion in a Red-Black Tree

Deletion in a Red-Black Tree

AVL Tree

Contents

Contents