Sort Algorithms

Bubble Sort:

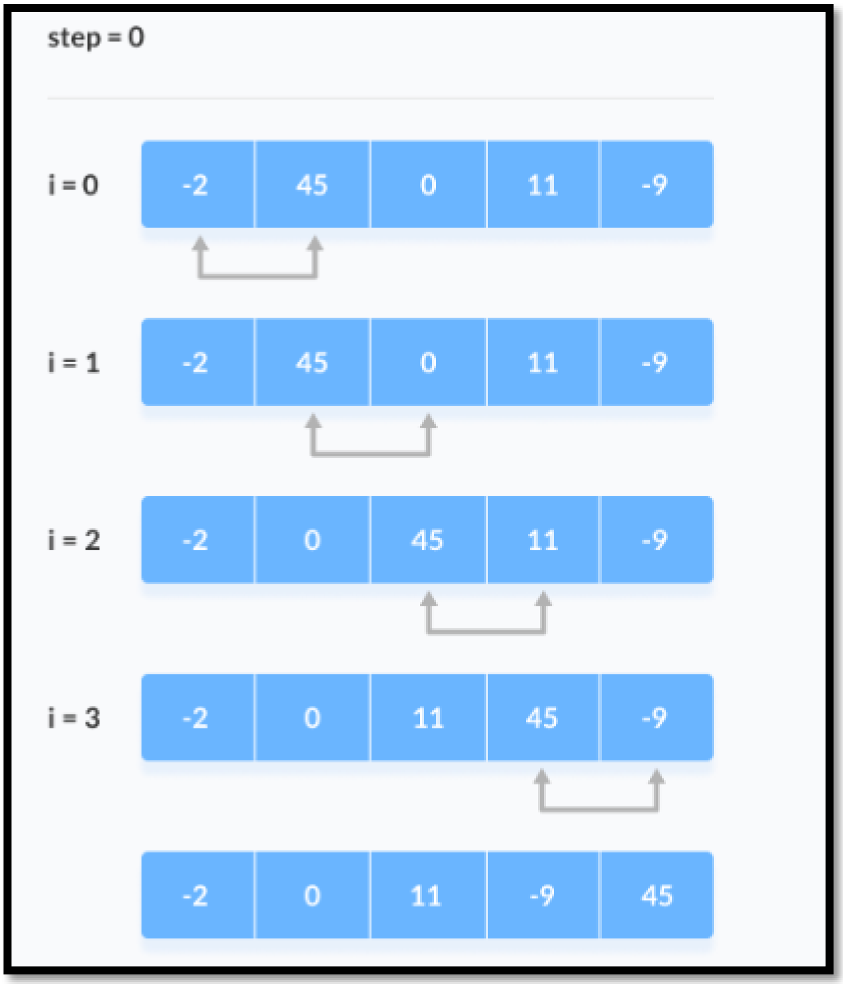
Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

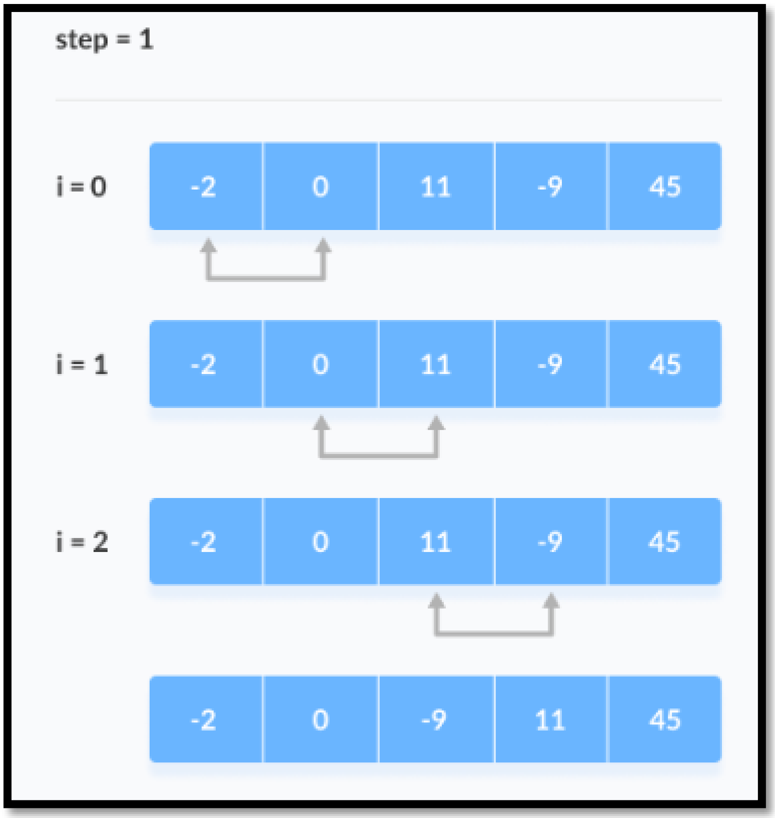
Explanation:

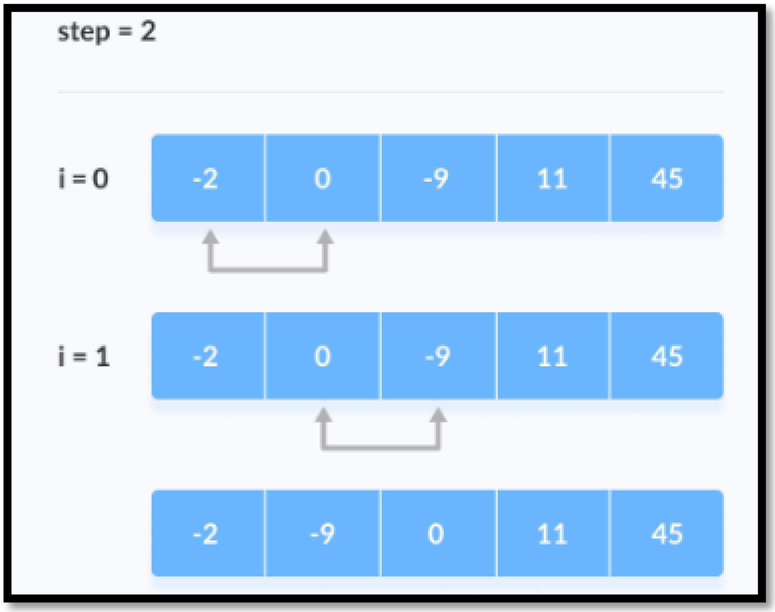
If we magic that we want to sort elements in ascending order, we need to follow three steps which is:

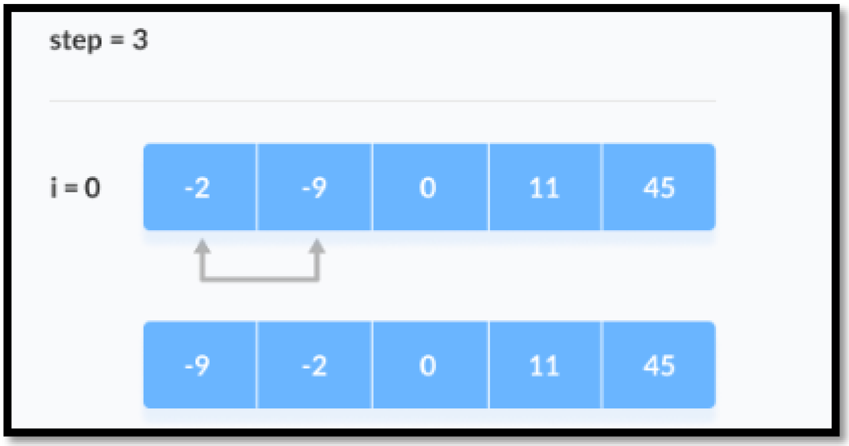
1- As we know bubble sort it is start from the first index, so, firstly it will compare first element with the second element.
2- If the first element is greater than second element it will swapped.
3- Compare second element with third element, if the second element it is greater it will be swapped if they are not in order.
4- The above process goes on until the last element.

Example:

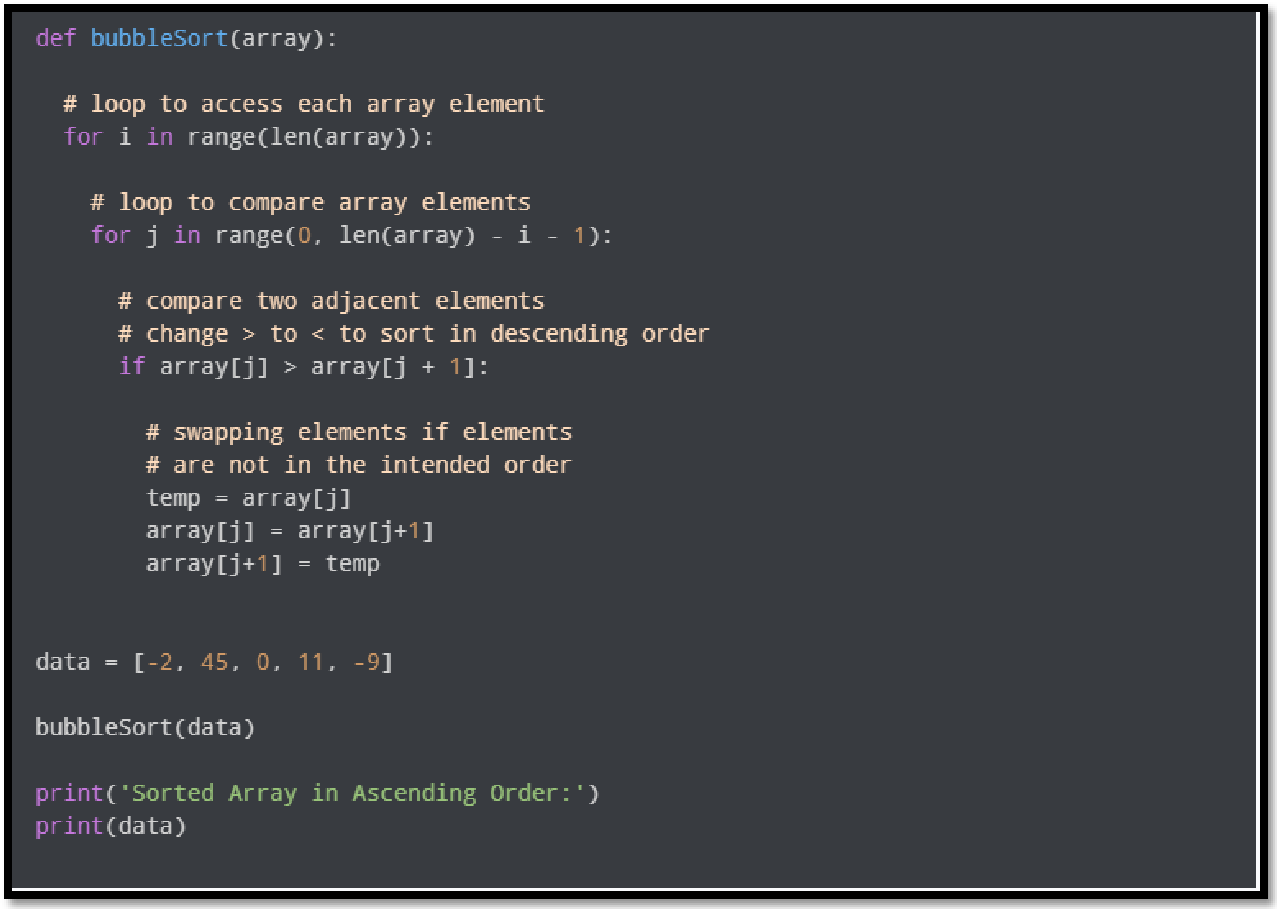

step = 0
i = 0
-2 45 0 11 -9
i = 1
-2 45 0 11 -9
i = 2
-2 0 45 11 -9
i = 3
-2 0 11 45 -9
-2 0 11 -9 45


step = 1
i = 0
-2 0 11 -9 45
i = 1
-2 0 11 -9 45
i = 2
-2 0 11 -9 45
-2 0 -9 11 45


step = 2
i = 0
-2 0 -9 11 45
i = 1
-2 0 -9 11 45
-2 -9 0 11 45



Python Code:



```
def bubbleSort(array):

  # loop to access each array element
  for i in range(len(array)):

    # loop to compare array elements
    for j in range(0, len(array) - i - 1):

      # compare two adjacent elements
      # change > to < to sort in descending order
      if array[j] > array[j + 1]:

        # swapping elements if elements
        # are not in the intended order
        temp = array[j]
        array[j] = array[j+1]
        array[j+1] = temp


data = [-2, 45, 0, 11, -9]

bubbleSort(data)

print('Sorted Array in Ascending Order:')
print(data)
```

Selection Sort:

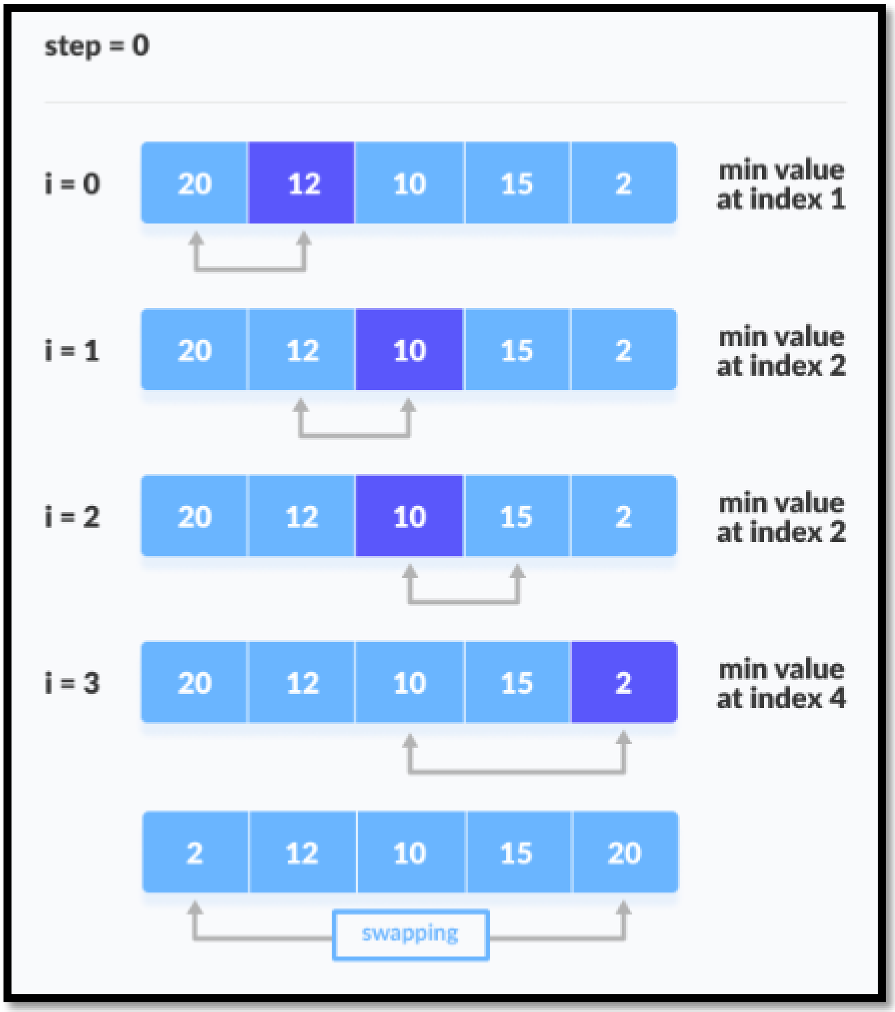
Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

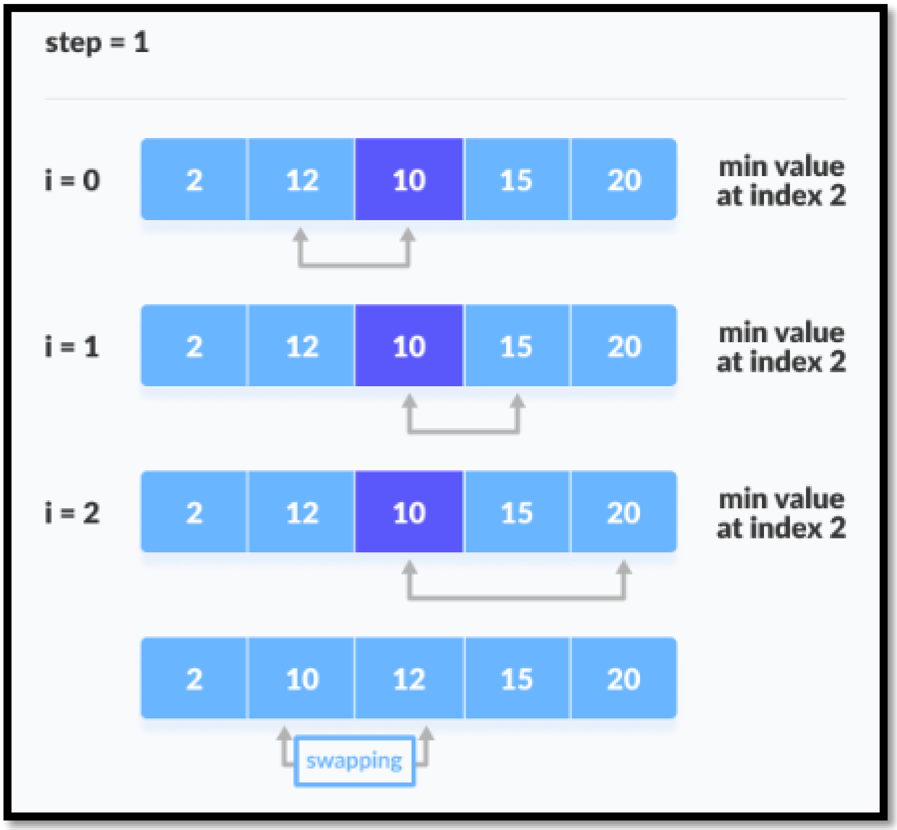
Explanation:

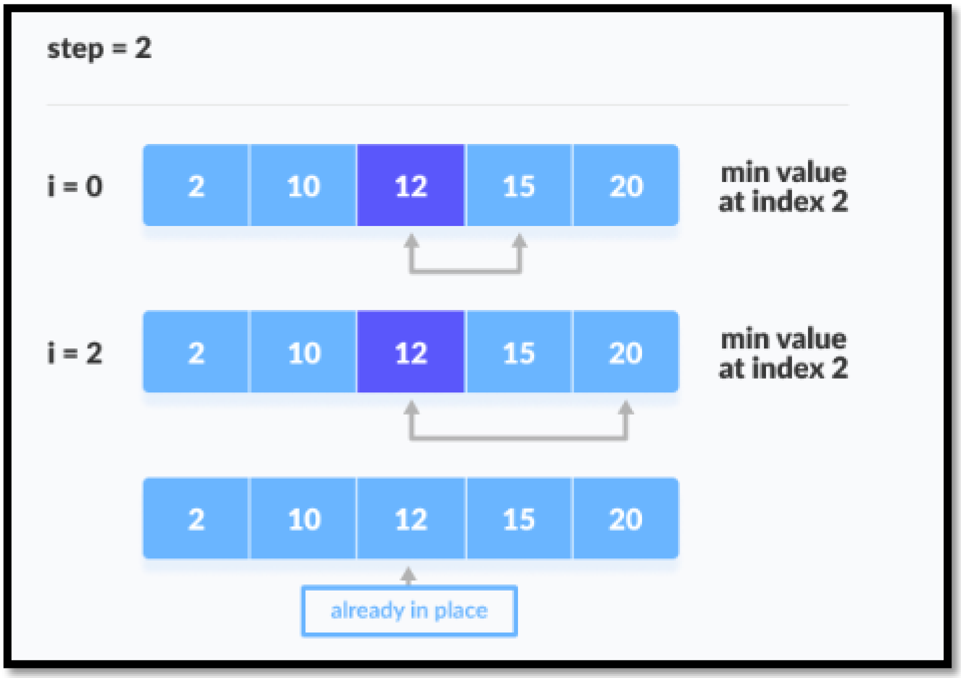
If we magic that we want to sort elements in a list, we need to follow three steps which is:

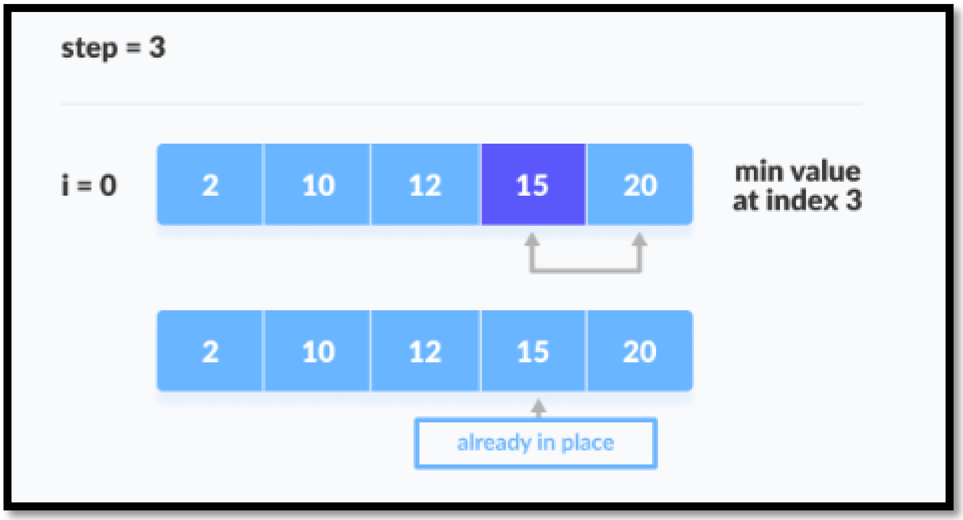
1- Set a first element as minimum.
2- Compare minimum element with second element, if the second element was smaller than minimum assign it as minimum.
3- Compare third element with minimum, if the third element was smaller than minimum assign it as minimum.
4- The above process goes on until the last element.

Example:

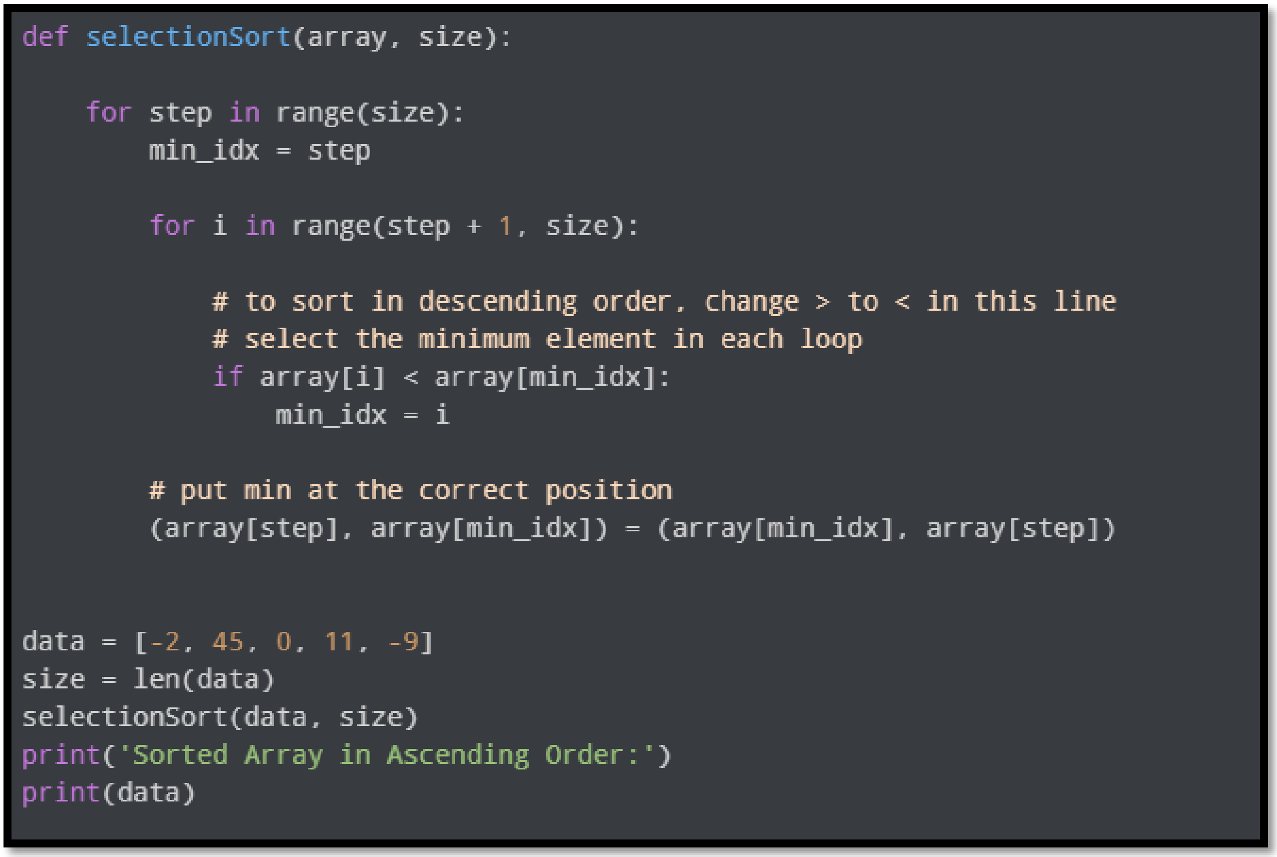

step = 0
i = 0
20
12
10
15
2
min value at index 1
i = 1
20
12
10
15
2
min value at index 2
i = 2
20
12
10
15
2
min value at index 2
i = 3
20
12
10
15
2
min value at index 4
2
12
10
15
20
swapping


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


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



Python Code:



```
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
            # select the minimum element in each loop
            if array[i] < array[min_idx]:
                min_idx = i

        # put min at the correct position
        (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:')
print(data)
```

Quick Sort:

Quicksort is a fast sorting algorithm that works by splitting a large array of data into smaller sub-arrays.

Explanation:

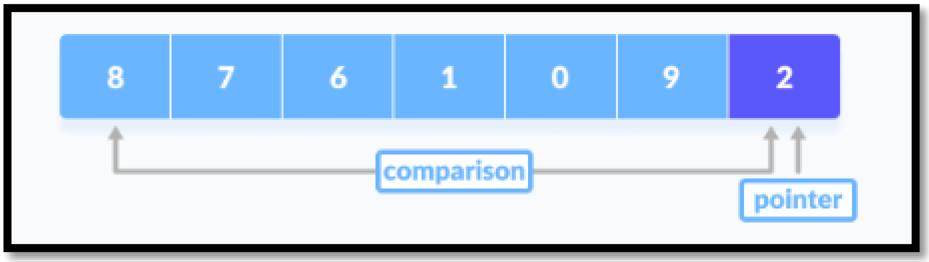
If we magic that we want to sort elements in a list, we need to follow three steps which is:

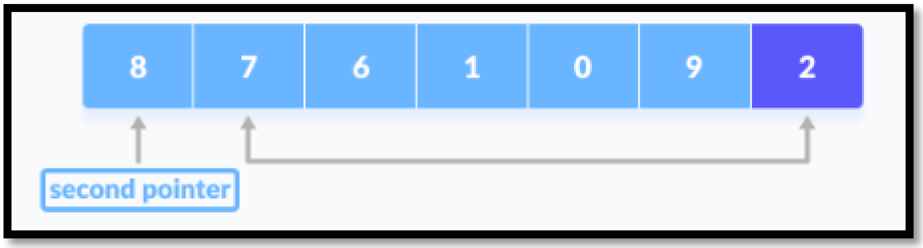
1- **Pick:** Select an element.

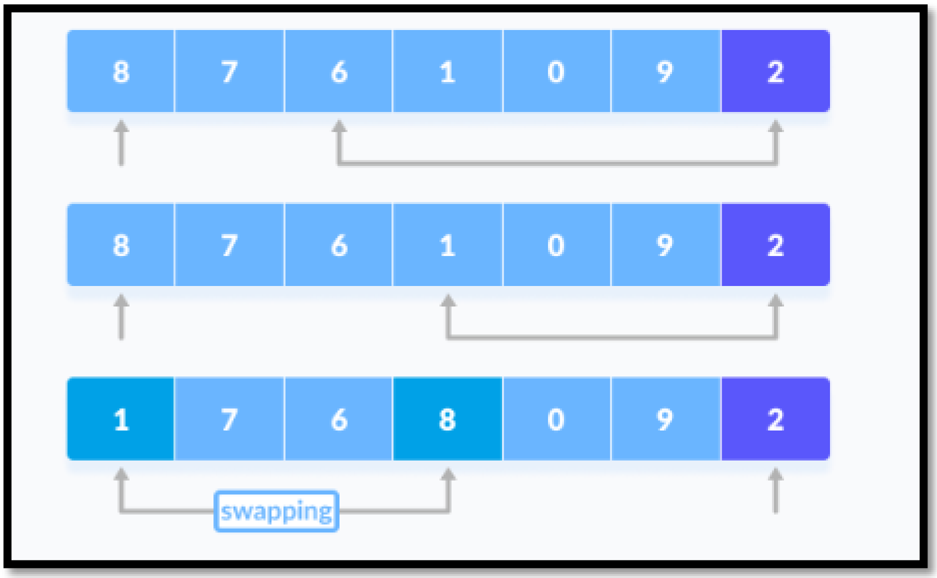
2- Divide: Split the problem set, move smaller parts to the left of the pivot and larger items to the right.

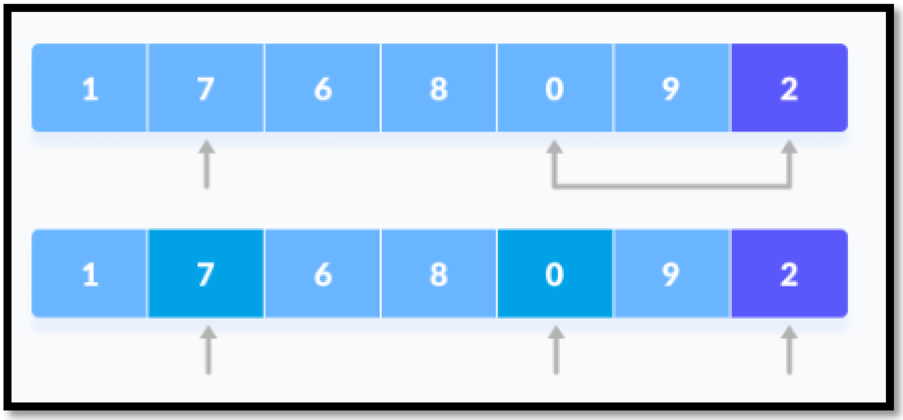
3- **Repeat and combine:** Repeat the steps and combine the arrays that have previously been sorted.

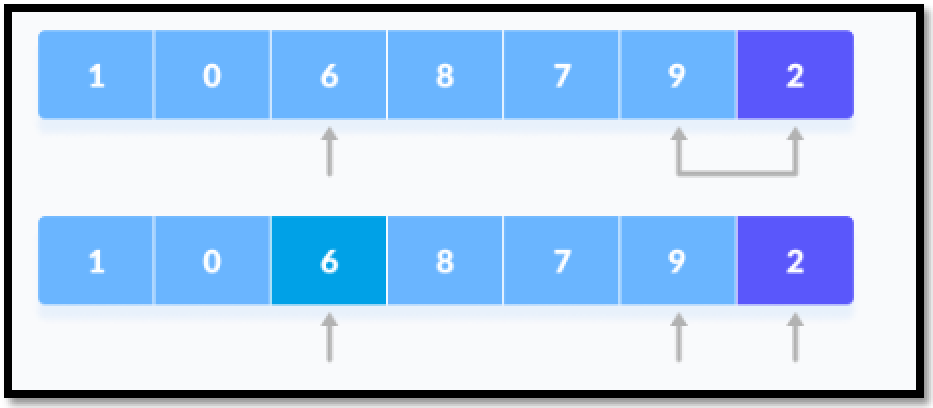
Example:

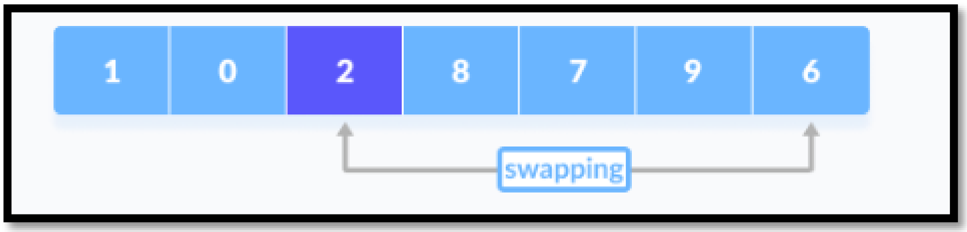


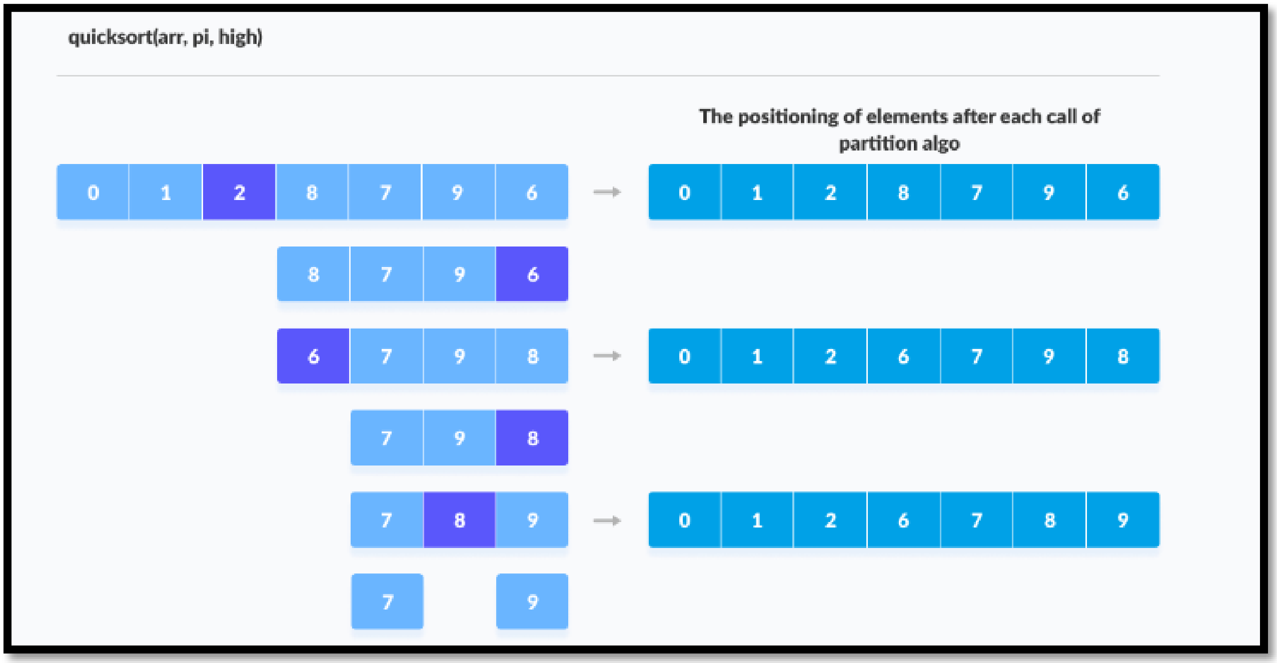

8 7 6 1 0 9 2
second pointer


8 7 6 1 0 9 2
8 7 6 1 0 9 2
1 7 6 8 0 9 2
swapping

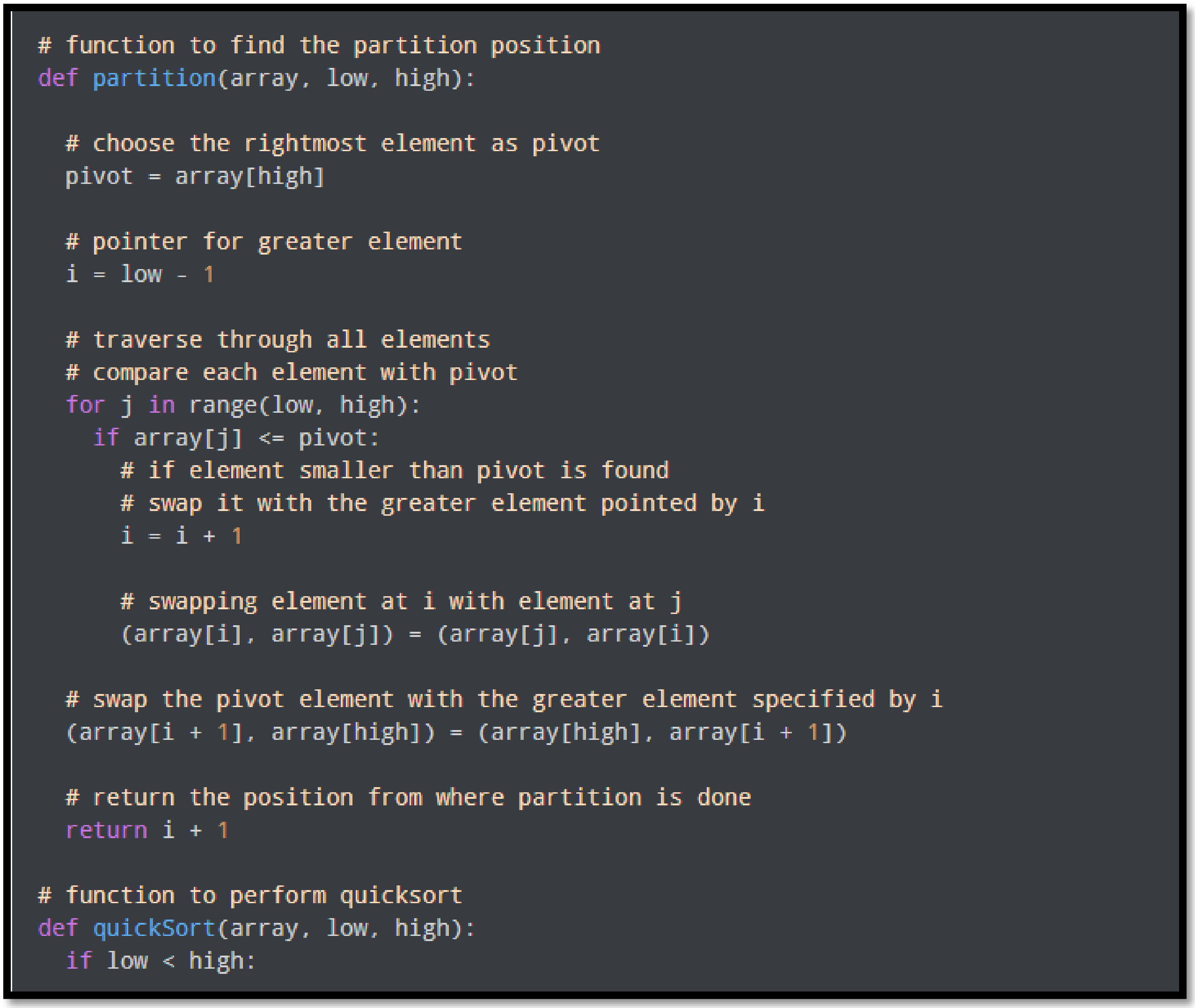

1 7 6 8 0 9 2
1 7 6 8 0 9 2


1 0 6 8 7 9 2
1 0 6 8 7 9 2

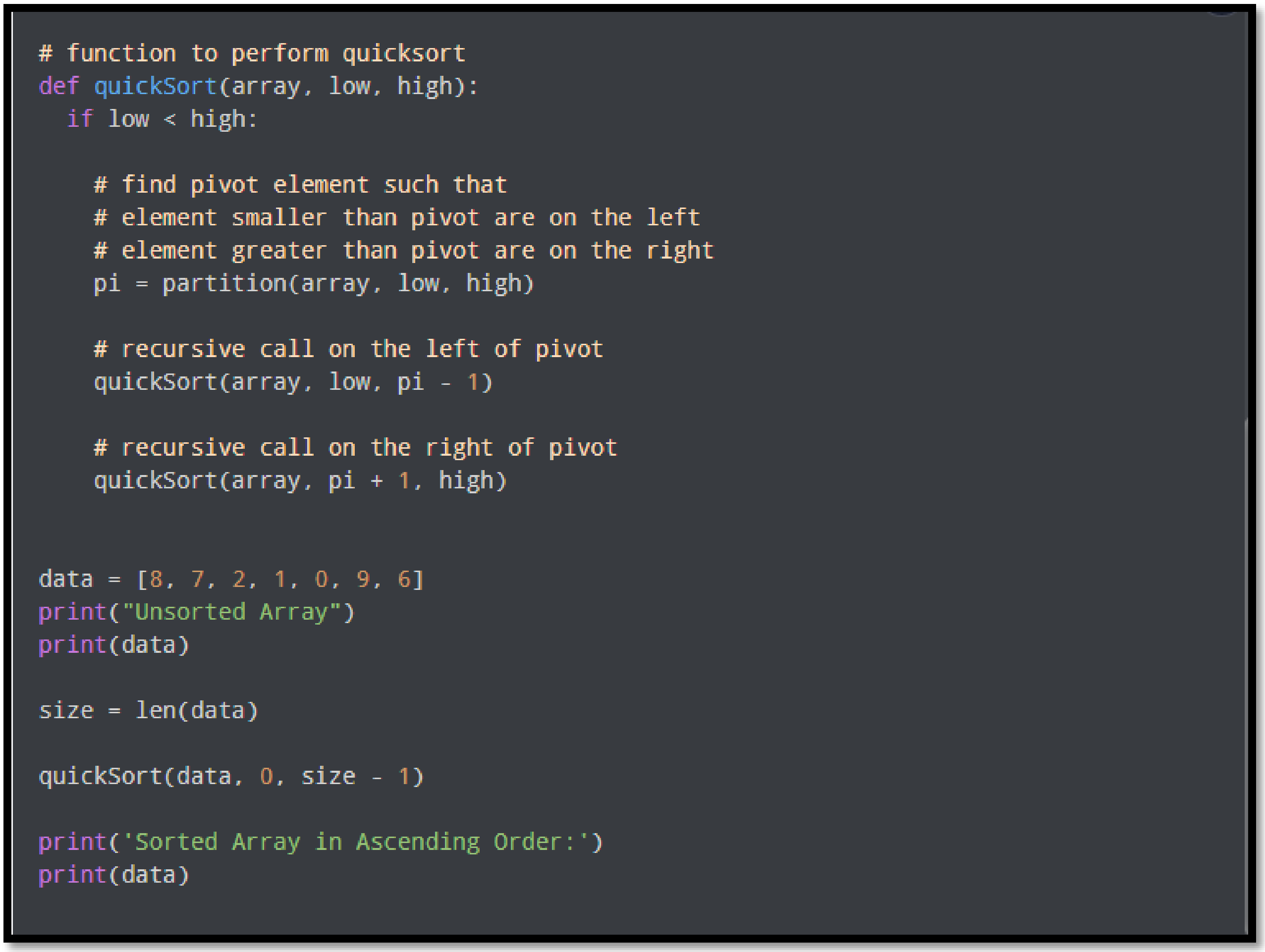

1 0 2 8 7 9 6
swapping


quicksort(arr, pi, high)
The positioning of elements after each call of partition algo
0 1 2 8 7 9 6
0 1 2 8 7 9 6
8 7 9 6
6 7 9 8
0 1 2 6 7 9 8
7 9 8
7 8 9
0 1 2 6 7 8 9
7
9

Python Code:



```python
# function to find the partition position
def partition(array, low, high):

  # choose the rightmost element as pivot
  pivot = array[high]

  # pointer for greater element
  i = low - 1

  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j] <= pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # return the position from where partition is done
  return i + 1

# function to perform quicksort
def quickSort(array, low, high):
  if low < high:
```



```
# function to perform quicksort
def quickSort(array, low, high):
  if low < high:

    # find pivot element such that
    # element smaller than pivot are on the left
    # element greater than pivot are on the right
    pi = partition(array, low, high)

    # recursive call on the left of pivot
    quickSort(array, low, pi - 1)

    # recursive call on the right of pivot
    quickSort(array, pi + 1, high)


data = [8, 7, 2, 1, 0, 9, 6]
print("Unsorted Array")
print(data)

size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')
print(data)
```