Riphah International University
I-14 Main Campus
Faculty of Computing

Class:	Fall-2024	**Subject:**	Data Structures & Algorithms
Course Code:	CS 2124	**Lab Instructor:**	Zeeshan Ali

Learning Objective:

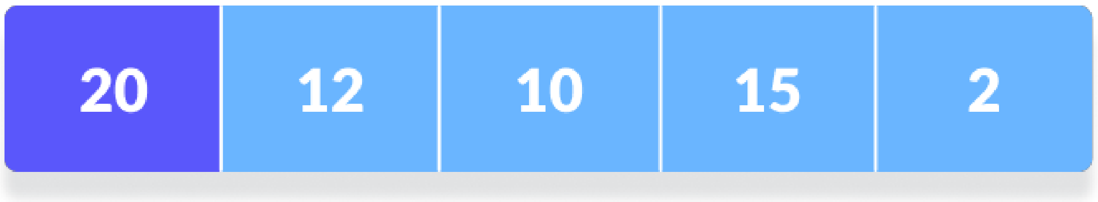
- Selection Sort
- Working of Selection Sort
- Implementation of Selection Sort
- Quick Sort
- Working of Quick Sort
- Implementation of Quick Sort
- Lab Tasks

Selection Sort

In the selection sort technique, the list is divided into two parts. In one part all elements are sorted and in another part the items are unsorted. At first, we take the maximum or minimum data from the array. After getting the data (say minimum) we place it at the beginning of the list by replacing the data of first place with the minimum data. After performing the array is getting smaller. Thus, this sorting technique is done.

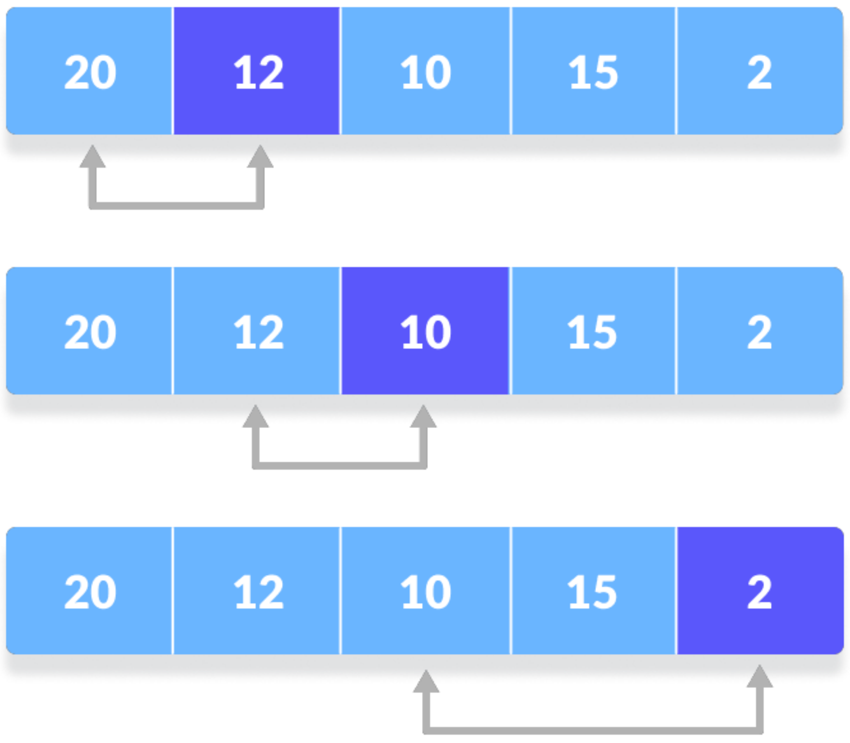
Working of Selection Sort

1. Set the first element as minimum.

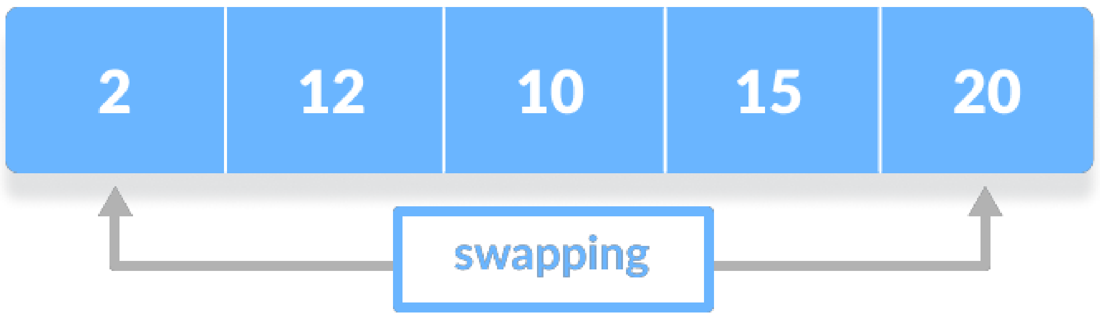


2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

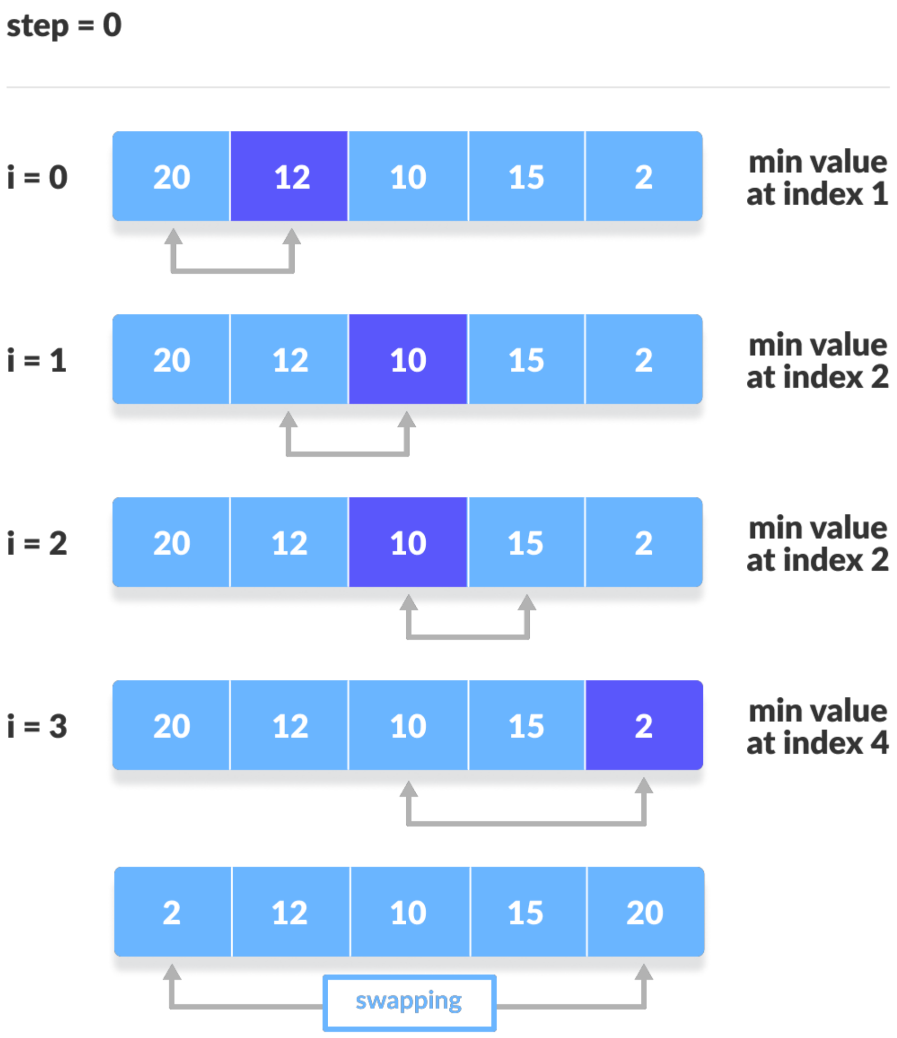
Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.



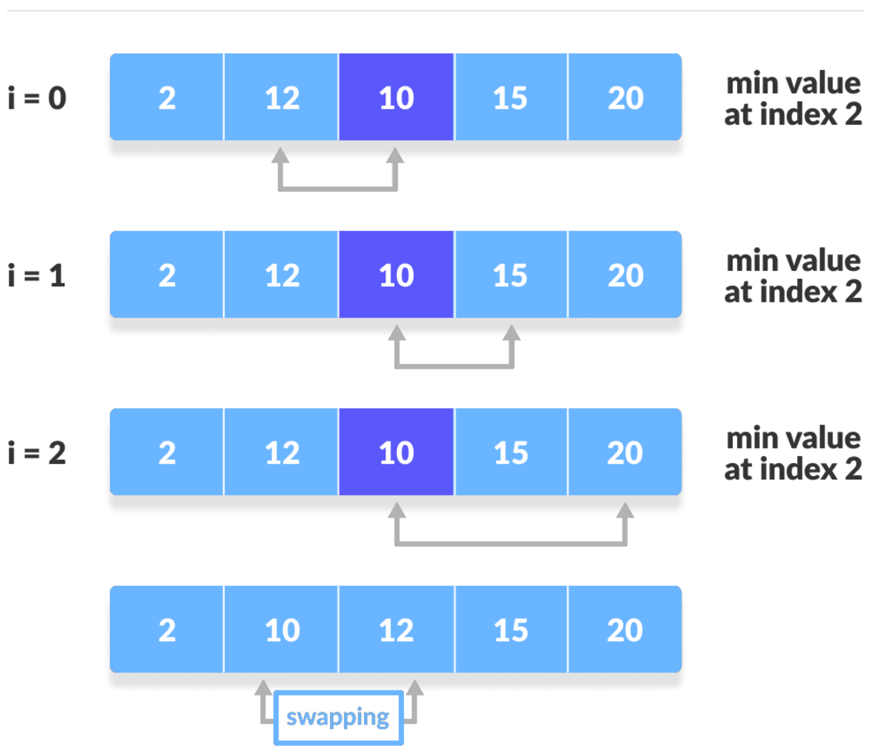
3. After each iteration, minimum is placed in the front of the unsorted list.



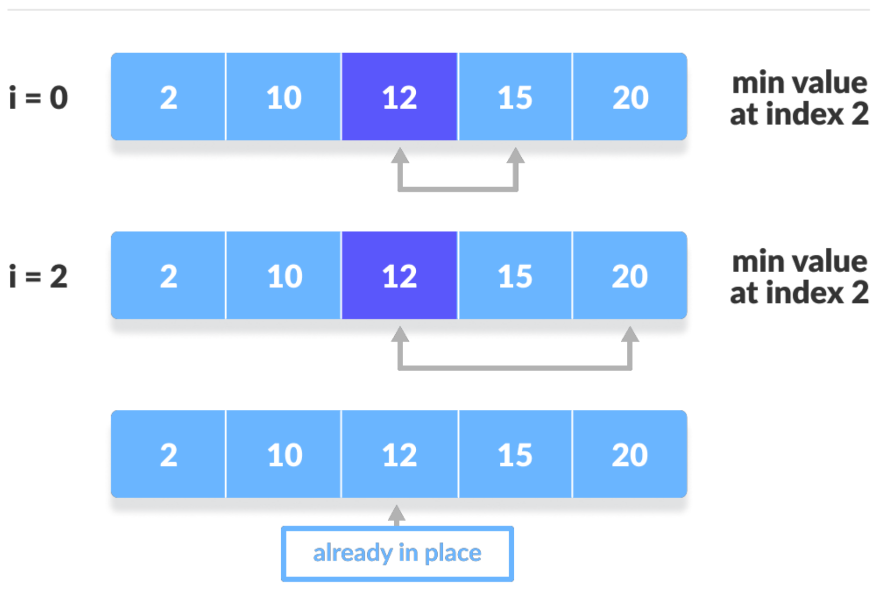
4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

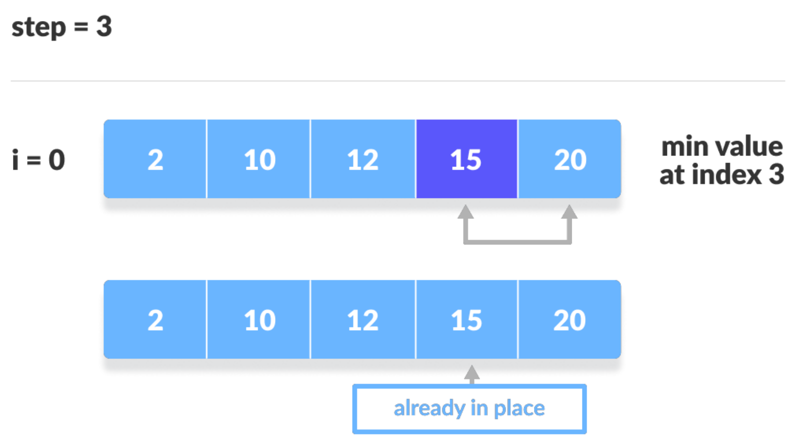

step = 0
i = 0
20
12
10
15
2
min value at index 1
i = 1
20
12
10
15
2
min value at index 2
i = 2
20
12
10
15
2
min value at index 2
i = 3
20
12
10
15
2
min value at index 4
2
12
10
15
20
swapping

step = 1

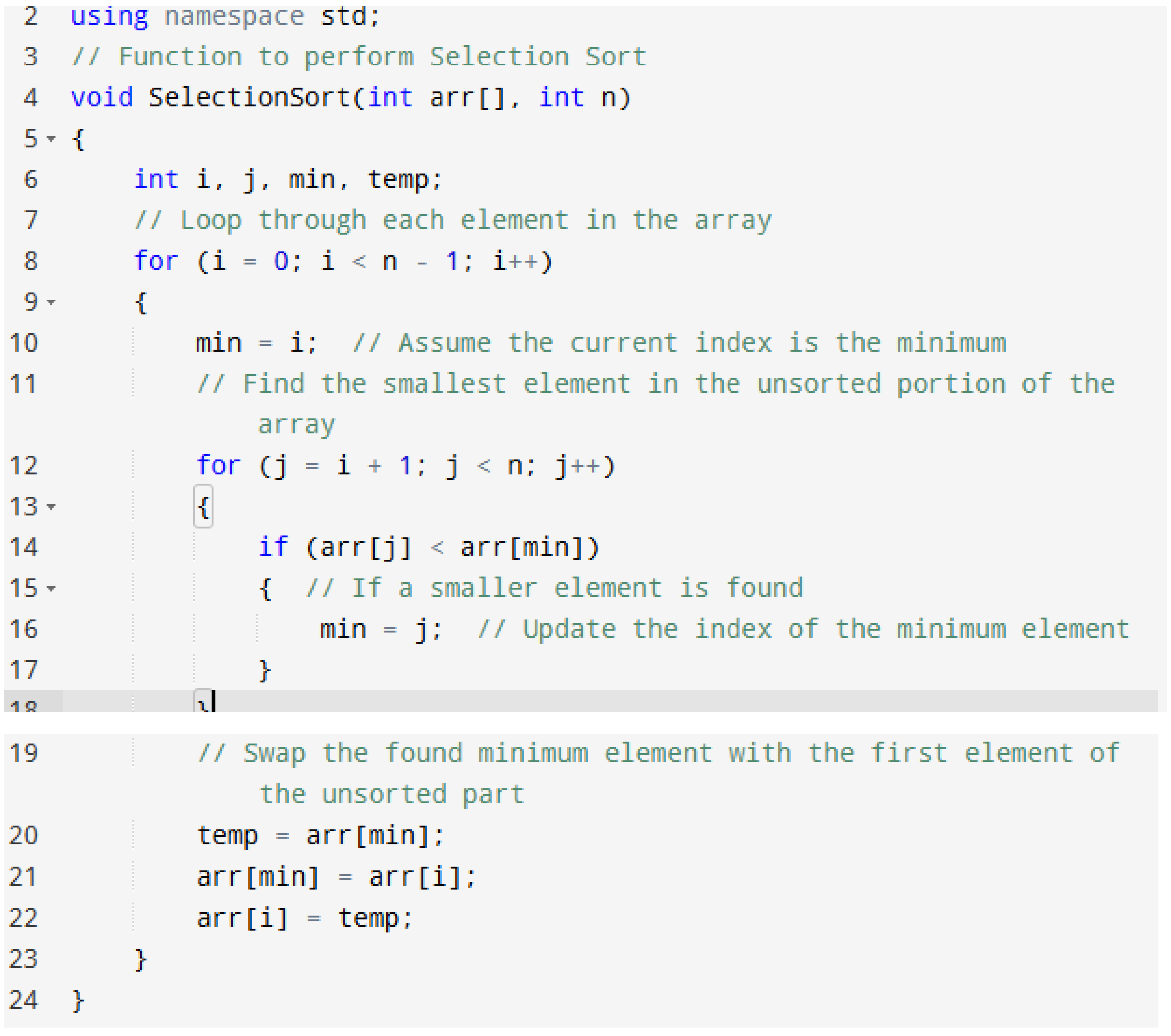

i = 0
2 12 10 15 20
min value at index 2
i = 1
2 12 10 15 20
min value at index 2
i = 2
2 12 10 15 20
min value at index 2
2 10 12 15 20
swapping

step = 2

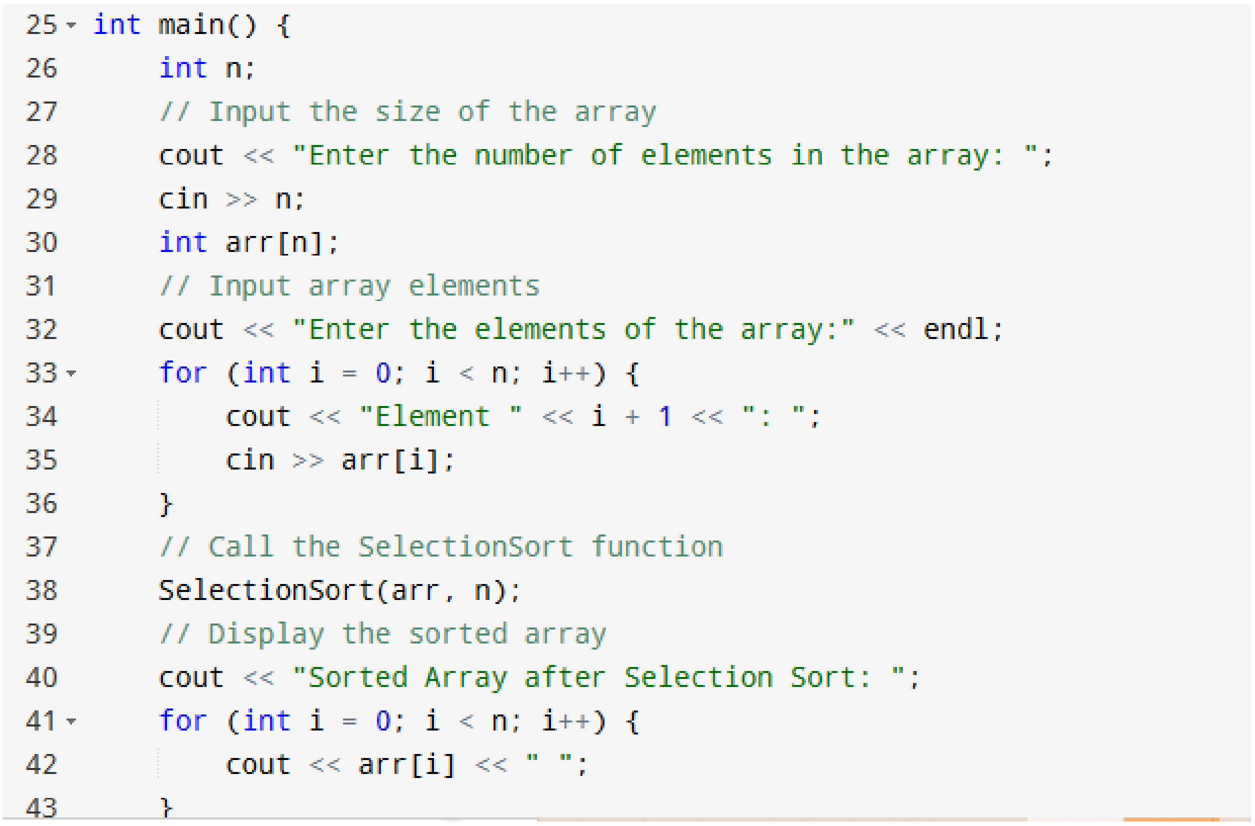

i = 0
2 10 12 15 20
min value at index 2
i = 2
2 10 12 15 20
min value at index 2
2 10 12 15 20
already in place



Section Sort Implementation

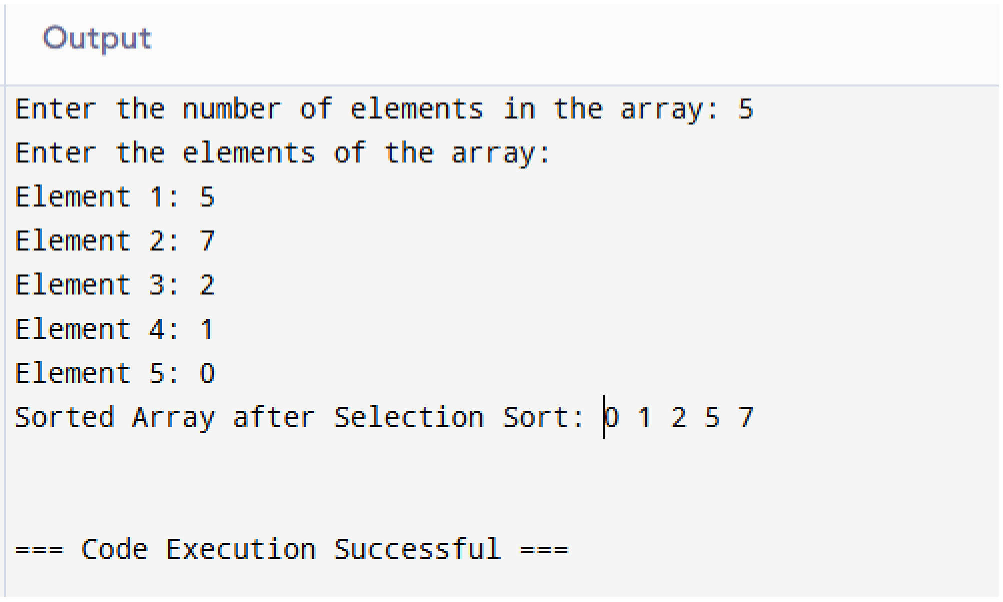


```
using namespace std;
// Function to perform Selection Sort
void SelectionSort(int arr[], int n)
{
    int i, j, min, temp;
    // Loop through each element in the array
    for (i = 0; i < n - 1; i++)
    {
        min = i;  // Assume the current index is the minimum
        // Find the smallest element in the unsorted portion of the array
        for (j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[min])
            {  // If a smaller element is found
                min = j;  // Update the index of the minimum element
            }
        }
        // Swap the found minimum element with the first element of the unsorted part
        temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}
```



```cpp
int main() {
    int n;
    // Input the size of the array
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n];
    // Input array elements
    cout << "Enter the elements of the array:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Element " << i + 1 << ": ";
        cin >> arr[i];
    }
    // Call the SelectionSort function
    SelectionSort(arr, n);
    // Display the sorted array
    cout << "Sorted Array after Selection Sort: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
```

Output



```
Enter the number of elements in the array: 5
Enter the elements of the array:
Element 1: 5
Element 2: 7
Element 3: 2
Element 4: 1
Element 5: 0
Sorted Array after Selection Sort: 0 1 2 5 7


=== Code Execution Successful ===
```

Quick sort

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(n2), respectively.

Quicksort is a sorting algorithm based on the divide and conquer approach where

1. An array is divided into subarrays by selecting a pivot element (element selected from the

array).

While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.

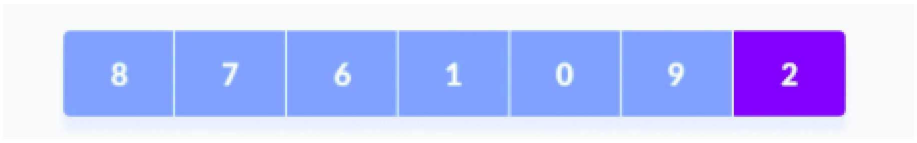
3. At this point, elements are already sorted. Finally, elements are combined to form a sorted

array.

Working of Quicksort Algorithm

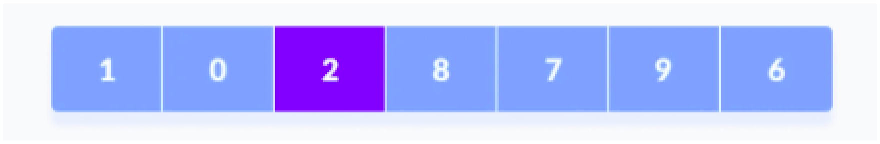
1. Select the Pivot Element

There are different variations of quicksort where the pivot element is selected from different positions. Here, we will be selecting the rightmost element of the array as the pivot element.



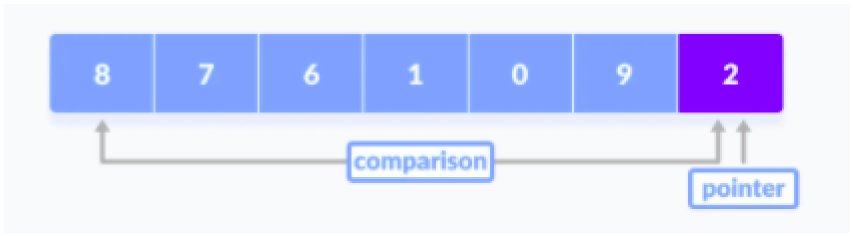
2. Rearrange the Array

Now the elements of the array are rearranged so that elements that are smaller than the pivot are put on the left and the elements greater than the pivot are put on the right.

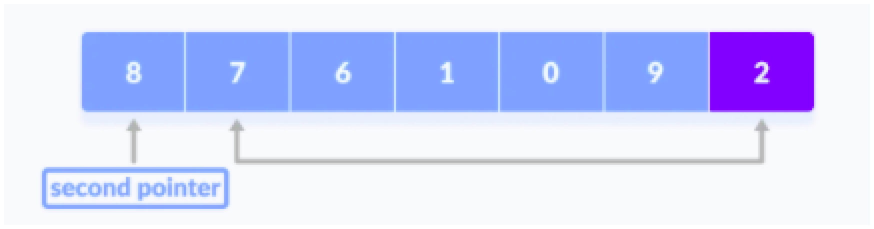


Here's how we rearrange the array

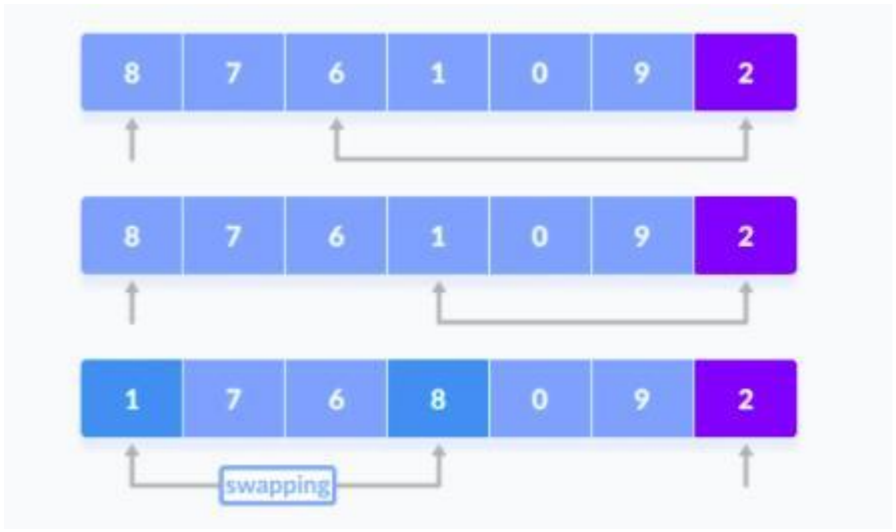
1. A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index.



2. If the element is greater than the pivot element, a second pointer is set for that element.



3. Now, pivot is compared with other elements. If an element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.

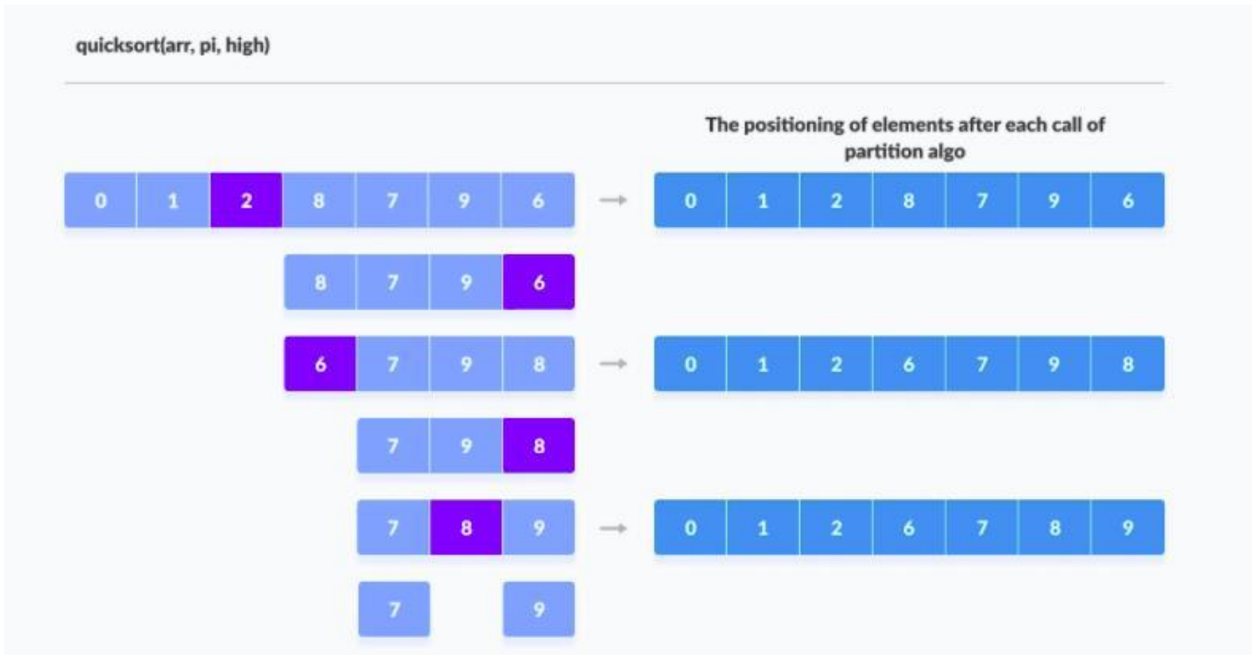


4. Again, the process is repeated to set the next greater element as the second pointer. And, swap it with another smaller element.

5. Finally, the pivot element is swapped with the second pointer.

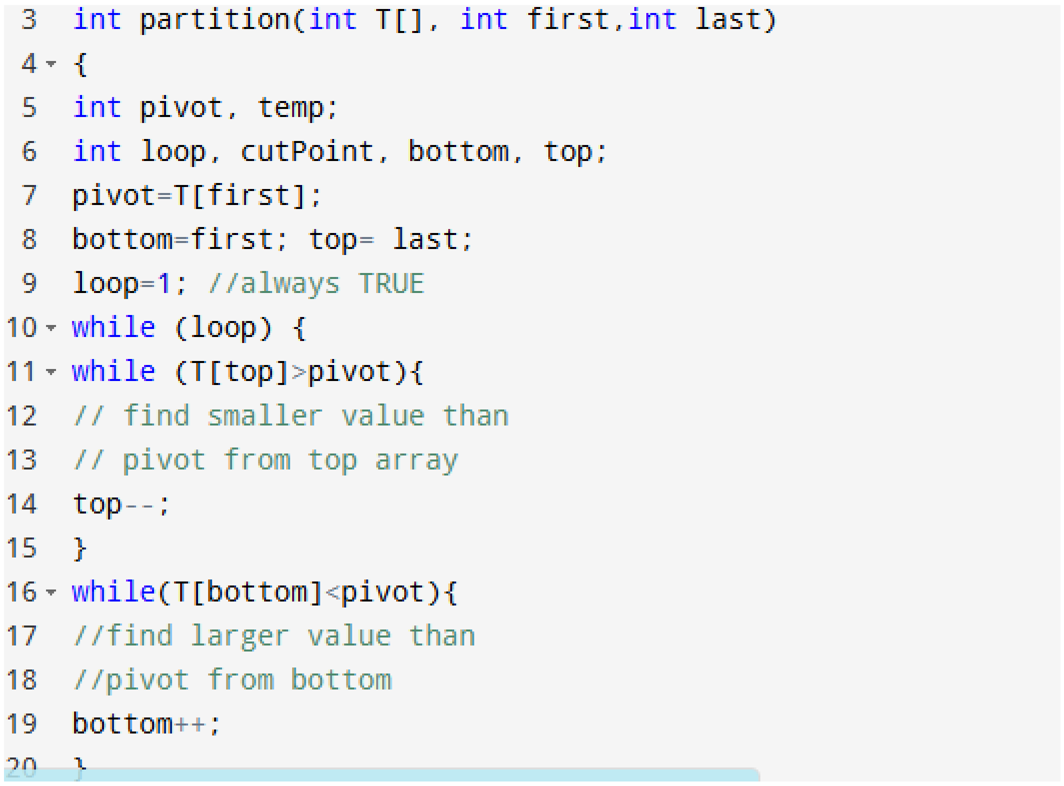
3. Divide Subarrays

Pivot elements are again chosen for the left and the right sub-parts separately. And, **step 2** is repeated.



The subarrays are divided until each subarray is formed of a single element. At this point, the array is already sorted.

Implementation



```
int partition(int T[], int first,int last)
{
int pivot, temp;
int loop, cutPoint, bottom, top;
pivot=T[first];
bottom=first; top= last;
loop=1; //always TRUE
while (loop) {
while (T[top]>pivot){
// find smaller value than
// pivot from top array
top--;
}
while(T[bottom]<pivot){
//find larger value than
//pivot from bottom
bottom++;
}
```

Quicksort Applications

Quicksort algorithm is used when

- the programming language is good for recursion
- time complexity matters
- space complexity matters

Lab Tasks.

Task# 1: Take an unsorted array of size 7. Sort the array in descending order using Quick Sort technique. Take the first element as pivot element and dry run your code to find the right position of that pivot element.

Task# 2: Take an unsorted array of size 5. Sort the array in descending order using Selection Sort technique. Dry run your code for each iteration. Implement the code in compiler and display the values of variables.