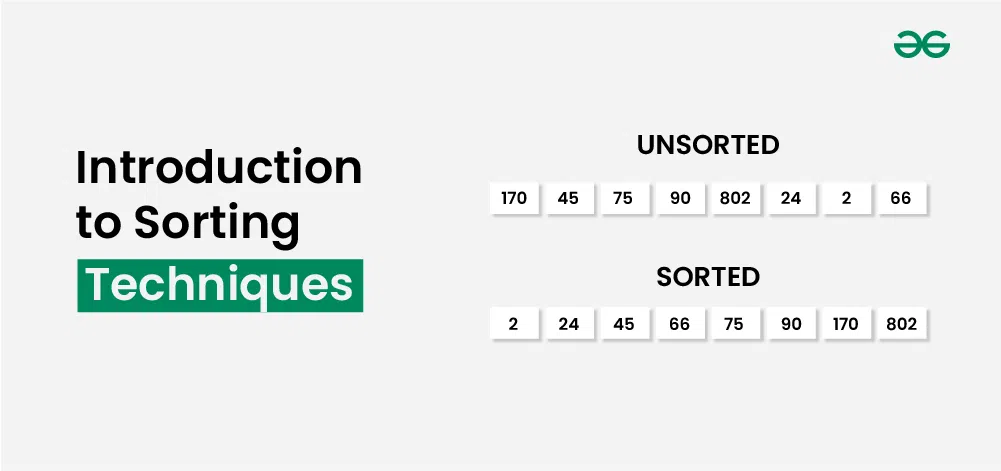
**LAB: Sorting and Recursion**

**Sorting**refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.



**Why Sorting Algorithms are Important**

The sorting algorithm is important in Computer Science because it reduces the complexity of a problem. There is a wide range of applications for these algorithms, including searching algorithms, database algorithms, divide and conquer methods, and data structure algorithms.

In the following sections, we list some important scientific applications where sorting algorithms are used

* When you have hundreds of datasets you want to print, you might want to arrange them in some way.
* Once we get the data sorted, we can get the k-th smallest and k-th largest item in O(1) time.
* Searching any element in a huge data set becomes easy. We can use Binary search method for search if we have sorted data. So, Sorting become important here.
* They can be used in software and in conceptual problems to solve more advanced problems.

[**Selection Sort**](https://www.geeksforgeeks.org/selection-sort/)**:**

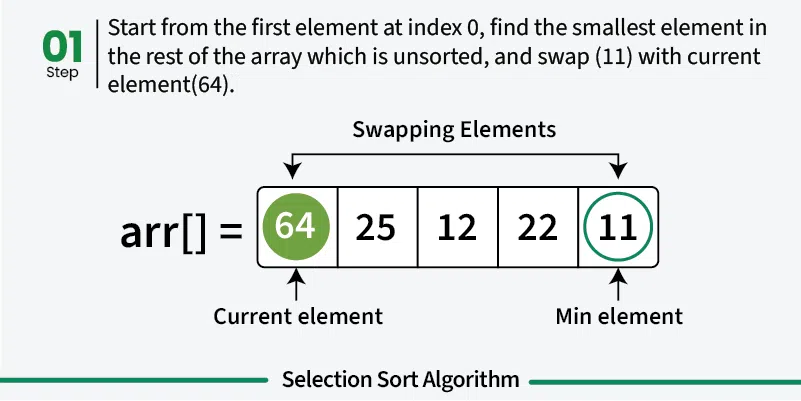
The selection sort algorithm generally is the first sorting algorithm that is taught to us. Here in every iteration of the inner loop, the smallest element is replaced with the starting element in each loop. After the end of each loop, we increment the starting position by 1 and run it till the second last element in the array. Hence, by doing so at the end of the outer loop we will be having a sorted array.

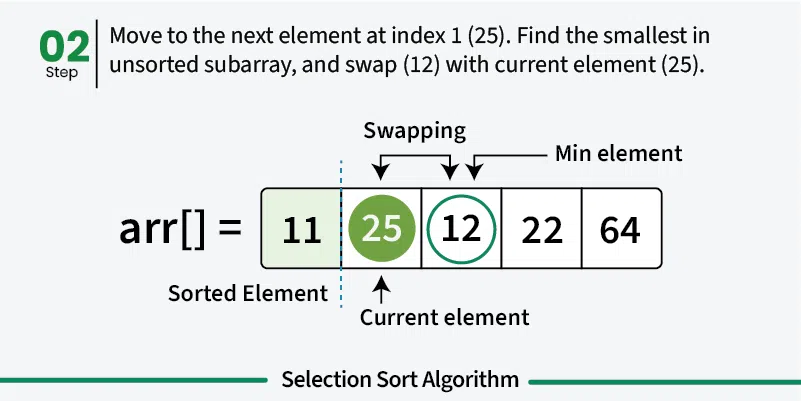
The image below explains the iteration of Selection Sort Algorithm.

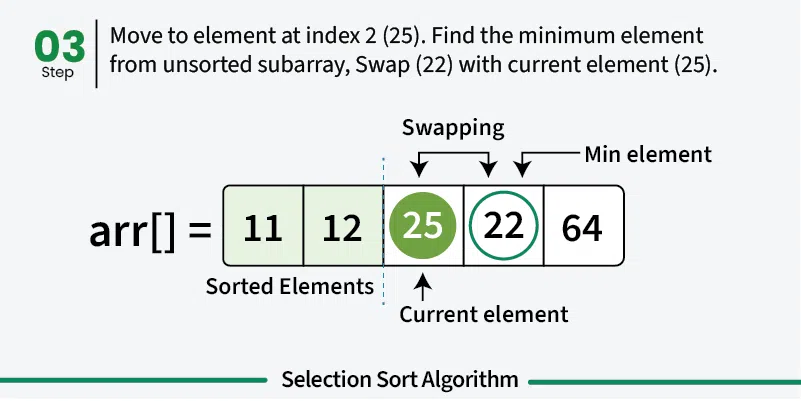


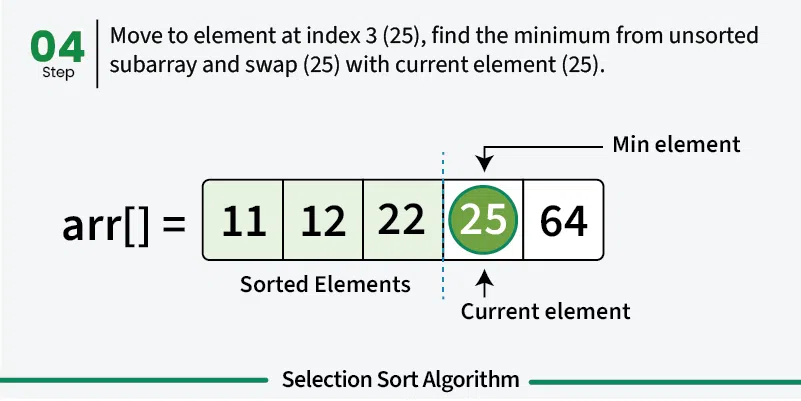
**Selection Sort** is a comparison-based sorting algorithm. It sorts an array by repeatedly selecting the **smallest (or largest)** element from the unsorted portion and swapping it with the first unsorted element. This process continues until the entire array is sorted.

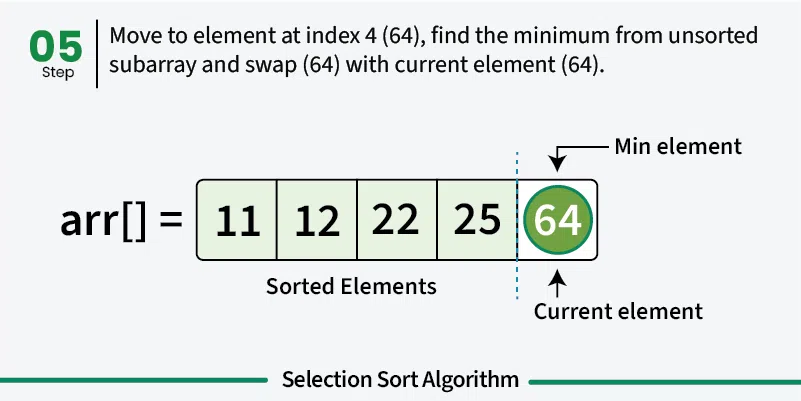
1. First we find the smallest element and swap it with the first element. This way we get the smallest element at its correct position.
2. Then we find the smallest among remaining elements (or second smallest) and swap it with the second element.
3. We keep doing this until we get all elements moved to correct position.

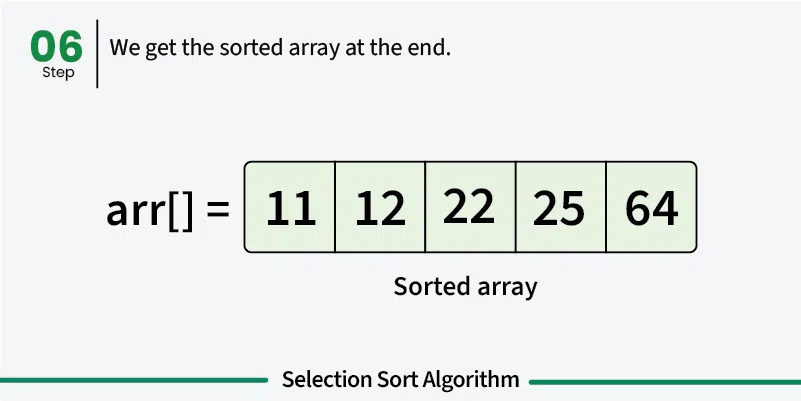










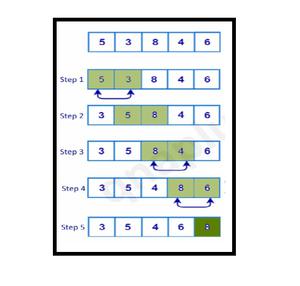


**CODE:**

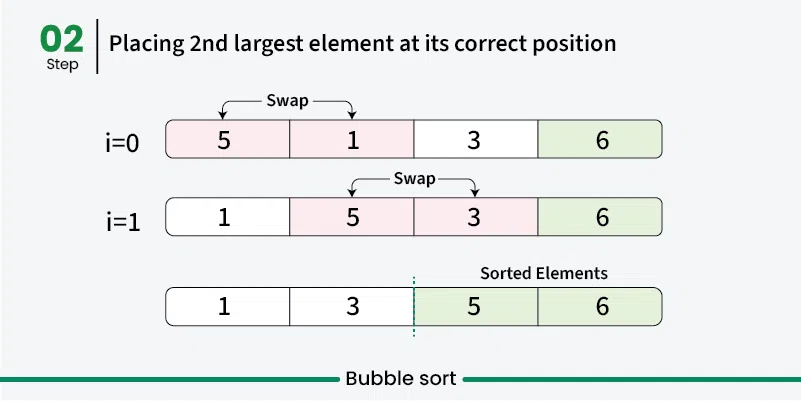
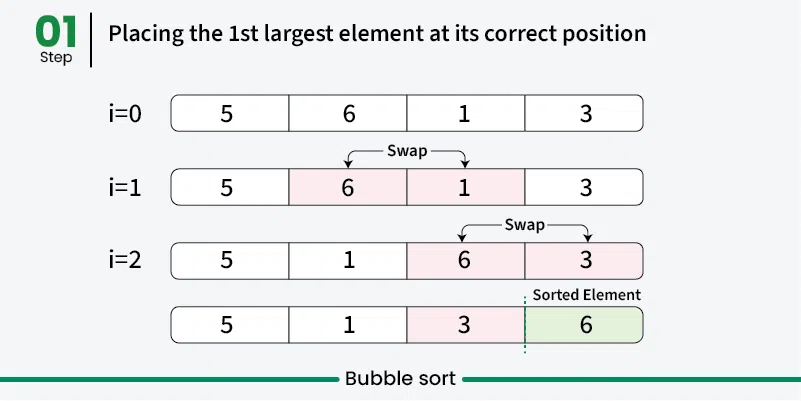
|  |
| --- |
| #include <iostream>  using namespace std;  void Selection\_Sort(int arr[], int n)  {  for(int i = 0; i < n - 1; ++i)  {  int min\_index = i;  for(int j = i + 1; j < n; ++j)  {  if(arr[j] < arr[min\_index])  min\_index = j;  }  swap(arr[i], arr[min\_index]);  }  }  int main()  {  int n = 5;  int arr[5] = {2, 0, 1, 4, 3};  Selection\_Sort(arr, n);  cout<<"The Sorted Array by using Selection Sort is : ";  for(int i = 0; i < n; ++i)  cout<<arr[i]<<" ";  return 0;  } |

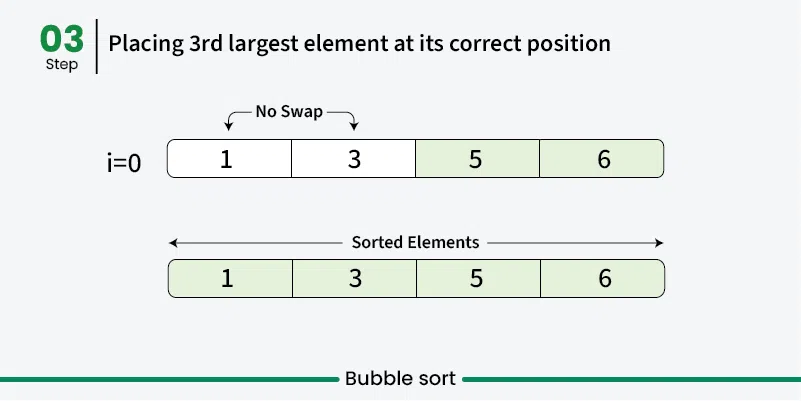
**BUBBLE SORT:**

Here swapping is carried on in two ways. In every iteration of the outer loop, the largest element is found and swapped with the last element in the loop. In the inner loop, we do pairwise swapping of two consecutive elements. In every inner loop, we go from the first element to the one less element we went in the previous loop.



* We sort the array using multiple passes. After the first pass, the maximum element goes to end (its correct position). Same way, after second pass, the second largest element goes to second last position and so on.
* In every pass, we process only those elements that have already not moved to correct position. After k passes, the largest k elements must have been moved to the last k positions.
* In a pass, we consider remaining elements and compare all adjacent and swap if larger element is before a smaller element. If we keep doing this, we get the largest (among the remaining elements) at its correct position.





CODE:

|  |
| --- |
| #include <iostream>  using namespace std;  void Bubble\_Sort(int arr[], int n)  {  for(int i = 1; i < n; ++i)  {  for(int j = 0; j <= (n - i - 1); ++j)  {  if(arr[j] > arr[j + 1])  swap(arr[j], arr[j + 1]);  }  }  }  int main()  {  int n = 5;  int arr[5] = {2, 0, 1, 4, 3};  Bubble\_Sort(arr, n);  cout<<"The Sorted Array by using Bubble Sort is : ";  for(int i = 0; i < n; ++i)  cout<<arr[i]<<" ";  return 0;  } |

**RECURSION:**

Recursion in C++ is a technique in which a function calls itself repeatedly until a given condition is satisfied. In other words, recursion is the process of solving a problem by breaking it down into smaller, simpler sub-problems.

**Syntax Structure of Recursion:**

return\_type *recursive\_func* {

*// Base Condition*

*// Recursive Case*

....  
}

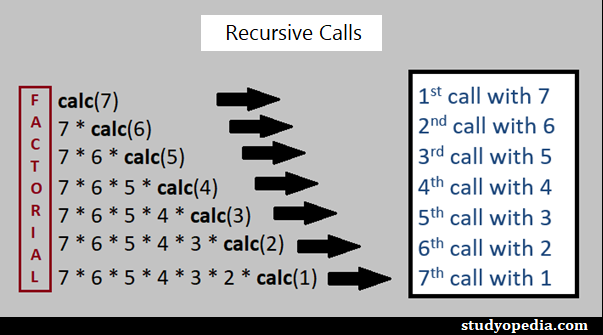
**Base Condition**

The base condition is the condition that is used to terminate the recursion. The recursive function will keep calling itself till the base condition is satisfied.

**Recursive Case**

Recursive case is the way in which the recursive call is present in the function. Recursive case can contain multiple recursive calls, or different parameters such that at the end, the base condition is satisfied, and the recursion is terminated.

|  |
| --- |
| // C++ Program to calculate the sum of first N natural numbers using recursion  #include <iostream>  using namespace std;  int nSum(int n)  {  // base condition to terminate the recursion when N = 0  if (n == 0) {  return 0;  }  // recursive case / recursive call  int res = n + nSum(n - 1);  return res;  }  int main()  {  int n = 5;  // calling the function  int sum = nSum(n);  cout << "Sum = " << sum;  return 0;  } |



LAB TASKS:

1. Write c++ code for Fibonacci Series using Recursion.
2. Write c++ code for Printing Array in reverse order using recursion.
3. Arranging Students by Roll Numbers (Bubble Sort): A school teacher wants to arrange the roll numbers of students in ascending order. The roll numbers are initially entered in random order. Write a program to sort the roll numbers using the Bubble Sort algorithm.
4. Arranging Students by Roll Numbers (Bubble Sort): A school teacher wants to arrange the roll numbers of students in ascending order. The roll numbers are initially entered in random order. Write a program to sort the roll numbers using the Bubble Sort algorithm.
5. Shortlisting Contestants by Age (Selection Sort): An event organizer is shortlisting participants based on their age in ascending order for a junior competition. Use the Selection Sort algorithm to sort the ages.
6. Ranking Exam Results (Selection Sort): A professor wants to rank students based on their exam marks in ascending order for a report. Write a program that uses the Selection Sort algorithm to sort the marks.