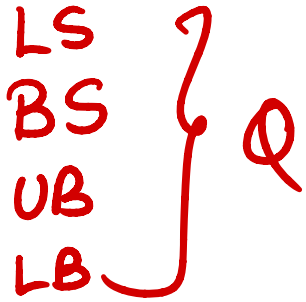
Searching & Sorting 2



- Viraj Chandra

Goal

To understand:

- What is Sorting?
- Applications in Real Life
- Types
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Counting Sort
- Inbuilt Sorting in C++ – CP

What is Sorting?

Sorting is the process of **arranging data or elements in a specific order,** typically **ascending or descending.**

Sorting can be applied to:

- Numbers (e.g., sorting test scores in ascending order).
- Strings (e.g., sorting names alphabetically).
- Objects (e.g., sorting a list of students by their grades or age).

Index

Parameter

What is Sorting? Interviews

Sorting is often used as a preprocessing step or a key part of problem-solving. Common scenarios:

- **Binary Search:** Searching for the smallest/largest element that satisfies a condition.
- **Sorting for Greedy Algorithms:** Sorting the input to process elements in a specific order. → L3, L2

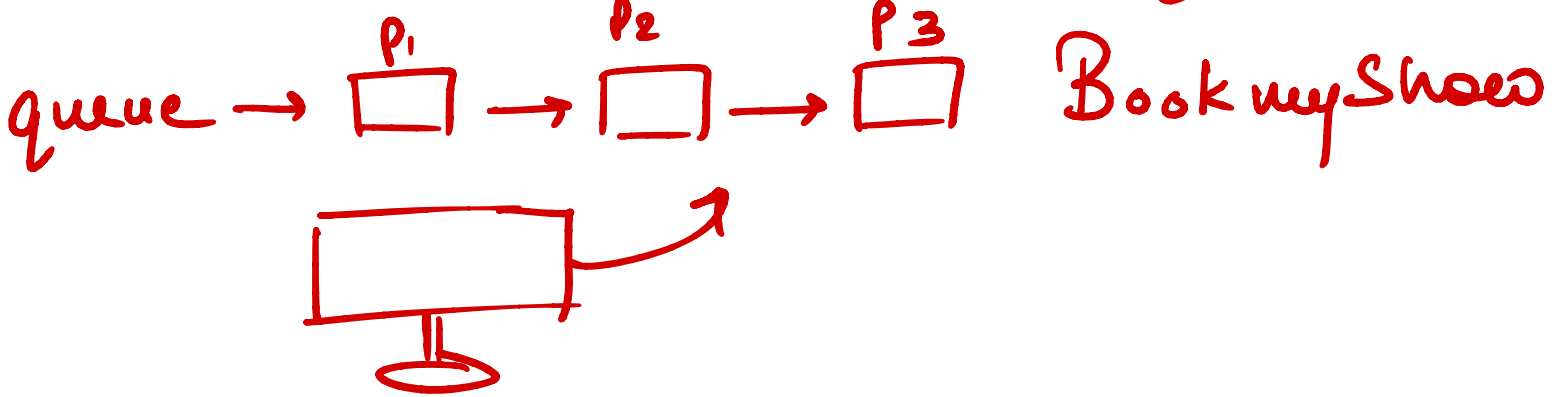
Applications in Real Life

Here are a few examples:

- ✓ TLE Leaderboard - Best Example?
- ✓ Financial Transactions
- ✓ Event Scheduling – OS

Coldplay

BookmyShow



Bubble Sort

Bubble Sort is a simple sorting algorithm where adjacent elements are repeatedly compared and swapped if they are out of order.

It **"bubbles"** the largest **(or smallest)** element to the end of the list in each iteration, much like a bubble rising to the surface.

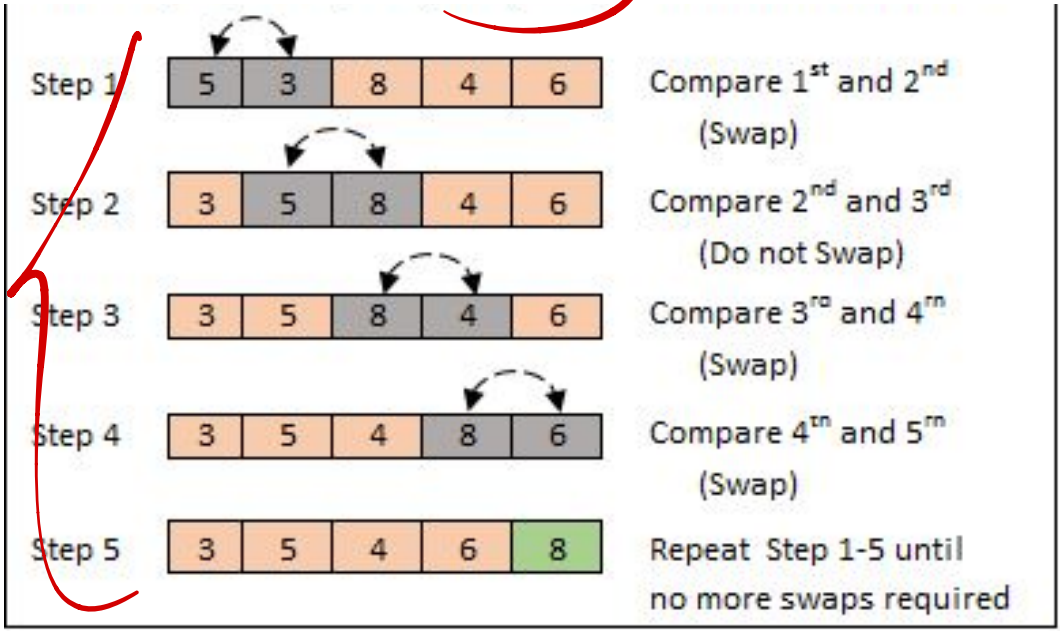
Time Complexity: Best Case (already sorted): **O(n)**
Worst Case (reversed order): **O(n^2)**
Average Case: **O(n^2)**

Space Complexity: **O(1)** (In-place sorting)

Bubble Sort

Example: [5, 3, 8, 4, 6]

One iteration of Bubble Sort



Bubble Sort - Code

```
1  void bubbleSort(int arr[], int n) {
2      for (int i = 0; i < n - 1; i++) {
3          bool swapped = false;  // Optimization to break if already sorted
4          for (int j = 0; j < n - i - 1; j++) {
5              if (arr[j] > arr[j + 1]) {
6                  swap(arr[j], arr[j + 1]);
7                  swapped = true;
8              }
9          }
10         if (!swapped) break;  // Exit early if no swaps
11     }
12 }
```

Selection Sort

Selection Sort divides the array into two parts: the sorted part and the unsorted part.

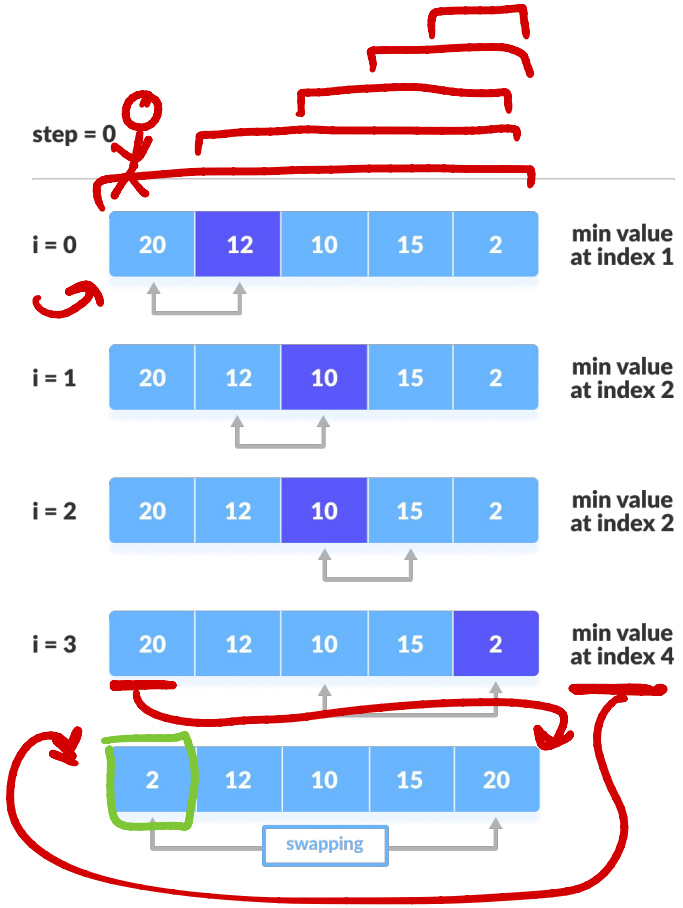
It **repeatedly selects the smallest** (or largest, for descending order) element from the unsorted part and **places it at the beginning** of the sorted part.

Time Complexity: Best Case (already sorted): **$O(n^2)$**
Worst Case (reversed order): **$O(n^2)$**
Average Case: **$O(n^2)$**
Space Complexity: O(1) (In-place sorting)

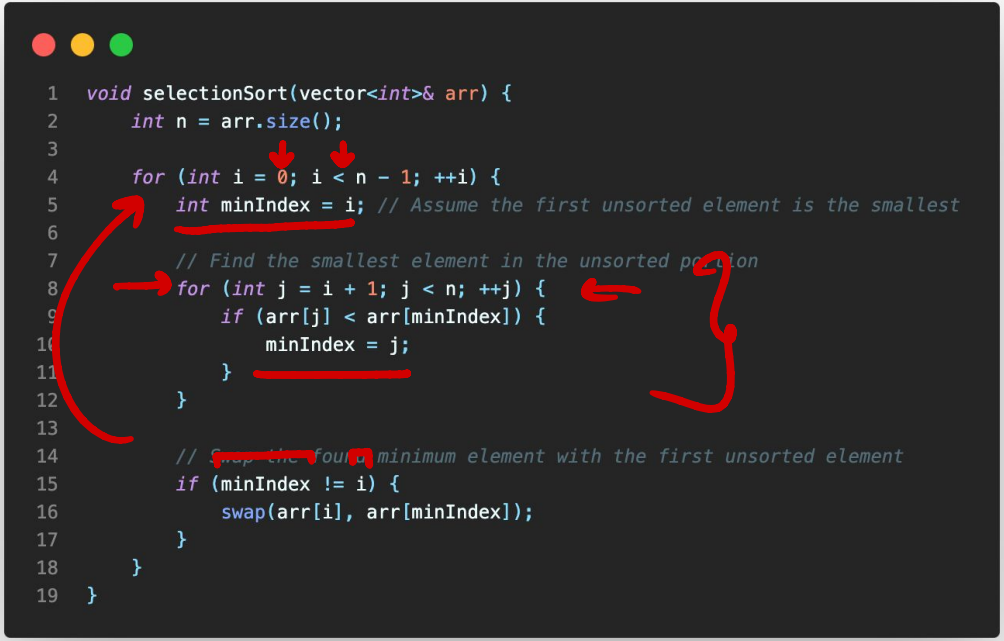
Selection Sort

Example: [20, 12, 10, 15, 2]

One iteration of selection sort



Selection Sort - Code



```cpp
void selectionSort(vector<int>& arr) {
    int n = arr.size();

    for (int i = 0; i < n - 1; ++i) {
        int minIndex = i; // Assume the first unsorted element is the smallest

        // Find the smallest element in the unsorted portion
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        // Swap the found minimum element with the first unsorted element
        if (minIndex != i) {
            swap(arr[i], arr[minIndex]);
        }
    }
}
```

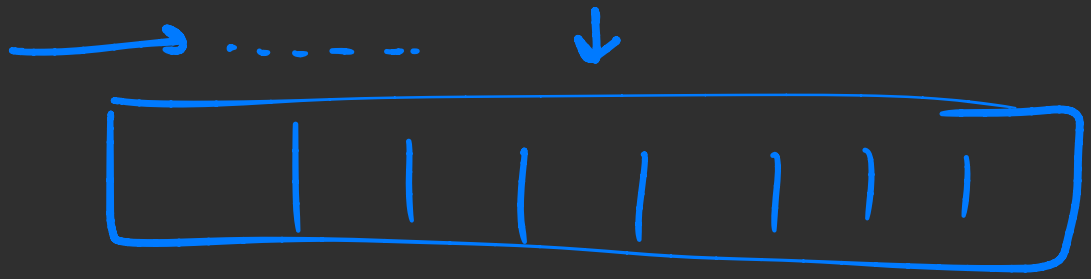
Insertion Sort

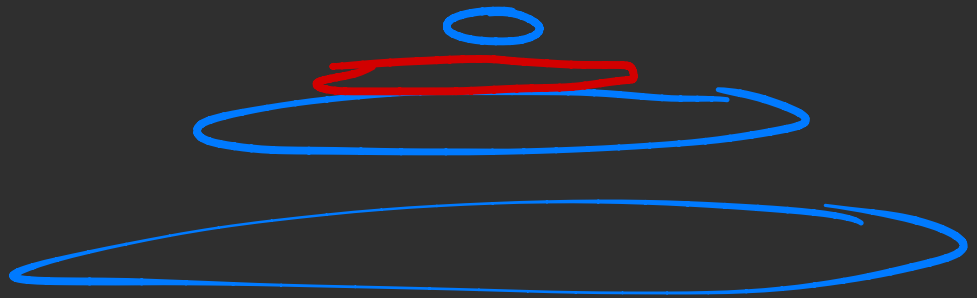
Insertion Sort is a simple and intuitive sorting algorithm that works like **sorting playing cards in your hand.**

It builds the **final sorted array one element at a time** by inserting each element into its correct position in the sorted part of the array.

Time Complexity: Best Case (nearly sorted): **O(n)**
Worst Case (reversed order): **O(n^2)**
Average Case: **O(n^2)**

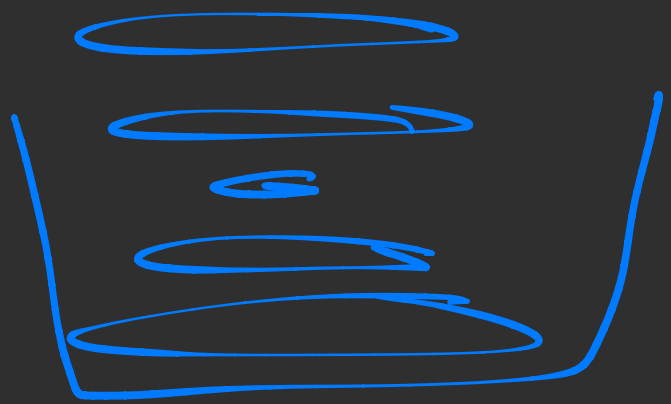
Space Complexity: O(1)





Mom

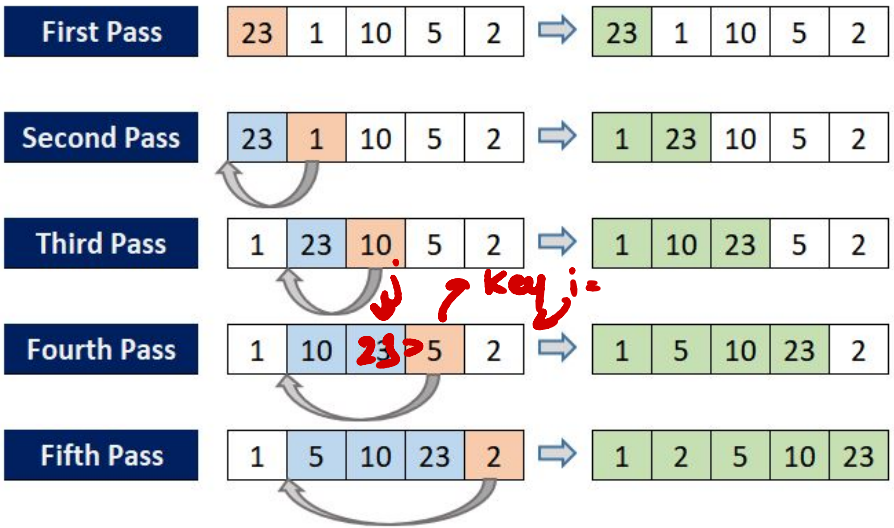
dishes →



wash

Insertion Sort

Example: [23, 1, 10, 5, 2]



Insertion Sort - Code

```cpp
void insertionSort(vector<int>& arr) {
    int n = arr.size();

    for (int i = 1; i < n; ++i) {
        int key = arr[i]; // Current element to be inserted
        int j = i - 1;

        // Shift elements of the sorted portion to the right
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }

        // Insert the key element into the correct position
        arr[j + 1] = key;
    }
}
```

0	1	2	3	4
1	5	10	23	2

1 > 5

i = 3

key = 5

j = ~~2~~ ~~1~~ 0

while (———)

arr[j+1] = arr[j] ;

j-- ;

Counting Sort

Counting Sort is a non-comparison-based sorting algorithm that sorts integers by counting the occurrences of each unique element in the input array.

It works efficiently for arrays where elements are within a small range and non-negative.

Unlike other sorting algorithms, it relies on the frequency of elements rather than comparisons.

Counting Sort

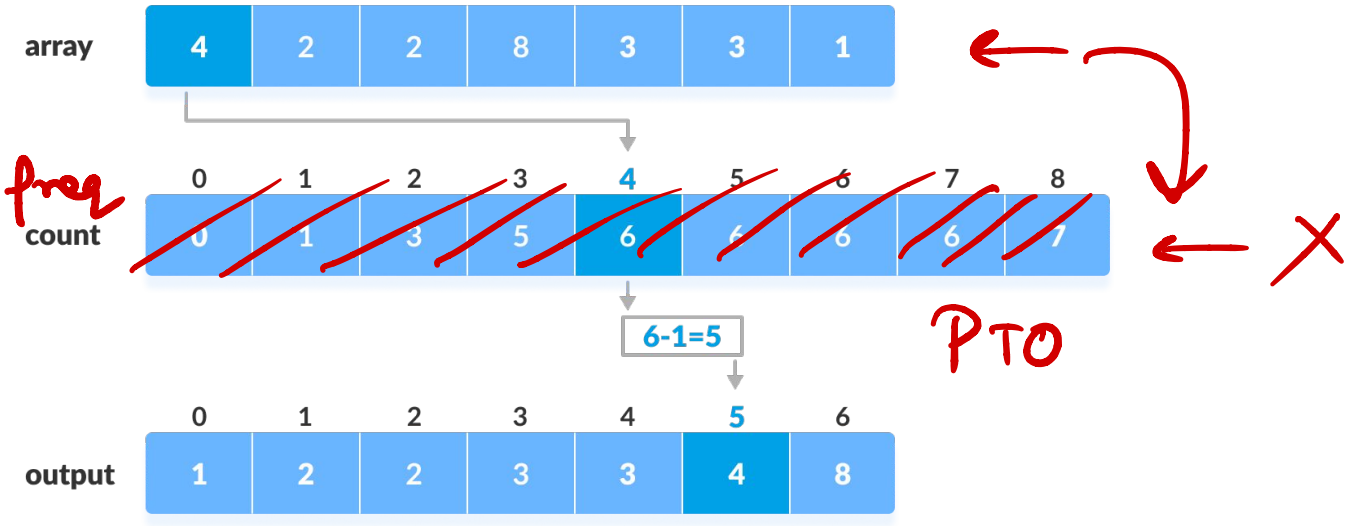
Time Complexity: Best Case : **O(n + k)**
Worst Case : **O(n + k)**
Average Case: **O(n + k)**

Space Complexity: O(k)

Where **'n'** is the number of elements in the array, and **'k'** is the range of the input elements (maximum value - minimum value).

Counting Sort

Example: [4, 2, 2, 8, 3, 3, 1]



Counting Sort - Code

```cpp
void countingSort(vector<int>& arr, int maxValue) {
    // Create and initialize the count array
    vector<int> count(maxValue + 1, 0);

    // Count the occurrences of each element
    for (int num : arr) {
        count[num]++;
    }

    // Overwrite the input array with sorted values
    int index = 0;
    for (int i = 0; i <= maxValue; ++i) {
        while (count[i] > 0) {
            arr[index++] = i;
            count[i]--;
        }
    }
}
```

Inbuilt Sorting in C++ (CP)

sort() is powerful and optimized for CP. We use it for fast and efficient sorting in contests.

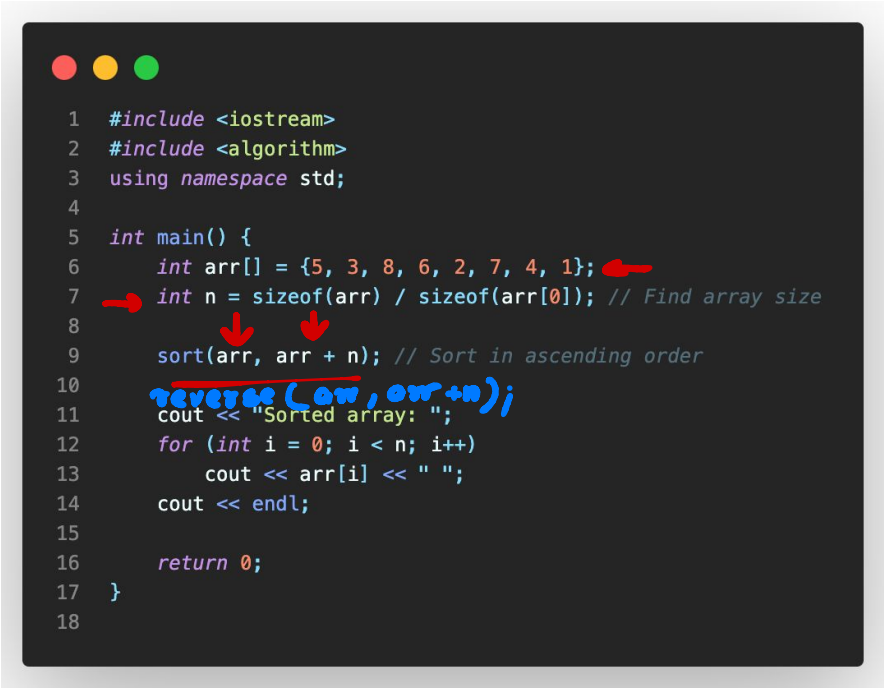
It uses a hybrid sorting algorithm that combines **QuickSort, HeapSort, and Insertion Sort** for optimal performance.

Time Complexity: Best Case (already sorted) : **O(n)**
Worst Case : **O(nlogn)**
Average Case: **O(nlogn)**

Space Complexity: O(n)

CP

Inbuilt Sorting in C++



```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int arr[] = {5, 3, 8, 6, 2, 7, 4, 1};
    int n = sizeof(arr) / sizeof(arr[0]); // Find array size

    sort(arr, arr + n); // Sort in ascending order
    reverse(low, low+n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Important Links

- Bubble Sort - https://www.geeksforgeeks.org/bubble-sort/
- Merge Sort - https://www.geeksforgeeks.org/merge-sort/
- Insertion Sort - https://www.geeksforgeeks.org/insertion-sort/
- Selection Sort - https://www.geeksforgeeks.org/selection-sort/
- Counting Sort - https://www.geeksforgeeks.org/counting-sort/
- Radix Sort - https://www.geeksforgeeks.org/radix-sort/
- Quick Sort - https://www.geeksforgeeks.org/quick-sort/
- Heap Sort - https://www.geeksforgeeks.org/heap-sort/