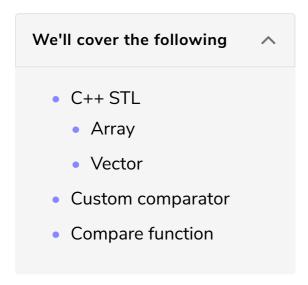
STL

In this lesson, we'll learn about the C++ STL sorting library and how to leverage it to perform custom sorting.



C++ STL

Here's what we will be doing when we have to sort an array or vector.

Array

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int N = 8;
    int arr[N] = {5, 3, 6, 4, 8, 1, 7, 2};

    sort(arr, arr + N);
    for(int i = 0; i < N; i++)
        cout << arr[i] << " ";

    return 0;
}</pre>
```

Vector

```
#include <bits/stdc++.h>
using namespace std;
int main() {
```

```
vector<int> vect{5, 3, 6, 4, 8, 1, 7, 2};
sort(vect.begin(), vect.end());

for(int i = 0; i < vect.size(); i++)
    cout << vect[i] << " ";

return 0;
}</pre>
```







[]

The actual sorting algorithm used will depend on the language but these are generally faster sorting algorithms of the order O(NlogN).

Custom comparator

The sort function sorts the integers in non-decreasing order. For other data types, default comparison is used. For example:

- float same as int.
- pair<first, second> the first part of the pair is compared first. If they are the same, then the second part is compared.

A lot of times, we want to define our own comparator for int or for a custom struct.

Compare function

To perform the comparison, we need to define a function that takes in two variables, (a, b), of the same type and returns an integer as below:

- ullet true or non-zero value: if a should appear **before** b
- false or zero: if a should appear **after** b

Based on the information above, the compare function for default behavior of sort would look like this:

```
int compare(int a, int b) {
   if (a < b) return 1;
   return 0;
}</pre>
```

or simply:

```
int compare(int a, int b) { // non-decreasing
    return a < b;
}</pre>
```

To sort in non-increasing order, just use the opposite comparison.

```
int compare(int a, int b) { // non-increasing
    return a > b;
}
```

Question: Given a list of integers, rearrange them so that all odd numbers appear before even numbers. Additionally, odd numbers are in non-decreasing order and even numbers are in non-increasing order.

Solution: We sort based on parity (the result of %2).

- If parity is different (one even, one odd), we want the odd number before the even number. So, we return true is a is odd.
- If parity is the same (both even or both odd), default sorting order for odd (a
 (b) and opposite for even (a > b).

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int compare(int a, int b) {
  int r1 = a\%2;
  int r2 = b\%2;
  if (r1 == 0 && r2 == 0)
   return a > b;
  if (r1 == 1 && r2 == 1)
    return a < b;
  if (r1 == 0) // r2 == 1
    return 0;
  else // r1 == 1 & r2 == 0
    return 1;
int main() {
  vector<int> v = {1,2,3,4,5,6,7,8,9};
  sort(v.begin(), v.end(), compare);
  for (auto it : v) cout << it << " ";
  return 0;
```







ב

In the next chapter, we'll start linked lists.