# Some Tips for Calling C++ Sorts from C

Applied Algorithms, Fall 2024

As discussed in class, the standard method of sorting in C is to use `qsort`. Unfortunately, this function is not very fast for a few reasons.

There are several ways to deal with this; for example, a handwritten sort implementation is often going to be competitive with `qsort`. This handout focuses on another method: calling `C++` code from your `C` code.

Using the simple `std::sort` method from the `C++` standard library, I was able to decrease the time sort took in my code by almost 50%, even on an array of `structs`. Using primitive types and/or calling one of the Boost library methods may be even faster for some use cases.

## Tips for Calling `std::sort`

### Making a Wrapper Function

It is almost certainly useful to create a file with a *wrapper function*—a simple function that will call the `C++` code for you. This is helpful because we can build only this file with `g++` (the `C++` variant of `gcc`), allowing us to build the rest of our files with `gcc`.

Your file should probably look something like the following excerpt. I called mine `sort.cpp`:

```
1  #include <algorithm>
2  #include "struct_file.h"
3
4   bool int_struct_Comp(Int_Struct s1, Int_Struct s2) {
5           return s1.cost < s2.cost;
6   }
7
8   extern "C" void cpp_sort_int(int* A, int size){
9           std::sort(A, A + size);
10  }
11
12  extern "C" void cpp_sort_sumSolutions(Int_Struct* A, int size){
13      std::sort(A, A + size, int_struct_Comp);
14  }
```

Let's go through this file line by line.

Line 1: the `algorithm` library is necessary to use `std::sort`.

Line 2: replace this line with an include of the file where you defined your `struct` (here, as an example, we have a `struct` called `Int_Struct` which contains a single `int`; this definition is stored in `struct_file.h`). If you're not sorting structs, just remove this line.

Line 4: comparison function for the objects you're sorting. Note: for `std::sort`, *you do not need to do this with primitive types.* `C++` knows how to sort `int`s, `float`s, etc. You only need to include this if you're sorting something like `struct`s.

Line 8: this is an example of a wrapper function that will sort integers. (Note that it does not pass the comparison function to `std::sort`.) It can be easily modified to sort doubles, etc. Note that `size` is the number of entries in `A`—unlike in `C`, you do not need to give the size of each entry.

Line 12: this is an example of a wrapper function that uses an arbitrary comparison function (from above) to sort an array. This is provided as the third argument to the function.