AlgorithmsLibrary

January 18, 2026

1 Algorithms Library

1.1 External Resources

- YouTube Video - https://youtu.be/yldnuWfZcW4
- YouTube Podcast - https://youtu.be/_bM3KR1Ms14
- C++ Reference for Algorithms: https://en.cppreference.com/w/cpp/algorithm
- NotebookLM learning materials - https://notebooklm.google.com/notebook/d2e20c24-dd73-4264-bdde-88b9cf6d343e

1.2 Overview

- The standard library provides implementations of various algorithms in C++.
- Some commonly used algorithms include sorting, searching, and manipulating sequences of elements.

1.3 Common functions defined in `<algorithm>`

- `std::sort`: Sorts a range of elements.
- `std::is_sorted`: Checks if a range is sorted.
- `std::find`: Searches for a specific value in a range.
- `std::copy`: Copies elements from one range to another.
- `std::accumulate`: Computes the sum of a range of elements.
- `std::transform`: Applies a function to a range of elements and stores the result in another range.

1.4 Example Usage

1.5 Numberic Operations

[1]:
```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;
```

[2]:
```
vector<float> data{1.5, 2.5, 3.5};
float sum = accumulate(data.begin(), data.end(), 0.0f);
```

```
cout << "Sum: " << sum << endl;
```

```
Sum: 7.5
```

[3]:
```
int a = 5;
int b = 10;
swap(a, b);
cout << "a: " << a << ", b: " << b << endl;
```

```
a: 10, b: 5
```

[58]:
```
cout << max(10, 20) << endl;
cout << min({10, 20, 30, -1, -100}) << endl;
```

```
20
-100
```

[62]:
```
auto compare = [](const string& a, const string& b) {
    return a.size() < b.size(); // Max-heap based on the size of the first␣
 ↪element
};
// create a initializer list of words
auto words = {"apple", "banana", "cherry", "date", "cat", "dog", "elephant"};
cout << max(words, compare) << endl;
cout << min(words, compare) << endl;
```

```
elephant
cat
```

1.5.1 Sorting Operations

[ ]:
```
vector<int> v{5, 3, 8, 1, 2};

// Sort the vector in ascending order
sort(v.begin(), v.end());
```

[4]:
```
v
```

[4]:
```
{ 1, 2, 3, 5, 8 }
```

[5]:
```
is_sorted(v.begin(), v.end()) ? cout << "Sorted\n" : cout << "Not sorted\n";
```

```
Sorted
```

1.5.2 Binary Search Operations

- `std::binary_search`: Checks if a value exists in a sorted range.
- `std::lower_bound`: Returns an iterator to the first element that is not less than a given value.
- `std::upper_bound`: Returns an iterator to the first element that is greater than a given value.

- `std::equal_range`: Returns a pair of iterators to the range of elements equal to a given value.

[6]:
```
vector<string> words{"apple", "orange", "banana", "grape"};
sort(words.begin(), words.end());
```

[7]:
```
auto it = binary_search(words.begin(), words.end(), "banana");
if (it) {
    cout << "\"banana\" found in the list.\n";
} else {
    cout << "\"banana\" not found in the list.\n";
}
```

```
"banana" found in the list.
```

[12]:
```
vector<int> nums{10, 20, 30, 40, 50};
auto it = lower_bound(nums.begin(), nums.end(), 25);
cout << "First element not less than 25 is: " << *it << endl;
```

```
First element not less than 25 is: 30
```

[14]:
```
auto it_upper = upper_bound(nums.begin(), nums.end(), 25);
cout << "First element greater than 25 is: " << *it_upper << endl;
```

```
First element greater than 25 is: 30
```

1.6 Set Operations

- `std::set_union`: Computes the union of two sorted ranges.
- `std::set_intersection`: Computes the intersection of two sorted ranges.
- `std::set_difference`: Computes the difference between two sorted ranges.
- `std::includes`: Checks if one sorted range includes another.
- examples follow

[17]:
```
vector<int> nums{1, 2, 3, 4, 5};
vector<int> other{1, 2, 3, 4, 5, 6, 7, 8};
```

[20]:
```
bool includes_result = includes(other.begin(), other.end(), nums.begin(), nums.
 ↪end());
cout << "other includes nums? " << (includes_result ? "Yes" : "No") << endl;
```

```
other includes nums? Yes
```

[21]:
```
vector<int> intersection;
set_intersection(nums.begin(), nums.end(),
                 other.begin(), other.end(),
                 back_inserter(intersection));
```

[22]:
```
intersection
```

[22]:
```
{ 1, 2, 3, 4, 5 }
```

[23]:
```
vector<int> union_result;
set_union(nums.begin(), nums.end(),
          other.begin(), other.end(),
          back_inserter(union_result));
```

[24]:
```
union_result
```

[24]:
```
{ 1, 2, 3, 4, 5, 6, 7, 8 }
```

1.7 Heap Operations

- heap is a specialized tree-based data structure that satisfies the heap property.
- heap is commonly used to implement priority queues.
- C++ provides max-heap operations in the `<algorithm>` header
- `std::make_heap`: Converts a range into a heap.
- `std::push_heap`: Adds a new element to the heap.
- `std::pop_heap`: Removes the largest element from the heap.
- `std::sort_heap`: Sorts the elements in the heap in ascending order.
- `std::is_heap`: Checks if a range is a valid heap.
- examples follow

[37]:
```
vector<pair<int, string>> items = {
    {3, "read"},
    {1, "write"},
    {4, "code"},
    {2, "play"}
};
```

[38]:
```
items
```

[38]:
```
{ {3 , "read"}, {1 , "write"}, {4 , "code"}, {2 , "play"} }
```

[39]:
```
make_heap(items.begin(), items.end(),
          [](const auto& a, const auto& b) {
              return a.first < b.first; // Max-heap based on the first integer␣
 ↪value
          });
```

[40]:
```
items
```

[40]: { {4 , "code"}, {2 , "play"}, {3 , "read"}, {1 , "write"} }

[41]:
```
// first push a new item
items.push_back({5, "sleep"});
```

[42]:
```
items
```

[42]: { {4 , "code"}, {2 , "play"}, {3 , "read"}, {1 , "write"}, {5 , "sleep"} }

[43]:
```
// then re-adjust the heap
push_heap(items.begin(), items.end(),
               [](const auto& a, const auto& b) {
                   return a.first < b.first;
               });
```

[44]:
```
items
```

[44]: { {5 , "sleep"}, {4 , "code"}, {3 , "read"}, {1 , "write"}, {2 , "play"} }

[46]:
```
is_heap(items.begin(), items.end(),
         [](const auto& a, const auto& b) {
             return a.first < b.first;
         }) ? cout << "items is a heap\n" : cout << "items is not a heap\n";
```

```
items is a heap
```

[47]:
```
while (!items.empty()) {
    // Move the largest element to the end
    pop_heap(items.begin(), items.end(),
             [](const auto& a, const auto& b) {
                 return a.first < b.first;
             });
    cout << "Process largest item: {" << items.back().first << ", " << items.
↪back().second << "}" << endl;
    // Remove the last element (the largest)
    items.pop_back();
    cout << "After popping, items size: " << items.size() << endl;
}
```

```
Process largest item: {5, sleep}
After popping, items size: 4
Process largest item: {4, code}
After popping, items size: 3
Process largest item: {3, read}
After popping, items size: 2
Process largest item: {2, play}
After popping, items size: 1
Process largest item: {1, write}
After popping, items size: 0
```

[ ]:

```

```

1.8 Permutation Operations

- `std::next_permutation`: Generates the next lexicographical permutation of a range.
- `std::prev_permutation`: Generates the previous lexicographical permutation of a range.
- `std::is_permutation`: Checks if two ranges are permutations of each other.
- examples follow

[ ]:

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

[4]:

```
string dna = "AGT";
string dna1 = "TAG";
if (is_permutation(dna.begin(), dna.end(), dna1.begin())) {
    cout << "dna and dna1 are permutations of each other." << endl;
} else {
    cout << "dna and dna1 are not permutations of each other." << endl;
}
```

```
dna and dna1 are permutations of each other.
```

[6]:

```
next_permutation(dna.begin(), dna.end());
cout << dna << endl;
```

```
AGT
```

[7]:

```
while (next_permutation(dna.begin(), dna.end())) {
    cout << dna << endl;
}
```

```
ATG
GAT
GTA
TAG
TGA
```

1.9 Other operations

- `std::transform`: Applies a function to a range of elements and stores the result in another range.
- `std::remove_if`: Removes elements from a range that satisfy a given condition.

[ ]:

```
char to_uppercase(unsigned char& c)
{
    c = std::toupper(c);
}
```

[9]:
```
string hello("hello");
transform(hello.begin(), hello.end(), hello.begin(), to_uppercase);
cout << hello << endl;
```

```
HELLO
```

[10]:
```
bool is_vowel(char c)
{
    c = std::tolower(c);
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
string text = "Hello, World!";
text.erase(remove_if(text.begin(), text.end(), is_vowel), text.end());
cout << text << endl;
```

```
Hll, Wrld!
```

[11]:
```
std::vector<int> v {1, 2, 3};
std::reverse(v.begin(), v.end());
```

[12]:
```
v
```

[12]:
```
{ 3, 2, 1 }
```

[ ]:
```
int a[] = {10, 4, 5, 6, 7};
// array doesn't have member functions for begin() and end()
std::reverse(std::begin(a), std::end(a));
```

[17]:
```
a
```

[17]:
```
{ 7, 6, 5, 4, 10 }
```

1.10 Exercises

- But I want to Win - https://open.kattis.com/problems/butiwanttowin
- CD - https://open.kattis.com/problems/cd