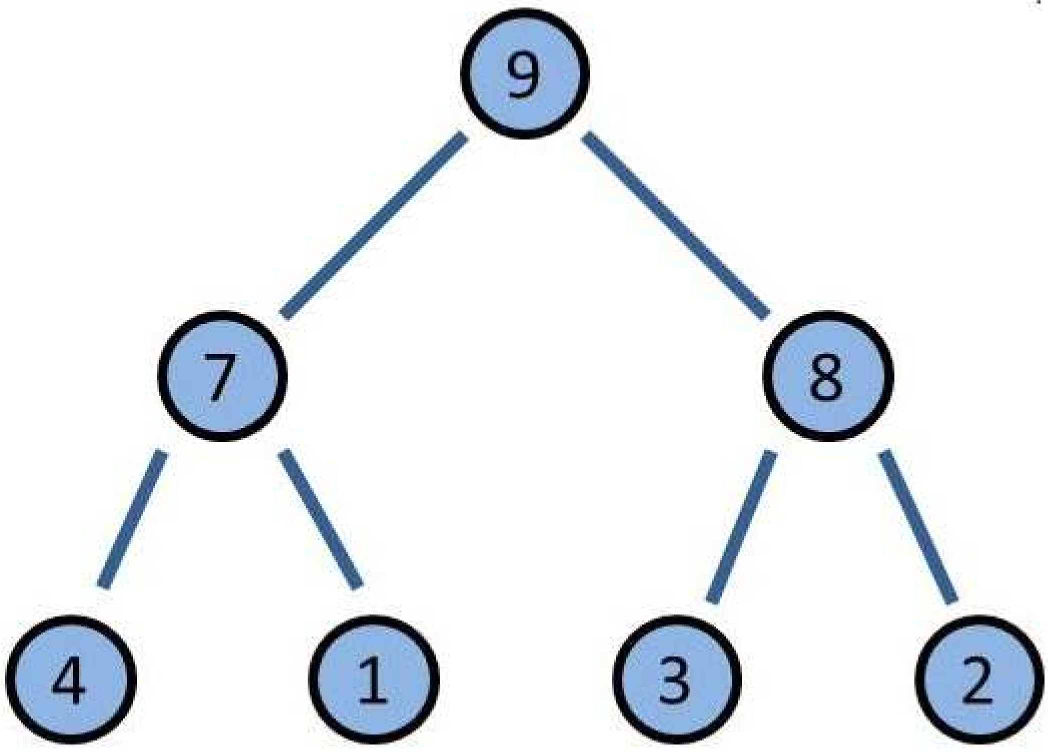
Heaps

This is another popular data structure implemented in C++ using a range.



> **i What is a heap?**
> A heap is a binary search tree in which parent elements are always bigger than their child elements. Heap trees are optimized for the efficient sorting of elements.

We can create a max heap with `std::make_heap`. With `std::push_heap`, we can push new elements on the heap. Conversely, we can pop the largest element with `std::pop_heap` from the heap. Both operations respect the heap characteristics. `std::push_heap` moves the last element of the range onto the heap; `std::pop_heap` moves the biggest element of the heap to the last position in the range. We can check with `std::is_heap` if a range is a heap. We can use `std::is_heap_until` to detemine at which position the range stops being a heap. `std::sort_heap` sorts the heap.

The heap algorithms require that the ranges and algorithms use the same sorting criterion. If not, the program is undefined. Per default, the predefined sorting criterion `std::less` is used. If we use our sorting criterion, it has to obey the strict weak ordering. If not, the program will be undefined.

`std::make_heap`: Creates a heap from the range.

```
void make_heap(RaIt first, RaIt last)
void make_heap(RaIt first, RaIt last, BiPre pre)
```

`is_heap`: Checks if the range is a heap.

```
bool is_heap(RaIt first, RaIt last)
bool is_heap(ExePol pol, RaIt first, RaIt last)

bool is_heap(RaIt first, RaIt last, BiPre pre)
bool is_heap(ExePol pol, RaIt first, RaIt last, BiPre pre)
```

`is_heap_until`: Determines at which position the range stops being a heap.

```
bool is_heap_until(RaIt first, RaIt last)
bool is_heap_until(ExePol pol, RaIt first, RaIt last)

bool is_heap_until(RaIt first, RaIt last, BiPre pre)
bool is_heap_until(ExePol pol, RaIt first, RaIt last, BiPre pre)
```

`sort_heap`: Sorts the heap.

```
void sort_heap(RaIt first, RaIt last)
void sort_heap(RaIt first, RaIt last, BiPre pre)
```

`push_heap`: Pushes the last element of the range onto the heap. `[first, last-1)` has to be a heap.

```
void push_heap(RaIt first, RaIt last)
void push_heap(RaIt first, RaIt last, BiPre pre)
```

`pop_heap`: Removes the biggest element from the heap and puts it at the end of the range.

```
void pop_heap(RaIt first, RaIt last)
void pop_heap(RaIt first, RaIt last, BiPre pre)
void pop_heap(RaIt first, RaIt last, BiPre pre)
```

With `std::pop_heap`, we can remove the biggest element from the heap. Afterward, the biggest element is the last element of the range. To remove it from the heap `h`, use `h.pop_back`.

```
#include <algorithm>
#include <iostream>
#include <vector>

int main(){

  std::cout << std::boolalpha << std::endl;

  std::vector<int> vec{4, 3, 2, 1, 5, 6, 7, 9, 10};
  for (auto v: vec) std::cout << v << " ";
  std::cout << std::endl;

  std::make_heap(vec.begin(), vec.end());
  for (auto v: vec) std::cout << v << " ";
  std::cout << std::endl;

  std::cout << "std::is_heap(vec.begin(), vec.end()): " << std::is_heap(vec.begin(), vec.end(

  std::cout << std::endl;

  vec.push_back(100);
  std::cout << "std::is_heap(vec.begin(), vec.end()): " << std::is_heap(vec.begin(), vec.end(
  std::cout << "*std::is_heap_until(vec.begin(), vec.end()): " << *std::is_heap_until(vec.beg
  for (auto v: vec) std::cout << v << " ";
  std::push_heap(vec.begin(), vec.end());
  std::cout << "std::is_heap(vec.begin(), vec.end()): " << std::is_heap(vec.begin(), vec.end(
  std::cout << std::endl;
  for (auto v: vec) std::cout << v << " ";

  std::cout << "\n\n";

  std::pop_heap(vec.begin(), vec.end());
  for (auto v: vec) std::cout << v << " ";
  std::cout << std::endl;
  std::cout << "*std::is_heap_until(vec.begin(), vec.end()): " << *std::is_heap_until(vec.beg
  vec.resize(vec.size() - 1);
  std::cout << "std::is_heap(vec.begin(), vec.end()): " << std::is_heap(vec.begin(), vec.end(

  std::cout << std::endl;

  std::cout << "vec.front(): " << vec.front() << std::endl;

  std::cout << std::endl;

}
```

Heap algorithms

In the next lesson, we’ll discuss functions that find the maximum and minimum in a range.