Use a priority queue to store frontier

For both A* search and best first search we need a priority queue to store the frontier so we can choose the "best" node to expand each time. In C++ the priority queue returns the largest value in the priority queue. To manage this, we will store pairs of values (priority, node), where the priority is used to store the priority of that node. In C++ we will store this pair as a tuple which gives us a declaration of the froniter as:

```
mutable std::priority_queue<std::tuple<int,Node>> _frontier;
```

This will setup a priority queue storing pairs of integers and Nodes. However, the priority queue needs to have an ordering on the elements it is storing. We will have to define the comparison operator in the header as follows:

```
bool operator<(const std::tuple<int,Node>& first, const std::tuple<int,Node>& second);
```

and it's implementation in the cpp file:

```
bool operator<(const std::tuple<int,Node>& first, const std::tuple<int,Node>& second) {
  return std::get<0>(first) < std::get<0>(second);
}
```

You should lookup the tuple class on cppreference.com. You can use std::make_tuple to create tuple instances and std::get to access each element in the tuple. I can help you with this if necessary.

Search methods

Since we don't care about the length of the solution you find, I suggest using a best first search strategy. Possible heuristic:

- The number of tiles out of place
- The Manhattan distance from each tile to its desired location
- A pattern database for subsets of tiles. (You might want to research disjoint pattern databases.)

Non-search based methods

A supper fast modification of the pattern database idea is to build databases of the exact move that should be performed to get a group of tiles into their exact position. And then lookup tables for other groups of tiles to put them into place.