H6: DIY Vector

In this final assignment, we combine two so-far untested concepts, namely template classes and dynamically allocating storage. Your task is to implement a template class Vector that works like std::vector. For simplicity, we implement only a tiny little subset of the interface of std::vector. This interface subset can also be found in the accompanying file `vector-skeleton.h`.

```
#include <stdexcept>

template <typename T> class Vector{
    public:
        Vector(const int newSize); // throws std::out_of_range("Vector")
        ~Vector();
        int size() const { return actualSize; };
        void resize(const int newSize); // throws std::out_of_range("Vector")
        T& at(const int index); // throws std::out_of_range("Vector")
    private:
        T* data;
        int actualSize;
        int capacity;
};
```

In this template, we call the type for the items to be stored `T` because we make no assumptions about what kind of data could be stored. In the implementation, the `data` pointer is supposed to point to an array of elements of type `T` that you should allocate in the constructor using `new[]`. The value `actualSize` holds the current size of the `Vector`. `capacity` holds the size of the `data` array as it has been allocated.

The most interesting function to implement is `resize()`. This function must allocate a new array if `newSize` is larger than `capacity`. For efficiency reasons, we only expand the array in `resize()`, but we never shrink it. This means that, if `newSize` is smaller than `actualSize`, we simply reduce `actualSize`, but we keep the array (and its `capacity`) untouched. Of course, when `resize()` allocates a new array, all data values must be copied over, and the old array must be deleted.

Your implementation must not cause any memory leaks. Please make sure that you `delete[]` everything that you allocate with `new[]`.

Please rename the skeleton file to `vector.h`. You implement the (missing) functions of the class in this file. In CodeGrade, you submit `vector.h` that will be included by the test program `vector-tester.cpp`. The test program checks the behavior of your code, and also tests if it throws the above mentioned exception whenever called with invalid parameters. `vector-tester.cpp` is also available to you for your own testing.