Algorithms & Data Structures: Lab 13

week of 28th January 2019

1 Setup

1.1 Saving your work from last week

By now you should be familiar with the operations needed to save your work. Make sure you commit your work to version control often, and always have a backup copy, ideally remotely (for example in your own account on the department's gitlab installation.)

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `13/`) alongside your existing directories.

2 Binary heap

2.1 Implementing the class

I have provided you with a skeleton implementation (as usual) of a heap, along with tests.

1. check that you can compile the skeleton implementation.
2. run the tests, and check that all the tests fail.
3. implement the methods required. I suggest attempting them in the following order:
   (a) the parent / left-child / right-child helper routines;
   (b) the simple constructor (with a single int argument, being the size of the heap storage area);
   (c) insert
   (d) maximum
   (e) the incremental constructor
   (f) maxHeapify and buildMaxHeap
   (g) the direct (non-incremental) constructor

   After each step, run the tests once more, and check that the number of failing tests has decreased.

2.2 Counting constructor operations

Write code to construct heaps with 10, 100, 1000, 10000 elements. Count the number of SWAP or element move operations when using each of the incremental and the direct constructors. What is the worst case input for constructing a max-heap? Is it the same for the incremental and or the direct constructor? Complete the following table.

N	worst case (incremental)	worst case (direct)
1	0	0
2	1	1
3	2	1
10		
100		
1000		
10000		

Verify empirically the time complexity results given in the lectures for the two different constructors? Is one of them always cheaper (in terms of element move operations) than the other?

2.3 Implement heapsort

Implement heapsort. You will need to:

1. extend the skeleton to add a method with an appropriate signature;
2. implement the heapsort method itself;
3. write tests to check the functionality of heapsort;
4. add the tests to the test suite by editing the Makefile appropriately.

(Before you start, think about which order you will do these things in, and why.)