Algorithms & Data Structures: Lab 03

week of 15th October 2018

1 Setup

1.1 Saving your work from last week

As with previous weeks, you will use `git` to download a bundle of lab code. You might have made modifications in your downloaded copy; if you have not already done so, you need to save those modifications. First examine the changes present in your downloaded copy by issuing the following commands from the labs directory:

```
git status
git diff
```

and if you are satisfied with the changes, store them in the git version control system by doing

```
git commit -a
```

and writing a suitable commit message

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 `03/`) alongside the existing directories.

2 Dynamic Arrays

2.1 Implement the data structure

Implement a dynamic array data structure, supporting the operations detailed in the lecture. You are provided with skeleton files and test files, as usual; running `make test` in the `cpp/` or `java/` directory should provide you with a test failure report, and once you have successfully implemented a dynamic array data structure, you should be able to rerun the tests I provide with success.

Your implementation will be a little different from the one presented in the lecture, which used a pair to be the container; in real implementations, we want to name fields and methods appropriately, rather than simply re-purposing a generic data structure.

2.2 Investigating extension

In implementing your version of a dynamic array, you will have had to choose:

1. how big the initial storage vector is when a dynamic array is constructed;
2. how to extend the storage vector when it is full and a new element needs to be added.

Copy the following table into a spreadsheet (or similar document), and use the `OpCounter` class (found in directory `00/` of the lab bundle) to count the number of *writes* to memory. In this investigation:

1. make the initial size of the storage vector be 5;
2. start with the dynamic array being empty;
3. push the number of elements indicated into the dynamic array;
4. try each of the possibilities in the table as the NEWLENGTH function.

length	x + 5	2 × x	x^2
1	1	1	1
5	5	5	5
10			
50			
100			
500			
1000			
5000			
10000			

You can use `make count` in the lab directory to help you; you may want to adapt your implementation of the `DynamicArray` class, or extend `DynamicArrayCounter`, in order to be able to choose how to compute NEWLENGTH.

2.2.1 Asymptotic complexity

Make a hypothesis for how the number of memory writes grows as a function of the number of elements pushed, for each of the possibilities for NEWLENGTH. How is your answer affected by the initial size of the storage vector?

2.2.2 Storage overhead

For each of the above possibilities for the NEWLENGTH function, compute the storage overhead of the dynamic array at length 1000. You can ignore the overhead coming from the `DynamicArray` class itself, and the length field of the storage vector; calculate the overhead from how much unused space there is in the storage vector, for each of the three cases.

2.3 Submission

There will be a submission related to this lab at the end of next week (deadline **16:00 26th October 2018**); you will be asked to submit work based on your implementation of the `DynamicArray` class **and** of your investigation into the scaling properties. Make sure you save your work, and that you understand what is going on.