Algorithms & Data Structures: Lab 06

week of 12th November 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 `05/`) alongside the existing directories.

2 Linked lists (cont'd)

2.1 Sublist

By adding to your existing `SLList` class from lab 04, implement a `sublist` method such that `sublist(int start, int end)` returns a fresh list whose contents are the elements in positions between `start` (inclusive) and `end` (exclusive). You may assume that `start` is less than or equal to `end`, and that `end` is less than or equal to the length of the list.

For example, if `x` represents the three-element list `(7, 9, 14)`, `x.sublist(1,2)` should return a fresh list `(9)` and `x.sublist(2,2)` should return `NIL`.

2.2 Merge

By adding to your existing `SLList` class from lab 04, implement a `merge` method such that `merge(SLlist b)` returns the result of merging (in order) the contents of `this` with `b`. You may assume that the contents of `this` and `b` are already sorted in ascending order.

2.3 Merge sort

By adding to your existing `SLList` class, implement a `mergesort` method such that `mergesort()` returns a list of the sorted contents.

If you divide the list as evenly as possible into two parts, how many calls to `mergesort()` will there be for a list of length 8? Of length 9? Of length 15?

Write down the recurrence relation for the number of calls to `mergesort` required to sort a list of length N. Do not forget to include the first call to `mergesort`.

Copy the following table into a spreadsheet (or similar document) and fill in the blanks for the number of calls to `mergesort` required to complete the sorting operation. Check that your answers are consistent with your recurrence relation. What is the solution to the recurrence relation? (Use the master theorem).

length	calls
1	1
2	3
3	
4	
7	
8	
9	
15	
16	
32	

3 Stacks and Queues

I have provided you with an implementation of stacks and queues, based on the basic operations `SLList` class. Make sure that you have implemented those four basic operations correctly, and read my implementations of the data structures. What is the complexity of the implementation of the `enqueue` and `dequeue` queue methods?

The `StackQueue` program prints some of the contents of a stack and a queue, which are initialised and populated by `prepare`. Write code in `prepare` so that running the program (which you can do using `make sq`) prints your 8-digit student number and exits without error.

4 Submission

Submit your work for this lab to the lab submission area on the module learn.gold page. The submission area will close at **16:00** on **16th November 2018**.