COMP20007 Design of Algorithms

Week 2 Workshop

Tutorial

Welcome to the first COMP20007 tutorial. Introduce yourself to your peers, and work through the following exercises together.

Reminder: Big O notation Recall from prerequisite subjects that big O notation allows us to easily describe and compare algorithm performance. Algorithms in the class $O(n)$ take time linear in the size of their input. $O(\log n)$ algorithms run in time proportional to the logarithm of their input, (increasing by the same amount whenever their input doubles in size). $O(1)$ algorithms run in 'constant time' (a fixed amount of time, independent of their input size). We'll have more to say about big O notation this semester, but these basics will help with today's tutorial exercises.

1. Arrays Describe how you could perform the following operations on **(i) sorted** and **(ii) unsorted arrays**, and decide if they are $O(1)$, $O(\log n)$, or $O(n)$, where n is the number of elements initially in the array. Assume that there is no need to change the size of the array to complete each operation.

- Inserting a new element
- Searching for a specified element
- Deleting the final element
- Deleting a specified element

2. Linked lists Describe how you could perform the following operations on **(i) singly-linked** and **(ii) doubly-linked lists**, and decide if they are $O(1)$, $O(\log n)$, or $O(n)$, where n is the number of elements initially in the linked list. Assume that the lists need to keep track of their final element.

- Inserting an element at the start of the list
- Inserting an element at the end of the list
- Deleting an element from the start of the list
- Deleting an element from the end of the list

3. Stacks A stack is a collection where elements are removed in the reverse of the order they were inserted; the first element added is the last to be removed (much like a stack of books or plates). A stack provides two basic operations: `push` (to add a new element) and `pop` (to remove and return the top element). Describe how to implement these operations using

(i) An unsorted array

(ii) A singly-linked list

4. Queues A standard queue is a collection where elements are removed in the order they were inserted; the first element added is the first to be removed (just like lining up to use an ATM). A standard queue provides two basic operations: `enqueue` (to add an element to the end of the queue) and `dequeue` (to remove the element from the front of the queue). Describe how to implement these operations using

(i) An unsorted array

(ii) A singly-linked list

Can we perform these operations in constant time?

5. Bonus problem (optional) Stacks and queues are examples of *abstract data types*. Their behaviour is defined independently of their implementation — whether they are built using arrays, linked lists, or something else entirely.

If you have access only to stacks and stack operations, can you faithfully implement a queue? How about the other way around? You may assume that your stacks and queues also come with a `size` operation, which returns the number of elements currently stored.

Computer Lab

The computer labs in this subject will present a number of C programming problems. These problems will reinforce and extend the C programming skills introduced in Foundations of Algorithms, allow us to apply the algorithms and data structures introduced in this course, and help with the C programming required for the assignments.

There's a *C Programming Refresher* document on the LMS if you need some help getting up to speed.

1. Hello, World! Create a new file called `hello.c` in your favourite text editor (if you're looking for nice text editors Atom, Visual Studio Code are popular choices; you could also try `vim` if you're up for a challenge).

Write a simple C program which prints `Hello, World!`. You can compile the program using `gcc`:

```
$ gcc -Wall hello.c -o hello
```

This will create an executable file called `hello` (or `hello.exe` on Windows). You can then run your program with:

```
$ ./hello           # On Linux or Mac OS X, or
$ ./hello.exe       # On Windows
```

2. Are we having fun(ctions) yet? Download the `functions.c` file from the LMS which provides some function prototypes and testing code. It's your job to implement the functions: `product()`, `print_sum()`, `increase_by_k()`, `max_in_array()` and `my_strlen()`.

You might need to go back and look at some of the content from Foundations of Algorithms, including functions, pointers, arrays and strings. The *C Programming Refresher* on the LMS might be a good place to start.

Compile and run your program to confirm that your implementations are correct.

3. Dynamically Resizing Arrays. Write a C program that utilises dynamically resizing arrays (*i.e.,* arrays created using `malloc` and resized using `realloc`) to store an unspecified number of Student IDs for a class enrolment list.

You should initially allocate enough space for 4 Student IDs, and each time the array runs out of capacity you should double the size using `realloc`.

The program's behaviour should be as follows:

```
$ ./class_list
Enter the Student IDs for the class list, followed by an empty line:
123456
770660
345345
1222333
400500
123123

The IDs for the students in the class are:
123456, 770660, 345345, 1222333, 400500, 123123
The array contains 6 items, and has a capacity of 8.
```

Remember to `free` all the memory you have allocated. You might need to look up the man pages for `scanf`, `malloc`, `realloc`, `assert` and `free`.