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 capacity of the array to complete each operation.

	Sorted	Unsorted
Inserting a new element	$O(\log n)+O(n)=O(n)$	$O(1)$
Searching for a specified element	$O(\log n)$	$O(n)$
Deleting the final element	$O(1)$	$O(1)$
Deleting a specified element	$O(\log n)+O(n)=O(n)$	$O(n)$

2. Linked lists

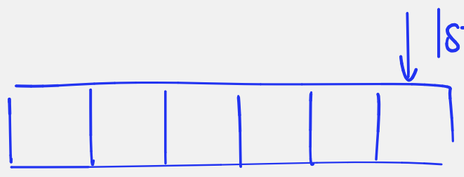
Describe how you could perform the following operations on **(i) singly-linked** and **(ii) doubly-linked lists**, and decide if they are $O(1)$, $O(\log n)$, or $O(n)$, where n is the number of elements initially in the linked list. Assume that the lists need to keep track of their final element.

	singly	doubly
Inserting an element at the start of the list	$O(1)$	$O(1)$
Inserting an element at the end of the list	$O(n)+O(1)=O(n)$	$O(1)$
Deleting an element from the start of the list	$O(1)$	$O(1)$
Deleting an element from the end of the list	$O(n)+O(1)=O(n)$	$O(1)$

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

1. An unsorted array



take control of the last element

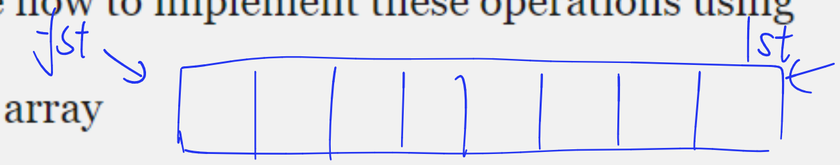
2. A singly-linked list

take control of the head element.

4. Queues

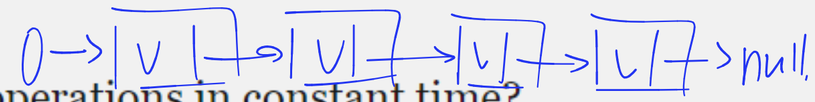
A standard queue is a collection where elements are removed in the order they were inserted; the first element added is the first to be removed (just like lining up to use an ATM). A standard queue provides two basic operations: `enqueue` (to add an element to the end of the queue) and `dequeue` (to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

1. An unsorted array



while the last element as' dequeue'.
take control of the first element as the' enqueue'

2. A singly-linked list



head -> enqueue. tail -> dequeue.

Can we perform these operations in constant time?

5. Bonus problem (Homework)

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.

double stacks