

CIS\*2420 - Data Structures

Quiz 1 - Fall 2002 - Thursday October 3, 2002

University of Guelph, Department of Computing & Information Science

Name	
Id.	

**Instructions**

- Write your name and student id number in the box above.
- This is an **CLOSED BOOK** exam. Time allowed: 70 minutes.
- Place all answers in the spaces provided on the question pages. **JUSTIFY** each answer appropriately.
- This exam counts 6% toward your final grade in this course. This exam is worth 70 points. The weight of each question is indicated in square brackets by the question number.
- This exam is not impossible, but the questions do not necessarily have obvious answers. Think about each question for couple of minutes before answering it.
- There should be 4 questions and 7 pages in this exam booklet. You are responsible for checking that your exam booklet is complete.
- Please write your answers **CLEARLY**. If I cannot read your answer, the mark on that question will be 0.

Question 1 [10 marks total]: Circle T or F for each of the statements and explain why:

Part 1.1 [2 marks]: T  F

if  $f(n)$  and  $g(n)$  are both  $O(h(n))$ , then  $f(n) + g(n)$  is  $O(h(2n))$ .

False. Big O notation is only concerned with powers of  $n$ , not constant factors.  $f(n) + g(n)$  would still be  $O(h(n))$  because they are only added.

Part 1.2 [2 marks]: T  F

A class is an abstract class if it fails to implement all methods declared in the interface which the class implements.

False. An abstract class is a class which holds only lower classes and no actual instantiations of the class. Ex. 'fruit' would be an abstract class because 'fruit' covers many different types of fruit, but there is no such thing that is only a 'fruit' and not a specific type of fruit.

Part 1.3 [2 marks]:  T  F

Algorithm A uses  $1000n \log n$  operations, while algorithm B uses  $n^2$  operations, we say that A is faster than B.

In this case, A is  $O(n)$  and B is  $O(n^2)$  so A is said to be faster. A may or may not actually be faster, depending on how large  $n$  is.  $n^2$  might be less than  $1000n \log n$  but A is the faster algorithm when considering all possible sizes of  $n$ .

Part 1.4 [2 marks]: T  F

The following Java code is correct:

```
int b = 10;
Integer a = new Integer(20);
int sum = a + b;
```

One cannot add an object (Integer) to an int.

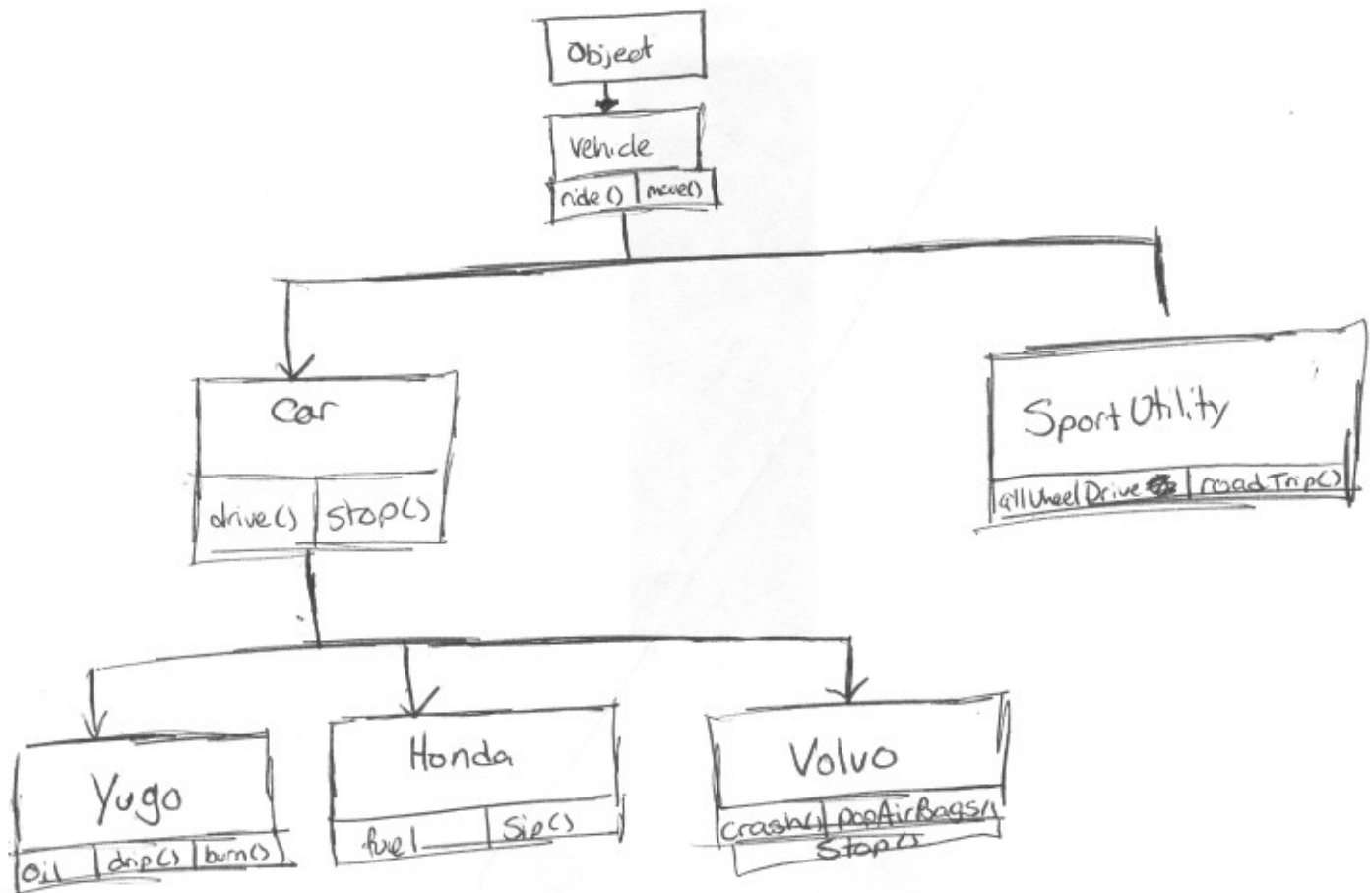
Part 1.5 [2 marks]: T  F

Using SinglyLinkedLists to implement the Stack ADT is not more efficient than using the DoublyLinkedLists.

Using Singly linked lists IS more effective for a stack because a stack adds and removes only from one side of the list. The second pointer should never be used in a stack and hence it is a waste of system resources.

Question 2 [15 marks]: Draw a single class inheritance diagram for the following set of classes:

- class Vehicle extends Object and defines methods ride() and move().
- class Car extends Vehicle and defines methods drive() and stop().
- class Yugo extends Car and defines instance variable oil and methods drip() and burn().
- class Honda extends Car and defines instance variable fuel and method sip().
- class Volvo extends Car and defines methods crash(), popAirBags(), and stop().
- class SportUtility extends Vehicle and defines instance variable allWheelDrive and method roadTrip().



15

Question 3 [15 marks]: Counting all primitive operations for each statement, characterize the worst-case running time (in terms of  $n$ ) of the following algorithm using Big-Oh notations, please explain your answer

```

Let A be a given array of n integers.
for i ← 0 to n - 1 do
  if A[i] = 0 then
    for j ← 0 to i do
      Let A[i] ← A[i] + A[j].
    end for
  end if
end for
    
```

worst case scenario

1<sup>st</sup> for loop - runs n times always  $O(n)$

if loop - will enter if  $A[i] = 0$   
 - worst case scenario - every  $A[i] = 0$ , so enters every time ✓

2<sup>nd</sup> for loop - runs i times → how so  
 - i in terms of n so  $O(n)$

$O(n)$  loop running inside of  $O(n)$  loop ∴ code is  $O(n^2)$  ✓

$$\frac{n(n+1)}{2}$$

**Question 4 [30 marks total]:** Suppose you are asked to use two stacks, `inStack` and `outStack`, as your only instance variables to implement the `Queue` abstract data type in the class `QueueObject` as follows:

```
public class QueueObject implements Queues {
    StackObject inStack;
    StackObject outStack;

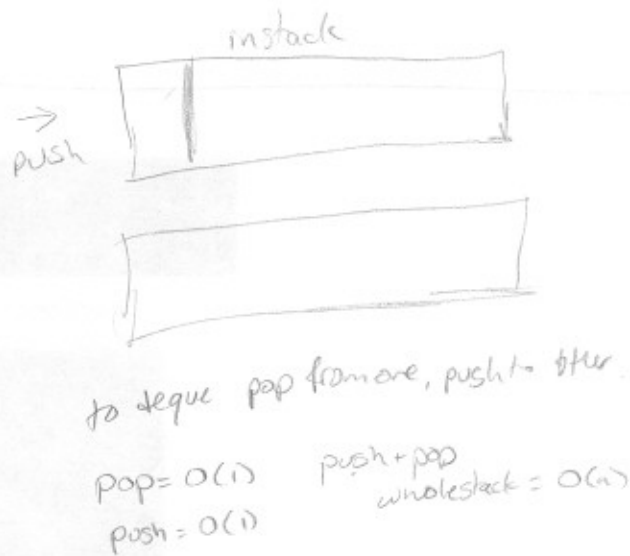
    QueueObject(){
        // ***** ASSUME THIS IMPLEMENTATION *****
    }

    public int size(){
        // ***** ASSUME THIS IMPLEMENTATION *****
    }

    public boolean isEmpty(){
        // ***** ASSUME THIS IMPLEMENTATION *****
    }

    public void enqueue(Object anObject){
        // ***** YOU MUST IMPLEMENT THIS *****
    }

    public Object dequeue() throws EmptyQueueException{
        // ***** YOU MUST IMPLEMENT THIS *****
    }
}
```



Please note that `Queues` is the interface for the `Queue` ADT, and `StackObject` is the class which implements the `Stack` ADT. For this question you may not use any ADT other than `Stack` (i.e. `StackObject` class) making no assumptions as to how the `Stack` is implemented.

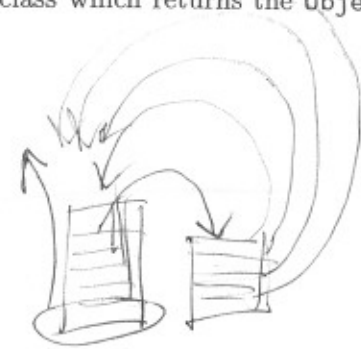
**Part 4.1 [5 marks]:** Implement the method `enqueue(Object anObject)` for the `QueueObject` class which adds the `anObject` at the tail of the `Queue`.

```
public void enqueue (Object anObject)
{
    inStack.push (Object an Object);
}
✓
```

S/S

20

Part 4.2 [15 marks]: Implement the method dequeue() for the QueueObject class which returns the Object found at the front of the Queue and removes it from the Queue.



- 1) ~~find~~ find size of inStack
- 2) for 0-(n-2) pop off + push onto outStack
- 3) pop off last as dequeued Object
- 4) ~~pop off~~ for 0-(n-2) pop off + push onto inStack

public Object dequeue () throws Empty Queue Exception

```
{
    if (inStack.isEmpty() == false)
    {
        throw Empty Stack Exception ("Stack is empty");
        return null;
    }
}
```

```
else {
    Object ret = null;
    Object temp = null;
    int size = inStack.size();
    for (int i = 0; i < (size - 2); i++)
    {
        temp = inStack.pop();
        outStack.push(temp);
    }
    ret = inStack.pop();
    for (int i = 0; i < (size - 2); i++)
    {
        temp = outStack.pop();
        inStack.push(temp);
    }
    return ret;
}
return null;
}
```

10  
15

why 2?

no need to put them back! (-5)  
see solutions.

Part 4.3 [10 marks]: Using the Big-Oh notations, state and explain the complexity of your implementation of each of the methods enqueue() and dequeue(). *one time only.*

Hint: please be careful with the complexity of dequeue(), you must optimize it.

enqueue(): ~~Pop~~ Push method on a stack is  $O(1)$  ✓

∴ enqueue() =  $O(1)$  ✓

dequeue: check empty =  $O(1)$

Pop off instack + push on outstack: Pop =  $O(1)$ , Push =  $O(1)$ , done  $n$  times  
∴  $O(n)$

Pop off last as return value:  $O(1)$

Pop off outstack + push on instack: Pop =  $O(1)$ , Push =  $O(1)$ , done  $n-1$  times  
∴  $O(n)$

return value:  $O(1)$

there is a big difference between + and \* (5)

dequeue =  ~~$O(n) \cdot O(n)$~~  =  $O(n^2)$

∴ dequeue is  ~~$O(n^2)$~~

$\frac{5}{10}$