CPSC 331 — Term Test #1 Solutions
February 13, 2012

Name: _____

Please **DO NOT** write your ID number on this page.

Instructions:

Answer all questions in the space provided.

Point form answers are acceptable if complete enough to be understood.

No Aids Allowed.

There are a total of 45 marks available on this test.

Duration: 90 minutes

ID Number: _____

Question	Score	Available
1		10
2		8
3		5
4		7
5		11
6		4
Total:		45

1. Short answer questions — you do *not* need to provide any justifications for your answers. Just fill in your answer in the space provided.

(1 mark) (a) True or false: black box tests for a function cannot be designed until after the function has been implemented.

Answer: False

(1 mark) (b) True or false: if you do not know the postcondition of a function, then you cannot test whether it works correctly with white-box testing.

Answer: True

(1 mark) (c) True or false: an algorithm with worst-case running time $f(n)$ is faster than an algorithm with worst-case running time $g(n)$ on all inputs if $f(n) \in o(g(n))$.

Answer: False

(1 mark) (d) True or false: the exponential function a^n for $a > 1$ is in $\omega(n^d)$ for any $d > 0$.

Answer: True

(1 mark) (e) In general, is the non-empty stack `s` in its original state after the statement `s.push(s.pop())` is executed? Yes or no?

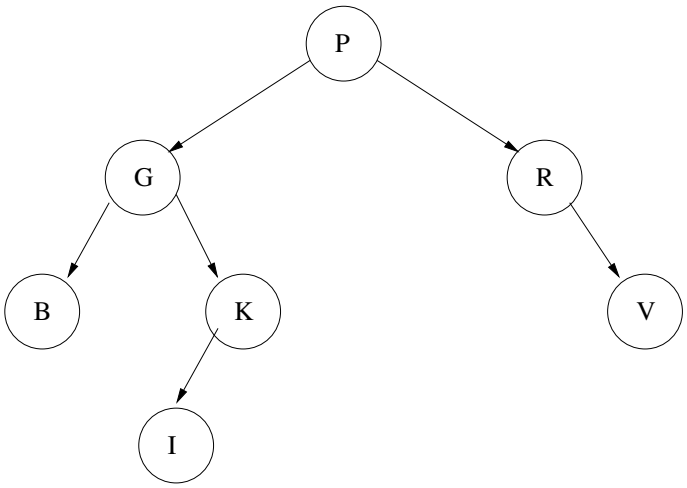
Answer: yes

(1 mark) (f) True or false: all operations of the `stack` abstract data type have worst-case running time $\Theta(1)$ when implemented with an array.

Answer: False

(4 marks)

(g) Consider the following binary tree `T`.



Which nodes are:

- leaves?

  `B,I,V`

- ancestors of `K`?

  `K,G,P` (note that `K` is technically its own ancestor)

What is the

- size of `T`?

  7

- height of `T`?

  3

2. Consider the following algorithm that copies the entries of an array to a second array in reverse order.

   PRECONDITION: `A` is a non-null array of integers
   POSTCONDITION: `B` contains the entries of `A` in reverse order
   $n = A.length$
   initialize B to be an integer array of length n
   for i from 0 to $n - 1$ **do**
   $B[i] = A[n - 1 - i]$
   end for
   return B

(4 marks)

(a) State **two** problems with the statement

$$B[j] = A[n - 1 - j] \text{ for } 0 \leq j \leq i;\ 0 \leq i < n$$

that show it is *not* a correct loop invariant for the example. Justify your answer.

Solution:

- the condition $0 \leq i < n$ does not hold after the last iteration of the loop, because $i = n$ at this point.
- the condition $B[j] = A[n - 1 - j]$ for $0 \leq j \leq i$ does not hold when $k = 0$, because, in general, $B[0] \neq A[n - 1]$ before the loop iterates.

(2 marks)

(b) Give a correct loop invariant for the example (a proof of correctness is **not** required).

Solution:

$$B[j] = A[n - 1 - j] \text{ for } 0 \leq j < i;\ 0 \leq i \leq n$$

(2 marks)

(c) Explain how your loop invariant can be used to prove that the entire algorithm is partially correct.

Solution:

- prove that the preconditions imply the truth of the loop invariant before the execution of the loop body
- prove that if the loop invariant holds and the loop guard is false, then the postconditions hold.

3. Consider the behaviour of the following algorithm when it is given a positive integer n as input (tests whether an integer is a power of 2):

```
int m = n
while m > 1 do
  if m is odd then
    return false
  end if
  m = m/2
end while
return true
```

(3 marks)

(a) Explain why the function $f(m) = \lfloor \log_2(m) \rfloor$ is a valid *loop variant* for the while-loop in the example. Recall that $\lfloor x \rfloor$ returns the largest integer less than or equal to the positive real number x.

Solution:

- $f(m)$ is an integer-valued function
- $f(m)$ decreases by 1 after each iteration, as $\log_2 m/2 = (\log_2 m) - 1$.
- $f(m) = 0$ when $m = 1$, and in the worst case, the function terminates when $m = 1$.

(2 marks)

(b) Give a function $T(n)$ such that the above algorithm uses $\Theta(T(n))$ steps on input n in the **worst case**. Briefly justify your answer (i.e., you do not need to give an explicit operation count for the algorithm nor a formal proof that a function describing the number of operations is in $\Theta(T(n))$).

Solution: $T(n) = \log_2 n$.

The number of steps is in $\Theta(\log_2 n)$, because the loop iterates at most $\lfloor \log_2 m \rfloor$ times (loop variant evaluated at initial value) and each iteration requires a constant number of steps.

4. Assume that f and g are functions mapping the natural numbers to the natural numbers:

$$f, g : \mathbb{N} \to \mathbb{N} \ .$$

(2 marks) (a) Define **big-O** by saying what it means when "$f \in O(g)$." A written definition is required (pictures will receive no marks).

Solution: $f \in O(g)$ if there exist real constants $c > 0$ and $N_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq N_0$.

(2 marks) (b) Define **little-o** by saying what it means when "$f \in o(g)$." A written definition is required (pictures will receive no marks).

Solution: $f \in o(g)$ if for every real constant $c > 0$ there exists a real constant $N_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq N_0$.

(3 marks) (c) Prove that if $f \in o(g)$ then $f \in O(g)$.

Solution:

- By assumption $f \in o(g)$, so we known that for every real constant $c > 0$ there exists a real constant $N_0 \geq 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq N_0$.
- To prove that $f \in O(g)$, we only need that there exists at least one real constant $c' > 0$ such that $g(n) \leq c \cdot g(n)$ for all $n \geq N_0$ for some other constant $N_0 \geq 0$.
- By our assumption that $f \in o(g)$ we have that any real constant > 0 satisfies this property, so by definition $f \in O(g)$.

5. The following questions deal with the `Queue` abstract data type.

(3 marks)

(a) Define the three main operations of the `Queue` abstract data type. Use the versions that throw exceptions when preconditions are violated (as opposed to returning `null` references).

Solution:

- `void add(T x)`: Adds `x` to the *rear* (or tail) of the queue.
- `T remove()`: Removes and returns the element at the *front* (head) of the queue.
  Throws a `NoSuchElementException` if the queue is empty.
- `T element()`: Returns the element at the *front* (head) of the queue without removing it (leaving the queue unchanged).
  Throws a `NoSuchElementException` if the queue is empty.

(3 marks)

(b) Describe, using pseudocode, how to implement the `remove` operation efficiently when a circular array is used to represent a queue.

How to implement the "remove" operation:

if the queue is empty **then**
  throw a `EmptyQueueException`
end if
$x = Q[head]$
$head = head + 1 \bmod Q.length$
$size = size = 1$
return x

(3 marks)

(c) Describe, using pseudocode, how to implement the `add` operation efficiently when a singly linked list is used to represent a queue.

How to implement the "add" operation:

create new node with data x and next reference *null*
if isEmpty() **then**
  $head = newNode$
else
  $tail.next = newNode$
end if
$tail = newNode$
$size = size + 1$

(2 marks)

(d) Are there any advantages to using a **doubly linked list** to implement a queue? Why or why not?

Solution: There are no advantages to using a doubly linked list to implement a queue because

- the only required operations are adding to the rear of the queue, removing from the front of the queue, and retrieving the data stored at the front of the queue
- identifying the head of the queue with the head (first element) of the list and the tail with the last element of the list means that all these operations can be implemented efficiently (constant time) using just a singly-linked list, so the extra overhead of storing previous references is not required.

6. The following questions deal with binary trees and binary search trees.

(2 marks) (a) State the **binary search tree property**.

Solution: If T is nonempty, then

- The left subtree T_L is a binary search tree including all dictionary elements whose keys are *less than* the key of the element at the root
- The right subtree T_R is a binary search tree including all dictionary elements whose keys are *greater than* the key of the element at the root

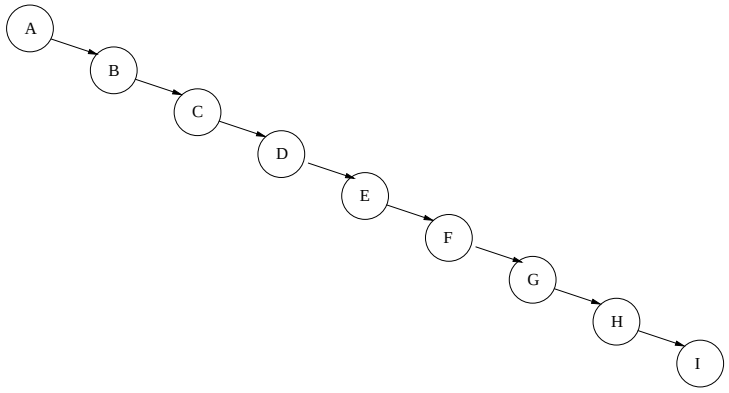
(2 marks) (b) Draw **binary search trees** containing the elements of

$$\{A, B, C, D, E, F, G, H, I\}$$

as keys with the following properties:

- the height is as large as possible

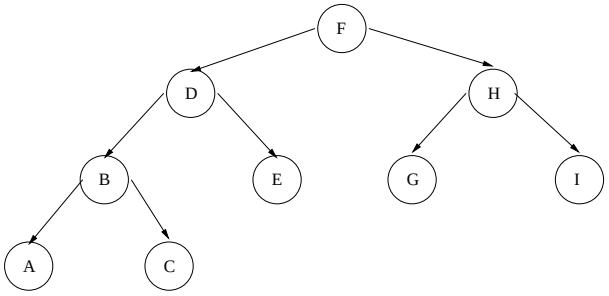
  Solution: One possible solution is



  Any *valid* binary search tree of height 8 was accepted.

- the height is as small as possible

  Solution: One possible solution is



  Any *valid* binary search tree of height 3 was accepted.