CPSC 331 — Term Test #2 Solutions

March 26, 2007

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 50 marks available on this test.

Duration: 90 minutes

ID Number: ________________________________

0.4pt0pt

Question	**Score**	**Available**
1		10
2		10
3		9
4		14
5		7
Total:		50

0.4pt0pt

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

(a) True or false: binary search is especially well-suited for searching a dictionary that is implemented as an ordered linked list.

Answer: F

(b) True or false: merge sort uses $O(n)$ auxiliary space.

Answer: T

(c) True or false: the insertion sort algorithm is faster than merge sort on data which is nearly sorted.

Answer: T

(d) True or false: the double hashing technique yields hash functions that are useful with the chaining collision resolution mechanism.

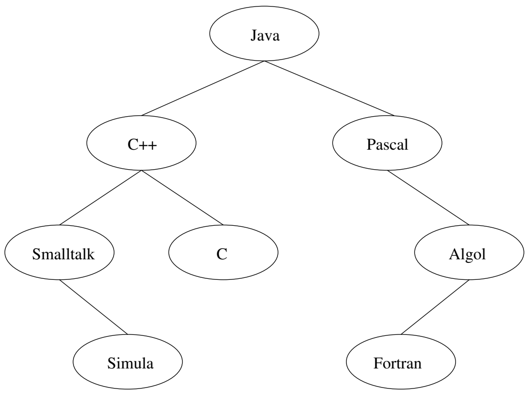
Answer: F

(e) Is the level-order traversal of a binary tree best implemented with a stack or a queue?

Answer: Queue

(f) In the binary tree below, which node follows Simula in a preorder traversal? Which node follows Simula in a postorder traversal?

Preorder: C Postorder: Smalltalk



(g) Consider the `search` function for the `dictionary` abstract data type. Using big-Oh notation, fill in the following table to indicate the *worst-case* asymptotic running time as a function of n, where n is the number of entries in the dictionary, assuming that the key being searched for is *not* in the dictionary.

Data Structure	**worst-case running time**
red-black tree	$O(\log n)$
hash table with chaining	$O(n)$
hash table with open addressing	$O(n)$

2. Consider the red-black tree data structure.

(5 marks) (a) State the five properties satisfied by a binary search tree that is also a red-black tree.

Solution: The five properties of a red-black tree, taken straight from the lecture notes, are:

- Every node is either red or black.
- The root is black.
- Every leaf (NIL) is black.
- If a node is red, then both its children are black.
- For each node, all paths from the node to descendant leaves contain the same number of black nodes.

(2 marks) (b) Define the *black height* of a node in a red-black tree.

Solution: The black-height of a node x is the number of black nodes on any path from, but not including, the node x down to a leaf.

(3 marks) (c) Give an informal description of how black height is used to prove a worst-case bound on the height of a red-black tree.

Solution: Outline of proof:

- prove lower bound on tree size in terms of black-height
- prove bound on height in terms of black-height
- combine to prove main theorem bounding height as a function of the tree size.

3. Consider **hash tables** using the **open addressing with linear probing** collision resolution mechanism, using **table size** $m = 7$ and the hash function

$$h(k, i) = k + ci \pmod{m}, \quad c = 2$$

for which we assume that the key k is an integer.

(2 marks) (a) Draw the hash table (with the above table size and hash function) that would be produced by inserting the following values into an initially empty table:

$$17, 21, 38, 24$$

Your hash table:

0	1	2	3	4	5	6
21		24	17		38	

(3 marks) (b) Describe an algorithm **using pseudocode** that can be used to search for a given value in a hash table with open addressing and linear probing *using the hash function described above*.

Solution:

```
Search(k)
  $i = 0$
  repeat
    $j = k$ mod $m$
    if $T[j] == k$ then
      return $j$
    end if
    $j = j + 2$ mod $m$
    $i = i + 1$
  until $T[j] ==$ NIL or $i == m$
  return Report that $k$ is not found
```

(2 marks) (c) Give one example of a set of 4 input values that, when inserted into an empty hash table as described above, would cause a search for the key 11 to result in the worst-case number of operations. Explain why your inputs do in fact result in a worst case for the search for 11.

Solution: Any four values, not including 11, that are all congruent to 11 mod 7 is acceptable. For example, since $11 \equiv 4 \pmod 7$, the keys

$$4, \quad 18, \quad 25, \quad 32$$

will work, because each of these is also 4 mod 7.

These inputs result in the worst case for the search for 11 because they all have the same initial probe value, namely 4. Thus, if the hash table contains these four values, a search for 11 will have to traverse all four values before verifying that 11 is not in the table.

(2 marks) (d) What is the expected number of comparisons required for an unsuccessful search in the hash table using open addressing with linear probing and $c = 1$? Your answer should be given as a function of both n (number of values stored in the hash table) and m (the size of the hash table). Under what assumption(s) does this estimate hold?

Solution: The expected number of comparisons is

$$\frac{1}{2}\left(1 + \left(\frac{1}{1 - n/m}\right)^2\right) \ .$$

Writing α in place of n/m is also acceptable, provided that it is explicitly stated that $\alpha = n/m$.

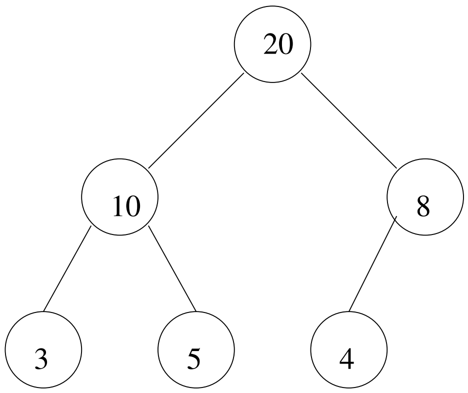
This estimate holds under the assumption that each of the m^n sequences of possible initial probes corresponding to the keys $k_1, \ldots, k_n$ are equally likely.

4. The following questions deal with the Heap Sort algorithm.

(2 marks) (a) Draw the binary tree representation of the Max-Heap stored in the following array:

0	1	2	3	4	5
20	10	8	3	5	4

Solution:



(2 marks) (b) Give an array that represents the Max-Heap obtained after calling **DeleteMax** on the heap from Question 4a.

Your array:

0	1	2	3	4	5
10	5	8	3	4	

(3 marks) (c) Describe the **MaxHeapify** operation **using pseudocode**.

Solution:

```
Max-Heapify($A$, $i$)
  $\ell$ = left($i$),    $r$ = right($i$)
  if ($\ell <$ heap-size($A$)) and ($A[\ell] > A[i]$) then
    $largest = \ell$
  else
    $largest = i$
  end if
  if ($r <$ heap-size($A$)) and ($A[r] > A[largest]$) then
    $largest = r$
  end if
  if $largest \neq i$ then
    Swap: $tmp = A[i]$; $A[i] = A[largest]$; $A[largest] = temp$
    Max-Heapify($A$, $largest$)
  end if
```

(3 marks) (d) Give a simple English description (or pseudocode if you prefer) of the **DeleteMax** algorithm.

Solution:

English description:

- copy the value from the node that must be deleted (last leaf in the heap) to the root
- call **Max-Heapify** on the root to restore heap-order
- return the value that was initially at the root.

Pseudocode:

```
**Delete-Max**($A$)
  **if** heap-size($A$) $> 1$ **then**
    $largest = A[0]$; $A[0] = A[\text{heap-size}(A) - 1]$
    heap-size($A$) = heap-size($A$) $- 1$; **Max-Heapify**($A$, 0)
    **return** $largest$
  **else if** heap-size($A$) $= 1$ **then**
    heap-size($A$) $= 0$
    **return** $A[0]$
  **else**
    **throw** *EmptyHeapException*
  **end if**
```

Note that only **one** of the english description or pseudocode was required.

(4 marks) (e) Give a simple English description (or pseudocode if you prefer) of the **heap sort** algorithm.

Solution:

English description:

- use Build-Max-Heap to convert the input array into a representation of a heap by:
  - applying Max-Heapify to every subtree starting at the last non-leaf node and working up to the root
- repeatedly use Delete-Max to extract the largest element in the unsorted part of the array (the part that represents the heap)
- move the extracted element into its correct position, noting that this position is no longer required to represent the heap after the call to Delete-Max

Pseudocode:

```
heapSort($A$)
  $n$ = heap-size($A$)
  if $n > 1$ then
    Build-Max-Heap($A$)
    $i = n - 1$
    while $i > 0$ do
      $largest$ = Delete-Max($A$)
      $A[i] = largest$
      $i = i - 1$
    end while
  end if
```

Note that only **one** of the english description or pseudocode was required.

5. Consider the following algorithm for sorting an array. Assume that the function inorder(T, B) stores the result of an inorder traversal of a binary search tree T in an array B.

```
TreeSort($A$, $n$)
PRECONDITION: $A$ is a array of $n \geq 1$ integers
POSTCONDITION: returns array with the elements of $A$ in increasing order
  Initialize $T$ to be an empty binary search tree
  for $i$ from 0 to $n-1$ do
    insert$(T, A[i])$
  end for
  inorder$(T, B)$
  return $B$
```

(3 marks) (a) What is the worst-case running time of this algorithm as a function of n, the number of integers in the array A? Justify your answer, and give a characterization of the input that results in the worst case.

Solution: The worst-case running time is $O(n^2)$ because:

- the for-loop executes n times
- worst-case cost of the loop body (binary tree insertion) is $O(n)$ steps

The inorder traversal requires time $O(n)$ and hence does not affect the overall asymptotic runtime.

Two examples of worst-case inputs are arrays that are already sorted and arrays in which the elements are in reverse order.

(2 marks) (b) What is the average-case running time of this algorithm as a function of n, the number of integers in the array A? Justify your answer.

Solution: The average case running time is $O(n \log n)$ because:

- average height of a binary tree, and hence the average-case runtime for binary tree insertion, is $O(\log n)$
- thus, inserting n elements into a binary search tree will require running time $O(n \log n)$ on average

(2 marks) (c) What is the worst-case running time of this algorithm as a function of n, the number of integers in the array A, assuming that a red-black tree is used instead of a binary search tree? Justify your answer.

Solution: The worst case running time using a red-black tree is $O(n \log n)$ because:

- worst-case height of a red-black tree, and hence the worst-case runtime for insertion, is $O(\log n)$
- thus, inserting n elements into a red-black tree will require running time in $O(n \log n)$ in the worst case.