CPSC 331 — Term Test #2 Solutions

March 26, 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 50 marks available on this test.

Duration: 90 minutes

ID Number: ____________________________________

Question	Score	Available
1		10
2		10
3		8
4		12
5		10
Total:		50

(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: the worst-case running time of binary search on a sorted array with n elements is in $O(n^2)$

Answer: <u>T</u>

(b) True or false: heap sort uses $\Theta(n)$ auxiliary space.

Answer: <u>F</u>

(c) True or false: the best case running time of selection sort is in $\Theta(n)$.

Answer: <u>F</u>

(d) True or false: quadratic probing yields probe sequences that are a permutation of all cells in a hash table.

Answer: <u>F</u>

(e) True or false: in a red-black tree, it is forbidden for a black node to have two red children.

Answer: <u>F</u>

(f) True or false: an array can be used to implement a binary heap.

Answer: <u>T</u>

(g) Using asymptotic notation, fill in the following table to indicate the *most accurate* statement of the *worst-case* running time as a function of n, where n is the number of entries in the data structure.

Operation	**Data Structure**	**Worst-Case Running Time**
delete	binary search tree	$\Theta(n)$
delete	red-black tree	$\Theta(\log n)$
delete	hash table with open addressing	$\Theta(n)$
deleteMax	binary heap	$\Theta(\log n)$

2. Consider a **hash table** using the **chaining** collision resolution mechanism, with **table size** $m = 7$ and the hash function

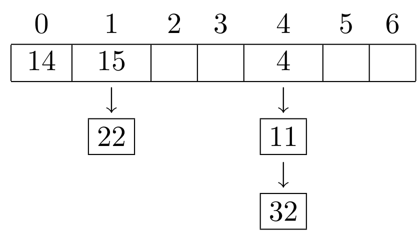
$$h(k) = k \pmod 7,$$

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

(4 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:

$$22, 32, 15, 11, 14, 4$$

Solution:



(3 marks) (b) Describe an algorithm **using pseudocode** that can be used to search for a given value in a hash table with chaining.

Solution:

search(k)
  $curr = H[k \bmod 7]$
  while $curr \neq null$ **do**
    if $curr.key == k$ **then**
      return $curr.value$
    end if
    $curr = curr.next$
  end while
  return Report that k is not found

(1 marks) (c) What is the expected number of comparisons required for an unsuccessful search in a hash table with chaining? 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).

Solution: $O(n/m)$

(2 marks) (d) Under what assumption does this estimate hold? Give both the name and a brief definition of this assumption.

Solution: Holds under the Simple Uniform Hashing assumption, which states that

- Each key is hashed to location ℓ with the same probability, $\frac{1}{m}$, for $0 \leq \ell < m$.
- Each key is hashed to a location *independently* of where any other key is hashed to.

3. The following questions deal with the selection sort algorithm.

(3 marks) (a) Give the arrays resulting after the first three iterations of the outer loop of selection sort when applied to the following array:

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

After 1st iteration:

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

After 2nd iteration:

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

After 3rd iteration:

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

(5 marks) (b) Give pseudocode for the selection sort algorithm.

Solution:

```
selectionSort(A)
  for i from 0 to n − 2 do
    min = i
    for j from i + 1 to n − 1 do
      if A[j] < A[min] then
        min = j
      end if
    end for
    swap(A[i], A[min])
  end for
```

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

(4 marks) (a) Describe an algorithm (using pseudocode) that can be used to *merge* two sorted arrays together, producing a sorted array including the entries from both inputs, using a number of operations that is linear in the sum of the sizes of the input arrays in the worst case.

Solution:

merge(A_1, A_2, B)
  $n_1 = length(A_1)$; $n_2 = length(A_2)$
  Declare B to be an array of length $n_1 + n_2$
  $i_1 = 0$; $i_2 = 0$; $j = 0$
  while ($i_1 < n_1$) **and** ($i_2 < n_2$) **do**
    if $A_1[i_1] \leq A_2[i_2]$ **then**
      $B[j] = A_1[i_1]$; $i_1 = i_1 + 1$
    else
      $B[j] = A_2[i_2]$; $i_2 = i_2 + 1$
    end if
    $j = j + 1$
  end while

  {Copy remainder of A_1 (if any)}
  while $i_1 < n_1$ **do**
    $B[j] = A_1[i_1]$; $i_1 = i_1 + 1$; $j = j + 1$
  end while

  {Otherwise copy remainder of A_2}
  while $i_2 < n_2$ **do**
    $B[j] = A_2[i_2]$; $i_2 = i_2 + 1$; $j = j + 1$
  end while

(4 marks) (b) Give pseudocode for the **Merge Sort** algorithm. You may use the "merge" algorithm discussed in Part (a) as a subroutine.

Solution:

mergeSort(A, B)
  $n = length(A)$
  if $n == 1$ **then**
    $B[0] = A[0]$
  else
    $n_1 = \lceil n/2 \rceil$
    $n_2 = n - n_1$
    Set A_1 to be $A[0], \ldots, A[n_1 - 1]$
    Set A_2 to be $A[n_1], \ldots, A[n - 1]$
    mergeSort(A_1, B_1)
    mergeSort(A_2, B_2)
    merge(B_1, B_2, B)
  end if

(2 marks) (c) Give a recurrence relation defining the number of steps required by Merge Sort to sort an array of size n in the worst case.

Solution:

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_2 n & \text{otherwise} \end{cases}$$

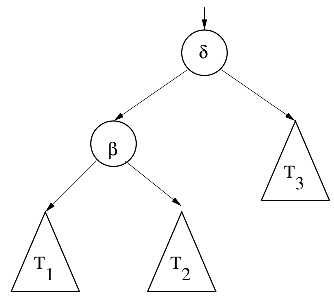
(2 marks) (d) State, using asymptotic notation, the number of operations required by Merge Sort to sort an array of size n in the worst case as an explicit function of n

Solution:

$$T(n) \in \Theta(n \log n)$$

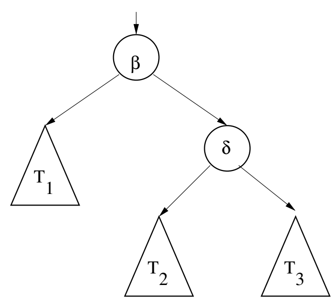
5. Consider the red-black tree data structure.

(3 marks) (a) Draw the binary tree that results from performing a right rotation about the node δ on the following tree.

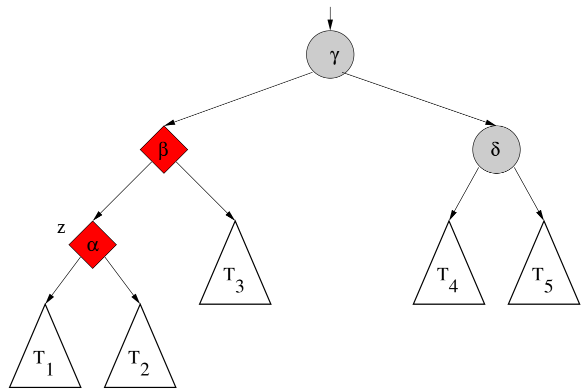


The objects labelled T_1, T_2, and T_3 are sub-trees, possibly equal to `NIL`.

Solution:



(2 marks) (b) Consider a tree that is produced from a red-black tree after a node has been inserted, but the insertion operation has not yet completed. The following tree describes one possible case arising during the adjustment phase:



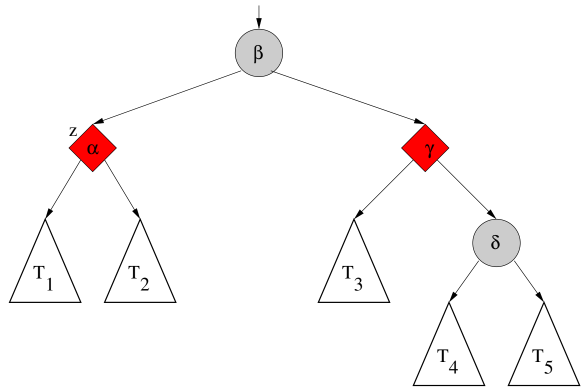
The shaded diamonds represent nodes of color red, and the circles represent nodes of color black. As above, the triangle-shaped objects represent subtrees, possibly equal to `NIL`.

Suppose that the black height of T_5 is equal to b. Give the black heights of T_1, T_2, T_3, and T_4 as functions of b.

Solution: If the black height of T_5 is equal to b, then in order for the black heigh to be well-defined for this subtree, we must have the following:

- black height of T_1, T_2, and T_3 is $b + 1$
- black height of T_4 is b

(c) After the appropriate adjustment to the previous example, the following tree is obtained:



(2 marks) What adjustment steps are required to transform the previous example to this one?

Solution: The required adjustment steps are:

- rotate right at γ
- recolor β and γ

(3 marks) Are any further adjustment steps required to transform this into a red-black tree? Justify your answer.

Solution: No further adjustment steps are required, because:

- the parent of z is no longer red
- assuming that the black heights of T_1 through T_5 are such that black height is well-defined before the transformation, it will still be well-defined after the transformation.

Alternative answer:

- by assumption (the loop invariant for the while loop in the adjustment stage of insert), the only problem with the tree was the fact that the red node z had a red parent, so after the transformation the while loop terminates and the result is a valid red-black tree.