CPSC 221 2019W1: Midterm Exam 2

November 7, 2019

SOLUTION KEY YEEEEEEhaw

bootleg version

1 Who gets the marks? [1 marks]

Please enter your 4 or 5 digit CSID in this box: [ ]

2 Things are still as complex as before [12 marks]

Each item below is a description of a data structure, its implementation, and an operation on the structure, or an algorithm with inputs. In each case, choose the appropriate worst case complexity from the list below. The variable n represents the number of items (keys, data, or key/data pairs) in the structure, unless otherwise stated. In answering this question you should assume the best possible implementation given the constraints, and also assume that every array is sufficiently large to handle all items, unless otherwise stated.

A $\Theta(1)$

B $\Theta(\log n)$

C $\Theta(n)$

D $\Theta(n \log n)$

E $\Theta(n^2)$

F None of these complexities is appropriate.

Place the LETTER corresponding to your response on the line beside each scenario.

C Add 2 to every key in an AVL tree.

C/B Find the maximum value in an AVL tree.

F Suppose you have an AVL tree of n keys, and a query set of $\log n$ keys. Determine how many keys from the query set are in the tree.

E Insert integer keys n to $\frac{n}{2}$, in that order, into an initially empty Binary Search Tree (not necessarily balanced).

B Remove a key with two children from an AVL Tree which subsequently becomes unbalanced, and whose balance is restored via the maximum number of rotations.

E Create a Voronoi diagram of an image with n centers using the Breadth First algorithm from PA2 on a region with n^2 pixels.

3 Hash Shorts [14 marks]

1. **[6 marks]** The table on the left gives a hash function for a set of keys. The keys are inserted into an originally empty hash table in some order, using linear probing to handle collisions. The state of the hash table after the insertions is illustrated below the hash function.

key k	hash $h(k)$
B	2
D	5
E	1
L	2
M	2
T	0
U	0

0	1	2	3	4	5	6
T	U	M	B	L	E	D

In the table below, fill in the bubbles corresponding to the keys that satisfy each situation.

	D	E	M	T
Could have been the first key entered:	○	○	●	●
Could have been the last key entered:	●	○	○	○
Must have been entered before E:	○	○	●	●
Must have been entered after E:	●	○	○	○

2. **[2 marks]** Give an expression for the *load factor* for a hash table of size s containing t keys:

t / s

3. **[2 marks]** Which of the following statements describes a *collision* in a hash table?

- ○ Two entries are identical, except for their keys.
- ○ Two entries with different data have the exact same key.
- ● Two entries with different keys have the exact same hash value.
- ○ Two entries with different hash values have the exact same key.

4. **[4 marks]** Suppose that your hash function does not spread keys uniformly in the hash table. Which of the following can result?

- ● Poor performance for `insert(k)`.
- ● Poor performance for a successful `find(k)`.
- ○ The time it takes to compute the hash function increases with every new insertion.
- ● Uneven distribution of chain lengths in a separate-chaining hash table.
- ● Large clusters could form in a linear-probing hash table.
- ○ The same key may hash to two different indices.
- ○ A separate-chaining hash table can become 100% full.

4 Choices, miscellany, and some originality [12 marks]

1. **[4 marks]**

   Select every condition that, by itself, ensures that a binary tree with n nodes has height $O(\log n)$.

   - T For every node, the heights of its left and right child trees differ by at most 2.
   - T For every node, the sizes of its left and right child trees differ by at most 3.
   - T The number of children at every node is either 0 or 2 and every leaf has depth between d and $d+4$ for some value d.
   - F The depth of any two leaf nodes differ by at most 5.

2. **[4 marks]** Suppose A is an algorithm with a worst case running time of $O(n)$ and B is an algorithm with a worst case running time of $\Omega(n^2)$ on inputs of size n. Select all true statements:

   - F For all $n \geq n_0$ (for some constant n_0) and all inputs X of size n, $A(X)$ (A on input X) finishes before $B(X)$.
   - T For all $n \geq n_0$ (for some constant n_0) there is some input X of size n, so that $A(X)$ finishes before $B(X)$.
   - F For some n and all inputs X of size n, $A(X)$ finishes before $B(X)$.
   - T For some n there is some input X of size n, so that $A(X)$ finishes before $B(X)$.

3. **[2 marks]**

   Suppose we have an arbitrary binary tree T. What is returned by calling the function `blueMoon` with (a pointer to) the root of T as its parameter? (i.e. `blueMoon(T.root);`) Assume we have a binary tree node definition with fields `data`, `left`, and `right`. (Select the one best answer.)

```
1 int blueMoon(Node * & p) {
2   if (p == NULL) return 0;
3   int s = blueMoon(p->left) + blueMoon(p->right);
4   if (p->left != NULL && p->right != NULL) return s+1;
5   else return s;
6 }
```

   - ○ This returns the height of the tree T.
   - ○ This returns the number of edges in the tree T.
   - ● This returns the number of nodes with more than one child in the tree T.
   - ○ This returns the number of nodes with at most one child in the tree T.
   - ○ None of these choices are correct.

4. **[2 marks]**

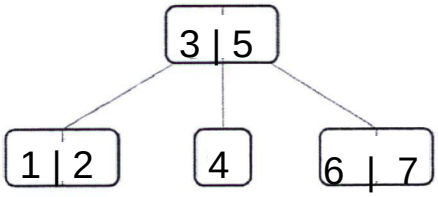
   In function `blueMoon` above, exactly how many times is the test `if (p == NULL)` made in the worst case as a function of the number, n, of nodes in the tree A.

   ○ $\lceil \log_2(n) \rceil - 1$ ○ $\lfloor \log_2(n) \rfloor + 1$ ○ n ○ $n+1$ ● $2n+1$

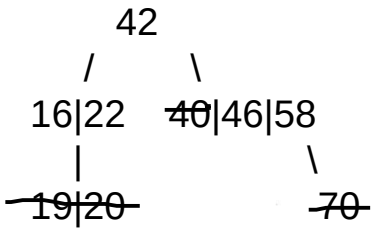
5 B-Trees [8 marks]

In this problem we explore the characteristics of order 3 B-Trees, commonly referred to as 2-3 trees.

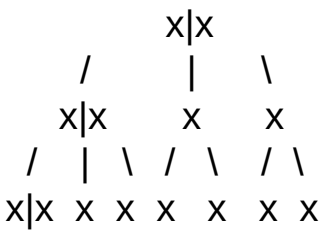
1. **[2 marks]** Add integer keys to the diagram below so that the result is a valid 2-3 tree. Only array locations that hold keys are shown, so all seven boxes should be filled.



2. **[2 marks]** This tree is *not* a valid 2-3 tree. Eliminate as few nodes as you must so that the tree is a valid 2-3 tree. (Simply cross off the nodes and/or keys you would like to eliminate.)



3. **[4 marks]** We describe a 2-3 tree as *almost perfect* if every level of the tree has exactly 1 node with 2 keys (and all the other nodes have only one key). Use a careful sketch to derive an exact expression for the number of keys in an almost perfect 2-3 tree of height h.



N(0)=2
N(1)=6
N(2)=14
...
Recurrence:
N(h)=2N(h-1)+2
Solving recurrence:
N(h)=2^(h+1)-2
=2(2^h - 1)

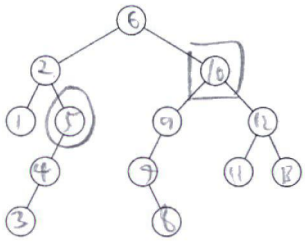
2(2^h - 1)

6 Find kth smallest key [12 marks]

We are asked to implement a dictionary structure. In addition to the normal dictionary operations, we must support a `findKmin` operation that takes an integer k and returns the (key,value) pair with the kth smallest key in the dictionary (the keys are orderable). For example, `findKmin(1)` would return the (key,value) pair with the smallest key. Let n be the number of (key,value) pairs in the dictionary.

We can choose to implement the dictionary as: 1) a hash table of size m (much bigger than n) using linear probing, 2) an AVL tree, or 3) a regular (non-self-balancing) binary search tree. Assume each tree node contains the size of its subtree. What is the tightest (i.e. largest) asymptotic lower bound we can claim on the worst-case running time of `findKmin(k)` for each of these approaches? Assume a fastest correct implementation using the approach. Here "worst-case" is over all possible insertion orders into the dictionary and values of parameter k to `findKmin(k)`.

1. **[1 mark] Hash table** Assume the hash function spreads keys uniformly in the table.
   ○ $\Omega(1)$ ○ $\Omega(\log n)$ ○ $\Omega(n)$ ● $\Omega(m)$
2. **[1 mark] AVL tree**
   ○ $\Omega(1)$ ● $\Omega(\log n)$ ○ $\Omega(n)$ ○ $\Omega(n^2)$
3. **[1 mark] Regular BST**
   ○ $\Omega(1)$ ○ $\Omega(\log n)$ ● $\Omega(n)$ ○ $\Omega(n^2)$
4. **[2 marks]** In the binary search tree shown to the right, circle the node that contains the (5th) smallest key and put a box around the node with the [10th] smallest key.



5. **[7 marks]** Suppose we choose an AVL tree to implement `findKmin`. Complete the following function, `findKmin`, that takes as input a pointer `root` to the root of an AVL tree (ordered by key) and an integer k, and returns a pointer to the node containing the kth smallest key in the tree, or `NULL` if no such key exists. You may assume the `size` field of a `Node` contains the number of nodes in its subtree.

```
struct Node { KeyT key; ValT value; int size, height; Node *left, *right; };
Node *findKMin(Node *root, int k) {
   if( root == NULL ) return NULL;
   int L;
   if( root->left == NULL ) L=0;
   else L= [ root->left->size ] ;
   if( L == [ k-1 ] ) return root;
   if( L >= [ k ] ) return findKmin( [ root->left ] , [ k ] );
   return findKmin( [ root->right ] , [ k-L-1 ] ); }
```

7 Universal Hash [8 marks]

We would like to hash k-bit integer keys down to b-bit integer indices for b much smaller than k. Thus the hash table has size $m = 2^b$. Our first attempt is to count the number of 1's in a k-bit integer x and use that as the hash function for x; call it count(x). For example, the 4-bit key $7 = 0111_2$ has count(0111_2) $= 3 = 11_2$, which is a 2-bit index.

1. **[1 mark]** What is the smallest value of b as a function of k so that every k-bit key will hash to a b-bit index if we use count() as our hash function? $\Theta($ log(k) $)$
2. **[1 mark]** Many k-bit keys hash to the same b-bit index when b is less than k. What is the maximum number of 4-bit keys that hash to the same 3-bit index using count()? 6

We decide to select a hash function from a universal set of hash functions. We pick k b-bit numbers $r_1, r_2, \ldots, r_k$ at random. We calculate the hash of x as follows: (Note that this function depends on the choice of $r_1, r_2, \ldots, r_k$. Different choices select different hash functions.)

```
hash( x )
  h = 00...0   (b bits)
  for i = 1 to k
    if ith bit of x is 1 then // Note: the first bit is the rightmost bit.
      h = h ⊕ r_i
  return h
```

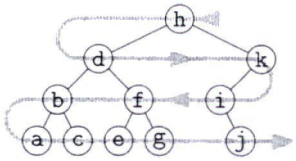
The operator $\oplus$ is bitwise exclusive-or, so $001_2 \oplus 011_2 = 010_2$, i.e., if the ith bit in x and y differ then the ith bit in $x \oplus y$ is 1, otherwise it's 0.

3. **[1 mark]** How many different hash functions are in this set? 2^(bk) m unique index for each of r0,r1...rk -> m^k=2^(bk)
4. **[1 mark]** Suppose $r_1 = 001_2$, $r_2 = 011_2$, $r_3 = 100_2$, and $r_4 = 110_2$. What is hash(0101_2)? {101}_2
5. **[1 mark]** Suppose $k = 4$ and $b = 3$. Let $x = 0101_2$ and $y = 1001_2$; and $r_1 = 001_2$, $r_2 = 011_2$, and $r_4 = 110_2$. How many possible choices of r_3 are there? 8
6. **[1 mark]** How many of choices of r_3 will cause hash(x) to equal hash(y)? 1
7. **[1 mark]** For two k-bit integers x and y with $x \neq y$ there is some i between 1 and k where the ith bits of x and y differ. Let's imagine that we have chosen $r_1, r_2, \ldots, r_k$ *except* for r_i. What is the probability we choose a b-bit integer r_i so that hash(x) equals hash(y)?

   ○ $1/2$ ● $1/2^b$ ○ $1/2^k$ ○ $1/x$

8. **[1 mark]** Does this scheme describe a universal set of hash functions? ● Yes ○ No.

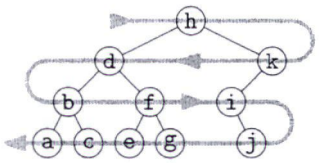
8 Snake-like Tree Traversals [9 marks]



`hdkifbacegj`

right-to-left S-order

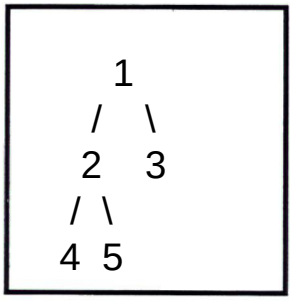
A **snake-like tree traversal** visits the nodes of a tree in level order (and prints their keys) but reverses direction at each level. For example, the snake-like order (or S-order) that starts right-to-left is shown on the left and the one that starts left-to-right is shown on the right.

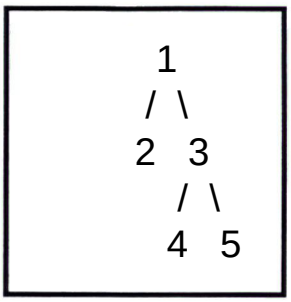


`hkdbfijgeca`

left-to-right S-order

1. **[2 marks]** Draw two **different** full binary trees (each node has 0 or 2 children) that have the same left-to-right S-orders and the same right-to-left S-orders. You will receive full credit only for the smallest examples.



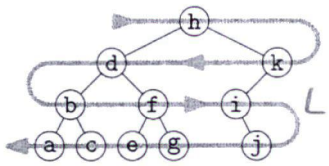


2. **[1 mark]** Suppose the right-to-left S-order of a rooted, ordered tree T with single character keys produces `abcdefghij` and the left-to-right S-order of T produces `adcbfeihgj`. How many nodes have depth 2 in T (assuming the root has depth 0)? 2

3. **[1 mark]** Given only the left-to-right and right-to-left S-orders of any rooted, ordered tree, is it always possible to determine the height of the tree? ● **Yes** ○ **No**

4. **[4 marks]** The following code uses two stacks to output the keys of a binary tree in **left-to-right S-order** given a pointer to its root. However, its not quite finished. Fill in the each blank with a single line of code so that the procedure produces the correct left-to-right S-order.

```
struct Node { KeyT key; Node *left, *right; };
void snakeOrderL2R(Node * p) {
  stack<Node *> A, B;
  A.push(p);
  while(!A.empty()){
    while(!A.empty()){
      Node *x = A.top(); A.pop();
      if(x != NULL) {
        cout << x->key << " ";
          B.push(x->left);
          B.push(x->right);
    } }
    while(!B.empty()){
      Node *x = B.top(); B.pop();
      if(x != NULL) {
        cout << x->key << " ";
          A.push(x->right);
          A.push(x->left);
} } } }
```



hkdbfijgeca
left-to-right S-order

5. **[1 mark]** What is the running time of the above code as a function of n, the number of nodes in its input? O(n)