CPSC 221 2017W2: Midterm Exam 2

March 8, 2018

SOLUTION

1 Who gets the marks? [1 marks]

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

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

Unless otherwise specified, select the **one** best answer among the choices.

MISC1 [2 marks]

Suppose that a client performs an *intermixed* sequence of stack push and pop operations. The push operations push the integers 0 through 9 in order on to the stack; the pop operations print out the return value. Which of the following sequences could not occur?

- ○ 4 3 2 1 0 9 8 7 6 5
- ○ 2 1 4 3 6 5 8 7 9 0
- ● 0 4 6 5 3 8 1 7 2 9
- ○ 4 6 8 7 5 3 2 9 1 0
- ○ All of these sequences are possible.

MISC2 (2pts)

A queue cannot be implemented using only _____ for holding data.

- ● a stack
- ○ a linked list
- ○ an array
- ○ More than one of a), b), c) can be used to fill in the blank.
- ○ None of a), b), c) can be used to fill in the blank.

MISC3 (2pts)

To justify the use of arrays for data structures of unbounded size (stacks, queues, lists, etc.), we proved that the following strategy results in $O(n)$ running time over a sequence of n inserts for an average of $O(1)$ per insertion.

- ○ When an array of size n fills, create a new array of size n, and maintain all the arrays. Do not copy any data.
- ○ When an array of size n fills, create a new array of size $2n$, and maintain all the arrays. Do not copy any data.
- ○ When an array of size n fills, create a new array of size $n + d$ for some large fixed constant d, and copy the data into the new array.
- ● When an array of size n fills, create a new array of size $2n$ and copy the data into the new array.
- ○ None of these strategies give the performance we describe.

MISC4 [2 marks]

Suppose that you have performed in-order, pre-order, post-order, and level-order traversals on an arbitrary AVL tree. If the keys from each traversal are inserted into an initially empty AVL tree, which traversal(s) will reconstruct the same structure of the original tree?

- ○ In-order
- ○ Pre-order
- ○ Post-order
- ● Level-order
- ○ All of these options will reconstruct the original tree.

MISC5 [2 marks]

Which of the statements about binary tree rotations are correct?

- ○ They can improve the asymptotic complexity of traversals after rotating
- ○ An arbitrary binary tree can be transformed into a binary search tree
- ○ Any complete binary tree can be transformed into a perfect binary tree
- ● A perfect binary tree can be transformed into a tree with a linear structure
- ○ All of the choices above are correct.

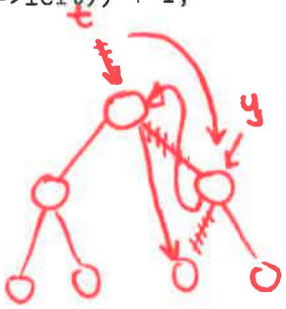
MISC6 [2 marks]

What is the real name of `mysteryFunction`? Note that in context, neither `t`, nor `t->right` will be null.

```
void mysteryFunction(treeNode * & t) {

   treeNode * y = t->right;
   t->right = y->left;
   y->left = t;
   y->height = max( height(y->right), height(y->left)) + 1;
   t->height = max( height(t->right), height(t->left)) + 1;
   t = y;

}
```



- ● `leftRotate`
- ○ `rightRotate`
- ○ `rightLeftRotate`
- ○ `leftRightRotate`
- ○ None of these choices are correct.

3 Efficiency [16 marks]

Each item below is a description of a data structure, its implementation, and an operation on the structure. In each case, choose the appropriate running time from the list below. The variable n represents the number of items (keys, data, or key/data pairs) in the structure. 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.

A $O(1)$

B $O(\log n)$

C $O(n)$

D $O(n \log n)$

E None of these running times is appropriate.

A Enqueue for a Queue implemented with an array.

A Dequeue for a Queue implemented with an array.

C Worst case for insertion into a Binary Search Tree.

C Worst case for removal from a Binary Search Tree.

C Worst case for an algorithm to return all keys that are greater than 20 and that are multiples of 3 in a Binary Search Tree.

B Worst case for insertion into an AVL Tree.

C Level order traversal of an AVL Tree.

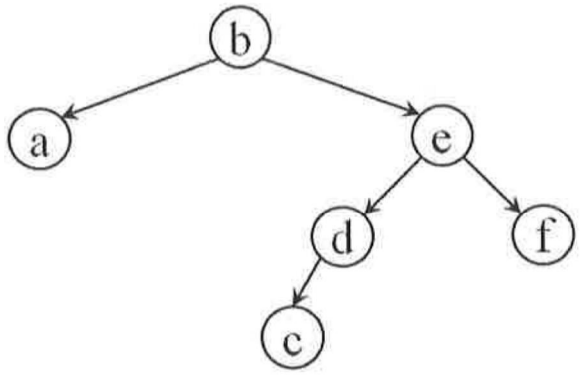
D Build an AVL tree with keys that are the numbers between 0 and n, in that order, by repeated insertion into the tree.

4 BSTs [16 marks]

Suppose we decide to implement a Binary Search Tree of non-negative integers by embedding it in an array as follows:

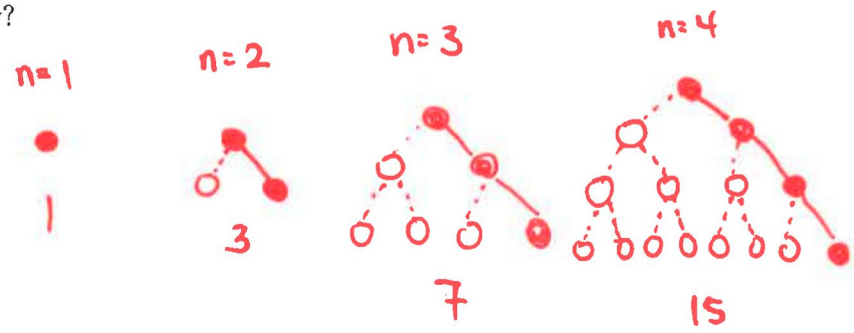
- The root's key is stored in array cell 1.
- If node v's key is in cell k, then the key of v's left child is in cell $2k$ and the key of v's right child is in cell $2k+1$.
- Any cell not corresponding to an existing node should have value -1.

(a) **[4 marks]** Show the embedding of this BST in the array below:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	b	a	e			d	f					c			

(b) **[3 marks]** What is the worst case space requirement for a BST with n elements implemented in this way?

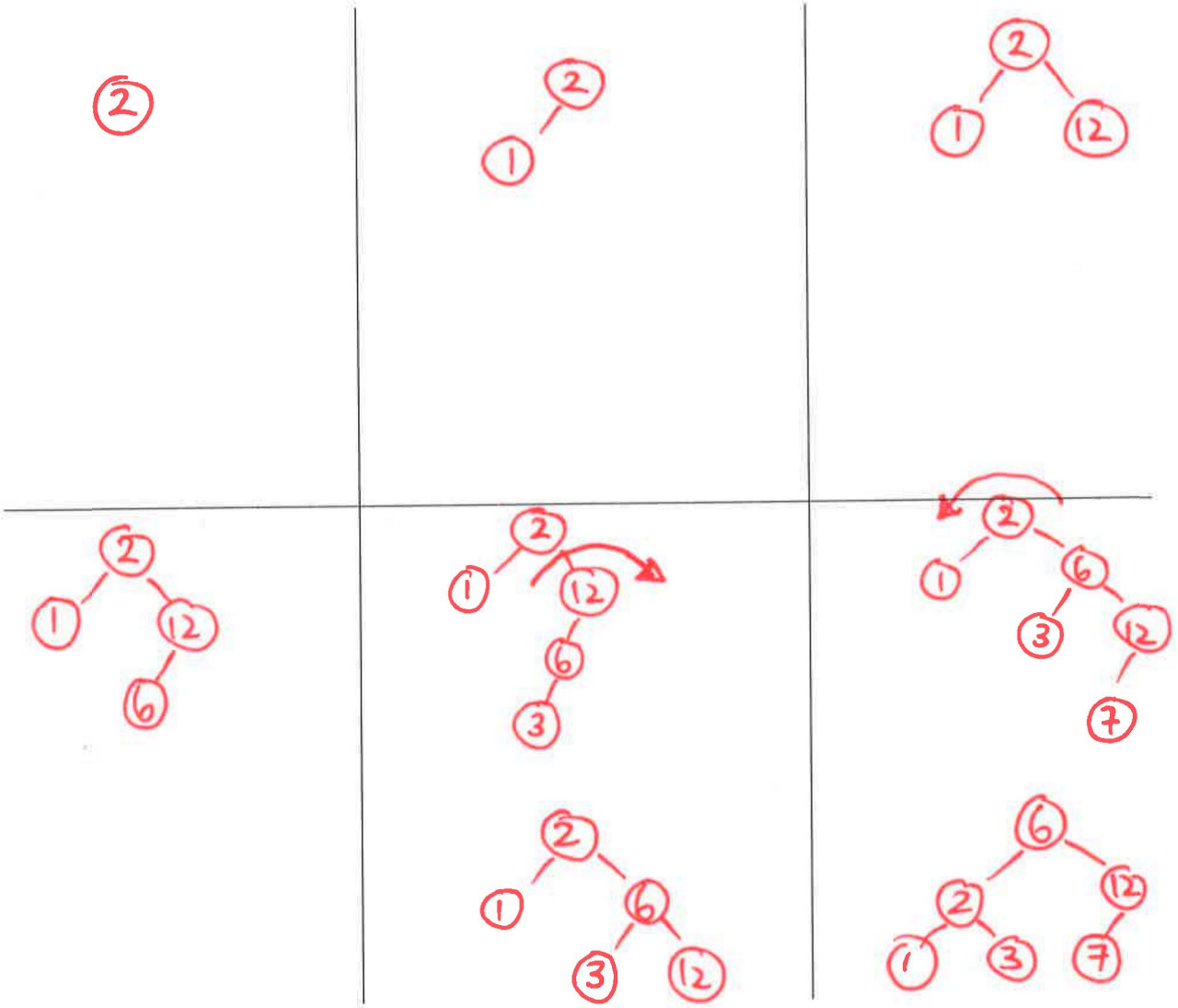


$2^n - 1 \; (+1)$

(c) **[3 marks]** Suppose we want to use the *indices* containing non-negative values in the array from part (a) as *keys* for another data structure. Do an inOrder traversal of the original BST, and as each node is processed, write its corresponding *array index* on the first available line below (reading left to right). As a sanity check, you should be writing *numbers* on the line, not letters.

2 1 12 6 3 7

(d) **[6 marks]** Create an AVL tree using the indices from part (c) as keys. Insert them into the tree in the order that they appear on the lines, reading from left to right. Show all steps of your insertions, including rotations. Please do this neatly, and make sure the graders can tell what you're doing. If you do a rotation, tell us what kind it is.



5 Stacks [11 marks]

For this problem we will design a data structure for integers called `StackSpecial` that supports all of the usual stack operations like `push()`, `pop()`, `top()`, and `isEmpty()`. It also supports an additional operation `getMin()` which returns the minimum valued element in `StackSpecial`. To implement `StackSpecial`, you should only use the `Stack` class that we provide below and no other data structure like arrays, list, .. etc. Moreover, you can use **at most two** `Stack` data structures in your implementation of `StackSpecial`. Keep in mind that the numbers pushed onto the stack are not guaranteed to be unique.

For example, consider the following `StackSpecial`:

```
18  --> TOP
15
29
19
18
```

When `getMin()` is called it should return 15, which is the minimum element in the current stack.
If we do `pop()` two times on the stack, it becomes

```
29  --> TOP
19
18
```

When `getMin()` is called, it should return 18 which is the minimum in the current stack.
Here are the `Stack` and `StackSpecial` class definitions:

```
class Stack  {
public:
    // ctors and dtor and all of the public methods, including:
    bool isEmpty();
    int top(); // returns the top item without removing it
    int pop(); // removes and returns the top item
    void push(int x);
private:
...

};
```

```
class StackSpecial  {
private:
    Stack data;
    Stack special;
public:
    bool isEmpty();
    int top(); // returns the top item without removing it
    int pop(); // removes and returns the top item
    void push(int x);
    int getMin(); // returns the smallest value in the stack
};
```

(a) **[3 marks]** Write the function `push(int x)` for the `StackSpecial` class.

```
void StackSpecial::push(int x) {
    if (x <= special.top())
        special.push(x);
    data.push(x);
}
```

(b) **[3 marks]** Write the function `pop()` for the `StackSpecial` class. You may assume the `StackSpecial` is not empty.

```
int StackSpecial::pop()
{
    if (data.top() == special.top())
        special.pop();
    return data.pop();
}
```

(c) **[2 marks]** Write the function `getMin()` for the `StackSpecial` class. You may assume the `StackSpecial` is not empty.

```
int StackSpecial::getMin()
{
        return special.top();
}
```

(d) **[3 marks]** Assume that all `Stack` class functions run in $O(1)$ time, worst case. What are the worst case running times of the functions in the `StackSpecial` class?

- `push(int x)` O(1)
- `pop()` O(1)
- `getMin()` O(1)

6 Sort of more efficient... [9 marks]

In this problem you will show us that you understand the concept of partitioning in Quicksort. Each of the lists of names below has been created by invoking Quicksort on some starting array, and stopping after 0 or 1 iteration of the partition function. For each list, fill in the box with *all* the possible pivot values for the iteration. If no iterations could have occurred, then write "none". Assume that the lists are to eventually be sorted in increasing alphabetical order.

	ex 1	ex 2	ex 3
0	Bautista	Morbidelli	Bautista
1	Abraham	Espargano	Espargano
2	Espargano	Abraham	Espargano
3	Espargano	Bautista	Abraham
4	Petrucci	Marquez	Marquez
5	Nakagami	Espargano	Morbidelli
6	Marquez	Miller	Miller
7	Rabat	Redding	Nakagami
8	Rins	Pedrosa	Pedrosa
9	Morbidelli	Rossi	Rabat
10	Redding	Rabat	Petrucci
11	Miller	Rins	Redding
12	Rossi	Zarco	Rossi
13	Pedrosa	Nakagami	Zarco
14	Zarco	Petrucci	Rins
Pivots	Espargano Espargano Zarco	none	Marquez Nakagami Pedrosa Redding

7 Recurrent meal planning [9 marks]

Geoff struggles to pay his monthly expenses, and reduces his food budget in order to make ends meet. On most days, Geoff eats the cheapest thing he can find: knockoff Kraft dinner (K). Occasionally he will eat more expensive items: fresh produce (P), or fresh meat (M), so that he does not get scurvy. However, he cannot afford to eat expensive items two days in a row. Assume that Geoff eats only one thing each day.

In this problem we'll build a recurrence to help us compute $M(n)$: the total number of different meal sequences in an n day span.

(a) **[1 marks]** Write a valid sequence of meals for a 5 day span, keeping in mind the constraint that Geoff cannot eat expensive food two days in a row: K K P K K

(b) **[1 marks]** How many choices does Geoff have if planning a meal for a single day?

$M(1) =$ 3

(c) **[2 marks]** How many different valid meal sequences can Geoff have for a period of 2 days?

$M(2) =$ 5

K < K, M, P

M - K

P - K

(d) **[1 marks]** Consider the last meal of an n day span. It is either expensive, or it is not. If it is expensive, then consider what must be true about day $n-1$. Given this, how many different choices are there for the last 2 days, ending in an expensive day? 2

K ex 2

(e) **[1 marks]** Consider the last meal of an n day span. It is either expensive, or it is not. If it is *not* expensive, then how many different choices are there for the last day? 1

ch 1

(f) **[3 marks]** Use the observations above to complete the general case of the recurrence relation below for $M(n)$, $n > 2$:

$$M(n) = M(n-1) + 2 \cdot M(n-2)$$

8 Divide and Conquer [12 marks]

Suppose you are given the following three algorithms:

ALG1 To solve a problem of size n, this algorithm solves one subproblem of size $\frac{n}{2}$ and spends an additional constant amount of time to produce the answer.

ALG2 To solve a problem of size n, this algorithm solves two subproblems of size $\frac{n}{2}$ and spends an additional constant amount of time to produce the answer.

ALG3 To solve a problem of size n, this algorithm solves two subproblems of size $\frac{n}{2}$ and spends an additional linear amount of time to produce the answer.

(a) **[6 marks]** For each algorithm write the recurrence that describes its running time on data of size n, for the general (not base) case.

i. Running time for ALG1: $T(n) =$ $T(\frac{n}{2}) + c$

ii. Running time for ALG2: $T(n) =$ $2 \cdot T(\frac{n}{2}) + c$

iii. Running time for ALG3: $T(n) =$ $2 \cdot T(\frac{n}{2}) + c \cdot n$

(b) **[3 marks]** For each recurrence relation, choose the tightest asymptotic solution for $T(n)$ from the list below. Write the letter corresponding to the correct expression in each box.

i. Asymptotic running time for ALG1: $T(n) =$ B

ii. Asymptotic running time for ALG2: $T(n) =$ C

iii. Asymptotic running time for ALG3: $T(n) =$ D

A $O(1)$
B $O(\log n)$
C $O(n)$
D $O(n \log n)$
E $O(n^2)$

(c) **[3 marks]** Label each of the following algorithms, with its appropriate general description chosen from ALG1, ALG2, and ALG3.

i. Merge sort is described by ALG 3.

ii. Binary search in a sorted array is described by ALG 1.

iii. Computing the sums of the keys in a perfect binary tree is described by ALG 2.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.