1 Who gets the marks? [1 marks]

Please enter your 4 or 5 digit CSID in this box: 60026

0 2 4 8 16 32 64

2 Choices, miscellany, and some originality [8 marks]

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

MISC1 [2 marks]

$N(h-1) = 2^h - 1 = 32 - 1 = 31$ for $h = 5$.

$42 > 2^{h+1} - 1$

$43 > 2^{h+1}$

$\log_2(h+1) < \lfloor \log 43 \rfloor$

5

Fill in the blanks to make the statement true:

A complete binary tree containing 42 nodes has height [5] and [11] leaves on the lowest level.

height 4 = perfect =

$N(h) = 2^{4+1} = 32 - 1 = 31$

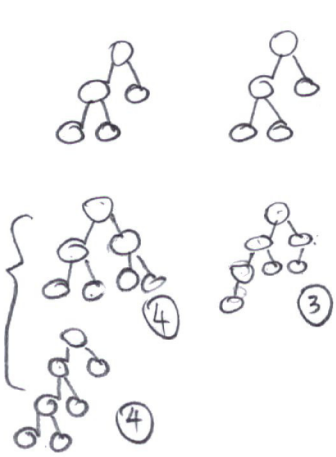
42 − 31 = 11

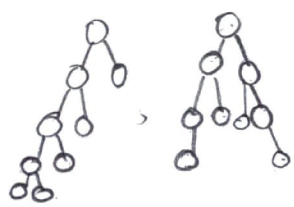
MISC2 [2 marks]

Consider two binary trees A and B, containing n keys each. Tree A is a full binary tree. Tree B is an AVL tree.

- ○ A traversal on A takes asymptotically less time than a traversal on B ✗
- ● Tree A has at least as many leaves as B.
- ○ The height of B is less than or equal to the height of A.
- ○ The height of A is less than or equal to the height of B.
- ○ More than one of the above options are true.
- ○ None of the above options is true.

$\lfloor \log_2 n \rfloor$ ≥ height

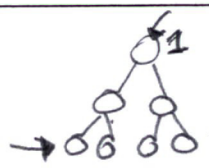




MISC3 [2 marks]

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

```
1  int springtime(Node*& nd) {
2    if (nd == NULL) {
3      return 0;
4    else if (nd->left == NULL && nd->right == NULL)
5      return 1;
6    else
7      return springtime(nd->left) + springtime(nd->right);
8  }
```

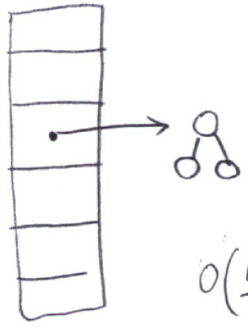


- ○ This returns the total length of all root-to-leaf paths in A.
- ○ This returns the total number of nodes in A.
- ○ This returns the number of internal nodes in A.
- ● This returns the number of leaf nodes in A.
- ○ None of these choices are correct.

MISC4 [2 marks]

Suppose you have implemented a hash table using a universal hash function, an array of size 23, and separate chaining, where the secondary storage for each cell is an AVL tree, rather than a list. Suppose you have hashed 2500 keys into that structure. Then the expected number of comparisons in an unsuccessful search for a key is most likely to be:

- ○ 1
- ○ 10
- ● 100
- ○ 1000
- ○ The answer cannot be determined.



3 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. 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 $O(1)$

B $O(\log n)$

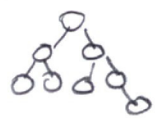
C $O(n)$

D $O(n \log n)$

E None of these complexities is appropriate.

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

C Make a copy of a complete binary tree. → Fill left left → level traversal then put int binary as insertion → $2n = O(n)$



B Find the maximum value in an AVL tree.

C Execute the `Find` function in a dictionary implemented as a binary tree (not necessarily BST).

D Insert integer keys n to $\frac{n}{2}$, in that order, into an initially empty AVL Tree.

A Perform a Right-Left double rotation in an AVL Tree.

E Execute the `fill` algorithm from PA2 on a region with n pixels. → valid question?

↳ $O(n^2)$ 3x3 pixel for each pixel neighbour

4 BTrees [9 marks]

(a) **[3 marks]** What is the maximum number of *nodes* in an order 37 BTree of height 10?

$2^{11}-1$

$= 2^{5+6}-1 = 2^6(2^5)-1$

$= (64)(32)-1$

$= 2047$

64
×32
128
1920
2048

$N(h) = 2^{h+1} - 1$

$= 2^{11} - 1$

$2^{11}-1$
$= 2047$

(b) **[3 marks]** What is the maximum number of *keys* in an order 37 BTree of height 10?

#keys = children - 1

$37^{10+1} - 1$

$= 37^{11} - 1$

min

$2\lceil \frac{37}{2} \rceil^{10} - 1$

$= 2(19)^{10} - 1$

$37^{11} - 1$

(c) **[3 marks]** Suppose that the order of a BTree is increased from m to m^2. What happens to the upper bound on the height of the structure? (Hint: $\log_b n = \frac{\log_c n}{\log_c b}$)

$\log_m n = \frac{\log_2 n}{\log_2 m}$

when m^2, $\frac{\log_2 n}{\log_2 m^2} = \frac{\log_2 n}{2\log_2 m} = \frac{1}{2}\left[\frac{\log_2 n}{\log_2 m}\right]$

$= \frac{1}{2}[\log_m n]$

The height of the structure ○ increases ● decreases.

Explain briefly and approximately how the new height differs from the old:

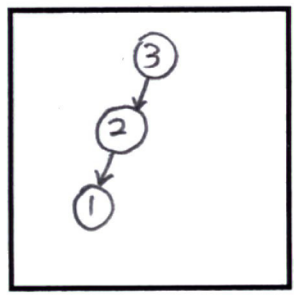
The new height decreases by 1/2 from the old one as more keys can fit in each level now.

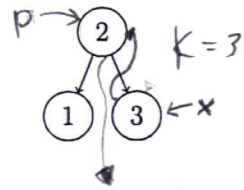
5 Mystery Tree Function [10 marks]

The following function, `foo`, takes as input a pointer `p` to the root of a binary search tree (ordered by key) and a key `K`.

```
struct Node { int key; Node *left, *right; };
Node *foo(Node *p, int K) {
    Node *x;
    if( p == NULL || K == p->key ) return p;
    if( K < p->key ) {
        x = foo(p->left, K);
        if( x != NULL ) {
            p->left = x->right;
            x->right = p;
        }
    } else {
        x = foo(p->right, K);
        if( x != NULL ) {
            p->right = x->left;
            x->left = p;
        }
    }
    return x;
}
```

1. **[2 marks]** Draw the tree resulting from calling `foo` with key 3 and a pointer to the root of:





Complete the following statements by filling in the blanks and/or selecting **all** choices that make the statement true. Assume that `p` is a pointer to the root of a binary search tree.

2. **[2 marks]** If key `K` is not in the tree, `foo(p,K)` returns [x (null)] and the tree ○ is ● is not modified.

3. **[2 marks]** If key `K` is in the tree and `q = foo(p,K)` then `q->key ==` [K] and the tree is modified so that `q` is ● the root of ○ a leaf of ○ no longer in the tree.

4. **[2 marks]** If key `K` is in the tree, after executing `foo(p,K)`, the tree is ○ an AVL tree ● a binary search tree containing ● exactly the nodes ○ all but one node of the original tree. O(log n) ← height

5. **[2 marks]** If d is the depth of the node containing key `K`, then the asymptotic running time of `foo(p,K)` is [O(d)]. → height (at worst)

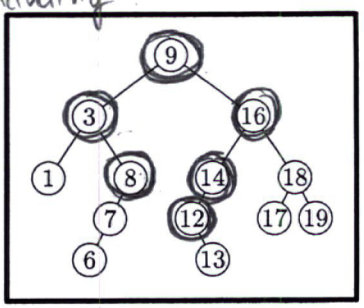
6 Print Range [10 marks]

The range is inclusive of k1 & k2

Suppose we store n (key, value) pairs in a dictionary. The keys are unique positive integers. In addition to the normal dictionary operations, we must support a **printRange** operation that takes two key values `k1` and `k2` (with `k1` $\leq$ `k2`) and prints all the keys in the dictionary between `k1` and `k2`. k1 — k2.

↑ including

1. **[2 marks]** Circle the nodes that must be visited in the picture to the right by any correct implementation of `printRange(root,k1,k2)` where `k1=8`, `k2=11`, and `root` is the root of the binary search tree shown to the right.
   Note: `printRange` can only obtain the key value in a node by visiting the node.



2. **[4 marks]** Complete the following function, `printRange`, that takes as input a pointer `root` to the root of a binary search tree (ordered by key) and two values `k1` and `k2` (with `k1` $\leq$ `k2`) and prints all the keys in the tree between `k1` and `k2` (inclusive).

```
struct Node { int key; Node *left, *right; };
void printRange(Node *root, KType k1, KType k2) {
   if( root == NULL ) return;

   if( [ root->key > k1 ] ) {   // no need = k1 because all left keys will definitely be < k1 if root->key = k1
      printRange(root->left, k1, k2);
   }

   if( [ root->key >= k1 && root->key <= k2 ] ) {
      cout << root->key << " ";
   }

   if( [ root->key < k2 ] ) {
      printRange(root->right, k1, k2);
   }
}
```

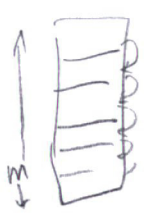
3. **[2 marks]** Suppose `root` points to the root of an **AVL tree** with n nodes. What is the runtime of `printRange(root,k1,k2)` as a function of n and the number k of keys in the tree that lie between `k1` and `k2`? Choose one.

   log n for each find k finds.

   ○ $\Theta(nk)$ ○ $\Theta(n+k)$ ○ $\Theta(\log n + k)$ ● $\Theta(k \log n)$

4. **[2 marks]** Suppose we wanted to implement `printRange` for a **hash table** using linear probing with table size m that holds n keys. What is the best (smallest) asymptotic runtime we could hope to obtain (assuming we can't choose the hash function, and assuming we don't care about the order in which they're printed)? Choose one.

   ○ $\Theta(nm)$ $m > n$ ● $\Theta(n+m)$ ○ $\Theta(n \log m)$ ○ $\Theta(m \log n)$



$\frac{n}{m}$

7 Cuckoo Hashing [8 marks]

Cuckoo hashing is an open-addressing collision resolution scheme for hash tables. It uses two hash functions h_1 and h_2. Each key k will be stored in slot $h_1(k)$ or slot $h_2(k)$ of the hash table T. Each slot can contain at most one key (or NULL if the slot is empty). The pseudo-code to the right describes how insertion works.

In words, when key k is inserted, it goes into slot $h_1(k)$. If some key b is already there, then key b gets kicked out and goes to its alternate slot (the one that is not $h_1(k)$) and settles there; perhaps kicking out another key. This repeats until all keys find a home.

```
insert( k ) {
   slot = h1(k)
   repeat {
      b = T[slot]
      T[slot] = k
      if( b == NULL ) return
      if( slot == h1(b) ) slot = h2(b)
      else slot = h1(b)
      k = b
   }
}
```

For this problem, keys are the letters from A to H with hash values $h_1(k)$ and $h_2(k)$ shown below:

$k =$	A	B	C	D	E	F	G	H
$h_1(k) =$	0	0	2	8	5	8	5	9
$h_2(k) =$	2	2	4	2	9	4	7	7

1. **[4 marks]** Show the contents of the hash table (of size 10) after inserting each of the specified keys in order. (If a slot contains NULL, please leave it blank; initially all slots contain NULL.)

	0	1	2	3	4	5	6	7	8	9
insert(A)	A									
insert(B)	B		A							
insert(C)	A		B		C					
insert(D)	A		B		C				D	

2. **[2 marks]** At this point, you might realize that the code given for `insert` might fail to work correctly. In fact, inserting one more key (after inserting keys A, B, C, and D) will cause insert to fail. Inserting which key will cause insert to fail? Choose one.

○ E ● F ○ G ○ H

3. **[2 marks]** The actual `insert` function detects this failure and attempts to fix the problem by doubling the size of the hash table, choosing two **new** hash functions h_1 and h_2 from a universal set of hash functions that map keys to the new table slots, and re-inserting all the keys from the old table (plus the key that caused the failure) into the new table. Why might this help? Choose all that apply.

- ● The probability that two keys collide decreases.
- ● The number of potential collisions decreases.
- ○ The number of hashed keys decreases.
- ○ The number of possible keys decreases.

8 Tree Traversals [12 marks]

On his way to class last week Geoff dropped his portfolio of binary trees, shattering them into mulch. Fortunately, his diary contains smudged records of many of the trees' traversals (among other mundane musings).

In addition to the set of traversals discussed in class, Geoff's notes refer to a traversal called `revPostO`, whose implementation is shown here:

```
struct Node { int key; Node *left, *right; };

void revPostO(Node * p) {
    if ( p != NULL) {
        revPostO(p->right);
        revPostO(p->left);
        cout << p->key << endl;
    }
}
```

Each problem part below contains as much information as we can read about each tree. Complete any missing traversal information, and then tell us whether or not the original tree can be reconstructed. If it **can**, then draw the tree. If it **cannot**, then draw two *different* trees that produce the given traversals.

1. **[5 marks]** The first tree *can* be drawn.

InOrder:	I	O	L	N	S	H	E	T

Left, root, right

RevPostO:	T	H	E	L	I	O	N	S

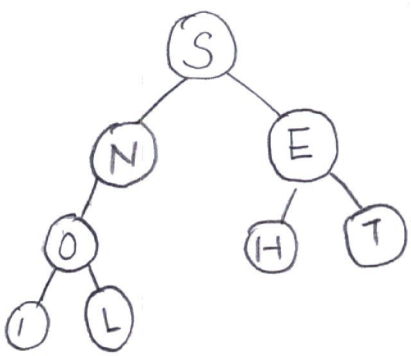
right, left, root

(a) **[1 marks]** Which key corresponds to the root of the tree? Circle it on both tables.

(b) **[1 marks]** Which keys must be in the left subtree? Underline those on both tables.

(c) **[1 marks]** Which keys must be in the right subtree? Box those on both tables.

(d) **[2 marks]** Finally, use the fact that binary trees are recursively defined structures to finish building the tree.



2. [4 marks]

PreOrder:	S	E	Y	M	O	U	R

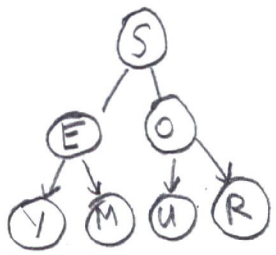
Root, left, right

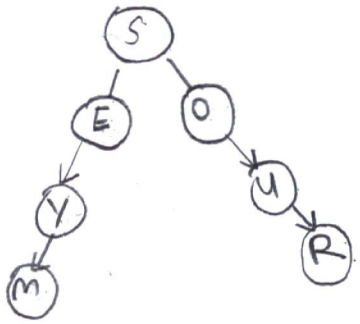
RevPostO:	R	U	O	M	Y	E	S

right, left, root.

Can these traversals be used to build a unique tree? ○ YES ● NO (If YES, then draw it. If NO, then draw two different ones.)

binary - no need to be full/complete or perfect.



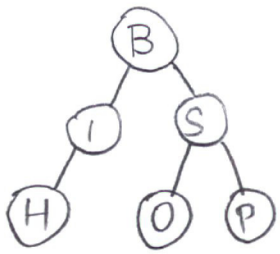


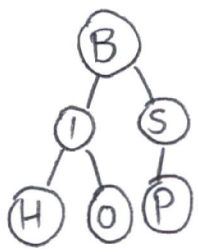
3. [3 marks]

LevelOrder:	B	I	S	H	O	P

Can this traversal be used to build a unique tree? ○ YES ● NO (If YES, then draw it. If NO, then draw two different ones.)

Binary tree not full, complete, perfect / balanced / ordered.





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.

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.