Virginia Tech

1 8 7 2

READ THIS NOW!

- Print your name in the space provided below.
- There are 6 short-answer questions, priced as marked. The maximum score is 100.
- Legibility is a requirement. Illegible responses will be penalized!
- This examination is closed book and closed notes, aside from the permitted one-page fact sheet. Your fact sheet may contain definitions and examples, but it may not contain questions and/or answers taken from old tests or homework. You may include examples from the course notes.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Until solutions are posted, you may not discuss this examination with any student who has not taken it.
- Failure to adhere to any of these restrictions is an Honor Code violation.
- When you have finished, sign the pledge at the bottom of this page and turn in the test and your fact sheet.

KEY

Name (Last, First) _____

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____

signed

1. [15 points] Prove the following Binary Tree Theorem:

Let T be a binary tree with λ levels. Then T has no more than $2^{\lambda} - 1$ nodes.

Proof:

Using strong induction on the number of levels, λ, let S be the set of all integers $\lambda \geq 1$ such that if T is a binary tree with λ levels then T has no more than $2^{\lambda} - 1$ nodes.

Base case: if $\lambda = 1$ then the tree only consists of a root node with no children. Thus there is one node which is $2^{\lambda} - 1$ if $\lambda = 1$ and so $1 \in S$.

Inductive assumption: suppose that for some integer $K \geq 1$, all the integers 1 through K are in S. That is, whenever a binary tree has M levels with $M \leq K$, it has at most $2^{M} - 1$ leaf nodes.

Let T be a binary tree with K + 1 levels. If T has the maximum number of nodes, T consists of a root node and two nonempty subtrees, say S_1 and S_2. Let S_1 and S_2 have $M_1$and M_2 levels, respectively. Since M_1 and M_2 are between 1 and K, each is in S by the inductive assumption. Hence, the number of nodes in S_1 and S_2 are no more than $2^{K} - 1$and $2^{K} - 1$, respectively. Since all the nodes of T must be nodes of S_1 or of S_2, the number of nodes in T is no more than $2^{K} - 1 + 2^{K} - 1$ + the root, which = $2^{K+1} - 2 + 1 = 2^{K+1} - 1$ Therefore, K + 1 is in S. Hence by Mathematical Induction, S = [1, ∞).

2. [20 points] Complete the implementation of the following BST member function, which is intended to determine if the current state of the BST is complete and full. The only relevant declarations from the BST implementation are that it has a member named `root` and an inner `BinaryNode` class. You may **not** invoke other tree methods. The function may call private helper functions, for which you **must** show implementations.

```
/** Returns true if the current state of the tree is Full and Complete,
 *  returns false otherwise
 *
 *  Post: the BST is not modified.
 *  Returns:
 *        true iff the BST is full and complete.
 */
public boolean fullComplete( )
{
    int nodeCount = numNodes(root);
    int level     = maxLevel(root);
    return (nodeCount == (Math.pow(2, level)-1);
}

/** Returns the number of nodes in the tree.
 *
 *  Post: the BST is not modified.
 *  Returns:
 *        number of nodes in the tree.
 */
public int numNodes(BinaryNode sroot)
{
    if (sroot == null) return 0;
    return (1 + numNodes(sroot.left) + numNodes(sroot.right) );
}

/** Returns the maximum level of the tree.
 *  Empty tree is level 0.
 *
 *  Post: the BST is not modified.
 *  Returns:
 *        max level of the tree.
 */
public int maxLevel(BinaryNode sroot)
{
    if (sroot == null) return 0;
    return (1 + Math.max(maxLevel(sroot.left), maxLevel(sroot.right)) );
}
```

3. Hash Table

a) [5 points] Suppose that a hash table implementation that uses *linear probing* to resolve collisions is initially empty, and N records, all with unique key values, are inserted into the table. Among them are K records, R_1 through R_K, all of which map to the same home slot H in the hash table. If records are inserted in the order: $R_1, R_2, R_3, \ldots, R_{K-2}\ R_{K-1}, R_K$, and no other records have H as their home slot, what can be said about the *minimum* number of slots that must be examined if we search for R_K? Why?

K
Since the table is initially empty with linear probing being used, and the records R_1 through R_K are inserted in a consecutive sequence of K cells, (possibly wrapping around the end). So, any search for R_K must begin in the home slot (holding R_1) and proceed slot by slot until R_K is found (after K comparisons), this assumes the other N-K records map to slots that do not interfere with the R_1 through R_K cluster for minimal probing.

b) [5 points] Assume the same conditions as given in part a). What can be said about the *maximum* number of slots that must be examined if we search for R_K? Why?

N
In the worst case, the insertion of the other N-K records will map to slots that interfere with the R_1 through R_K cluster.

c) [5 points] Would either answer above change if *chaining* were used to resolve collisions? Why?

It would have no effect on the answer for question a. However, the answer for question b would become K since the insertion of the other N-K records will map to buckets that do not interfere with R_1 through R_K bucket.

d) [5 points] Consider the following idea to eliminate tombstones. When deleting an element from the hash table, search for the element to be removed, when located continue the probing, (either linear or quadratic), until the last element in the probe sequence is found. Replace the deleted element with the last probe sequence element and mark the last probe sequence element's location as empty. Discuss whether this idea will work or whether it can be made to work.

This idea will *not* work since the last element in the probe sequence could have been inserted at its location due to a collision (or hashing) at a table location after the deleted element in linear probing or at a different location (or hashing) other than the deleted element in quadratic probing.

The only way this could be made to work would be to check each element along the probe sequence (either linear or quadratic) to confirm that it hashed to the same location as the element to be deleted. Only in this special circumstance could the last element of the sequence safely be moved to replace the deleted element.

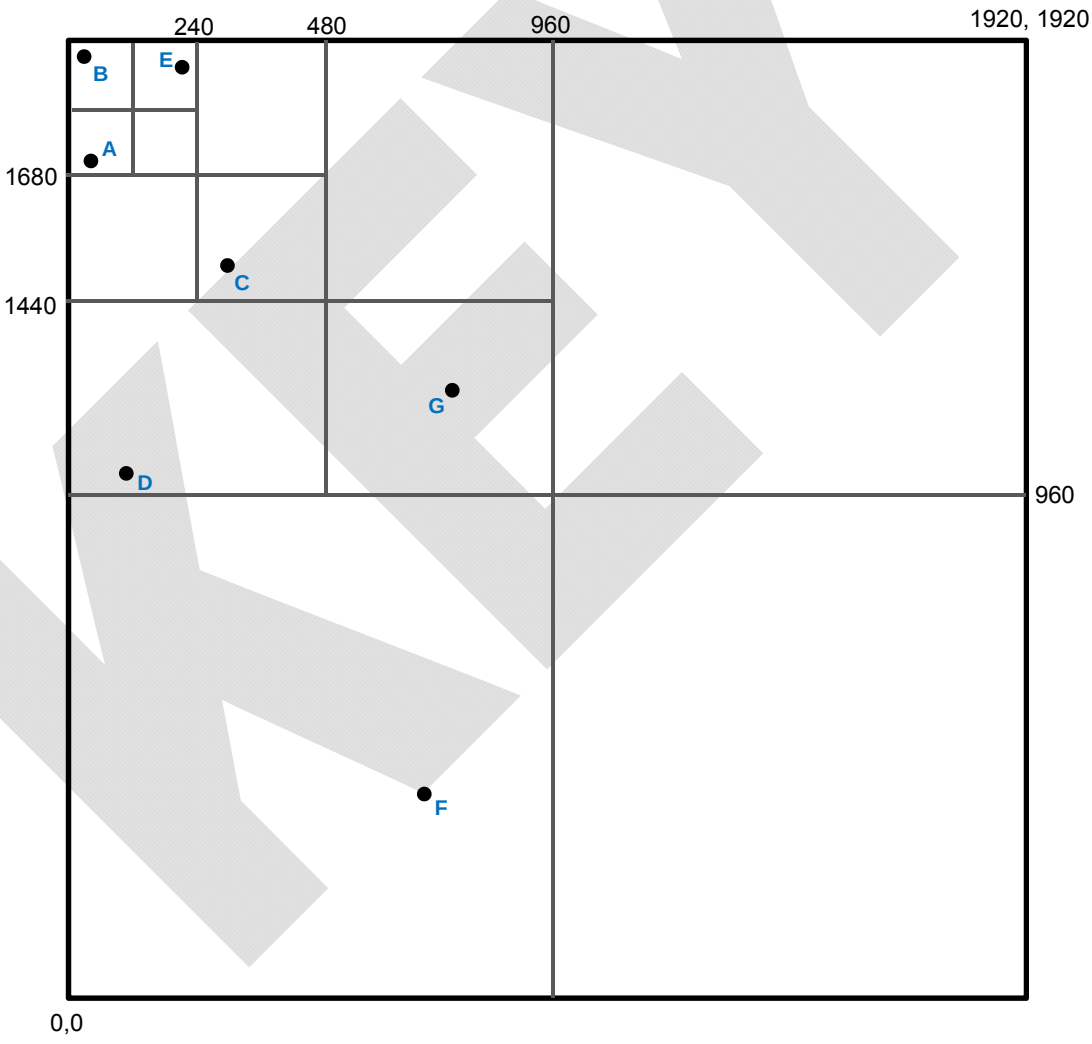
4. [20 points] A HDMI 1080p high-definition image has resolution 1920 x 1080 (width x height). In video there is often very little change in an image from one frame to the next. Many video codecs take advantage of this by only storing the regions of an image that has changed between frames.

a) Since PRQuadTrees require a square region you must set the world coordinate boundaries for the image data below to be (0, 0) .. (1920, 1920).

Draw the subdivision of the coordinate space, labeled appropriately, that would result from storing the following pixel coordinates, A-G, in a PRQuadTree, as a 2D diagram showing the partitions and the data point letter labels.

Insert the values in this order:

A(50, 1700), B(30, 1900), C(300, 1500), D(100, 1000), E(220, 1880), F(810, 440), G(880, 1244)



4.

b) Draw (or describe) how the subdivision partitioning of the coordinate space in part a) would change if the points were inserted into the PRQuadTree in the reverse order, i.e. G, F, E, D, C, B, A.

There would be no change. The structure of a PR Quadtree is independent of the order of insertion of the data.

5. [10 points] Heap

Given the following initial array representation:

0	1	2	3	4	5	6	7	8	9	10	11	12
R	M	D	X	B	P	F	Z	A	K	H	S	J

Complete the diagrams below to show the building of a **min-heap**. Show the array contents after the completed sifting of each element. Note that the heap root is stored at index zero.

0	1	2	3	4	5	6	7	8	9	10	11	12
R	M	D	X	B	***J***	F	Z	A	K	H	S	***P***

0	1	2	3	4	5	6	7	8	9	10	11	12
R	M	D	X	***B***	J	F	Z	A	K	***H***	S	P

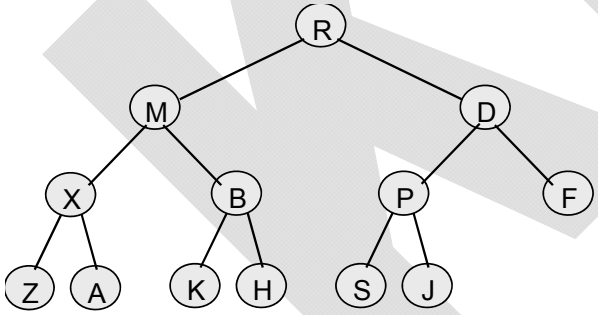
0	1	2	3	4	5	6	7	8	9	10	11	12
R	M	D	***A***	B	J	F	Z	***X***	K	H	S	P

0	1	2	3	4	5	6	7	8	9	10	11	12
R	M	***D***	A	B	J	F	Z	X	K	H	S	P

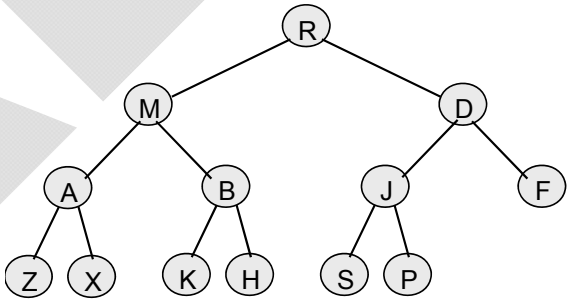
0	1	2	3	4	5	6	7	8	9	10	11	12
R	***A***	D	***M***	B	J	F	Z	X	K	H	S	P

0	1	2	3	4	5	6	7	8	9	10	11	12
A	***B***	D	M	***H***	J	F	Z	X	K	***R***	S	P

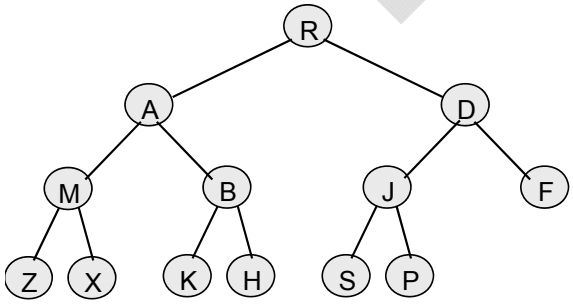
Diagrams below are not part of the required answer.

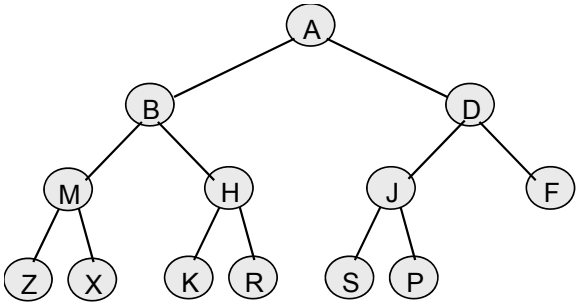


initial tree



after lowest level sifting





6. Consider the implementation and performance issues related to Pugh's probabilistic skiplist.

a) [10 points] Suppose as a slight variation to the implementation of a skip list as discussed in lecture, it is proposed that, in addition to the reference to the head node, a *current* node reference is also maintained. The *current* node reference will always be set to point to the node containing the element that was found in the last successful search.

Carefully explain in detail how the skip list search algorithm could take advantage of the *current* node reference. Do **not** give code.

The search could be modified to first compare the current node element with the target.
Case 1: If the target equals the current element then the search ends successfully.

Case 2: If the target is less than the current element then the search must begin at the head node normally.

Case 3: If the target is greater than the current element then it depends upon the level of the current node:

sub-case a: if the level of the current node is equal to the level of the head node the search begins at the current node.

sub-case b: if the level of the current node is shorter than the level of the head node the search should first start with the head node to determine if nodes past the current node can be skipped. ***(further explanation, not required)*** *If any node element, reached from the head node by examining the levels higher than the current node, is greater than the current element but less than the target the current node is ignored in the search. If any node element, reached from the head node by examining the levels higher than the current node, is equal to the target the search ends successfully. If any node element, reached from the head node by examining the levels higher than the current node, is less than the current element the search starts at the current node.*

b) [5 points] Describe how you would implement an efficient intersection algorithm in a SkipList class as a member method. The intersection operation will return a skip list object containing the common elements in the two skip lists. The two skip lists involved in the intersection must be unchanged by the operation. Do **not** give code.

A semi-efficient algorithm would be to simply perform a level zero traversal of the elements of one skip list and search for each element in the other skip list, inserting it into the intersection skip list if found. This algorithm would be O(nlogn).

A more efficient algorithm would be to traverse both skip lists concurrently. Starting at the head of both lists, if they are equal insert one into the intersection skip list and move to the next zero level element in both lists and continue. If they are not equal search for the larger of the two in the other skip list taking advantage of the level skip pointers. Stopping and inserting in the intersection when found. When not found the search must stop at the largest element less than the target and the search must switch to the other skip list to try and find the element stopped at in the prior search. This search must also take advantage of the level skip pointers. Any time a search is successful an insertion in the intersection is made and the next zero level element in both lists is used to continue. Any time a search is not successful the search switches to the other skip list as described previously.

The worst case would occur when the smaller list is a subset of the larger list and the last elements in both lists are the same. Thus the worst case efficiency of this algorithm would O(n + m + mlogm) where n & m are the lengths of the two skip lists & m < n, (i.e., the time to traverse both lists plus the time to insert all the common elements into the intersection skip list).