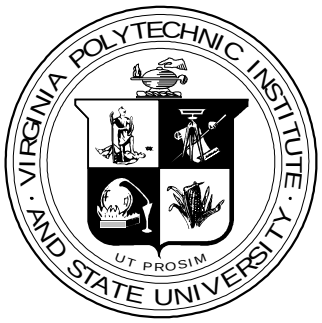


Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes. No calculators or other computing devices may be used.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 11 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- **Note that failure to return this test, or to discuss its content with a student who has not taken it, is a violation of the Honor Code.**

Do not start the test until instructed to do so!

Name Solution
printed

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

signed

1. [10 points] For each of the following sorting algorithms, give big-Theta notation for the average and worst-case running times (swaps plus compares) on input of size N:

Algorithm	Average	Worst
InsertionSort	n^2	n^2
QuickSort	n log n	n^2
HeapSort	n log n	n log n

2. [5 points] The implementation of QuickSort in Kruse/Ryba selects the pivot value from the middle of the sublist, rather than from one of the ends. Explain a reasonable motivation for that decision.

The worst case for QuickSort is achieved if each pivot is either the maximum or minimum value in the current sublist. If the list is already sorted, then choosing either the first or last element guarantees the worst case is achieved. Choosing from the middle is a hedge against that.

3. [5 points] Aside from switching to a nonrecursive implementation, describe one modification of the implementation of QuickSort given in the course notes that will improve its performance.

When sublists become relatively short, say length 10, apply an efficient alternative, such as an insertion sort to complete the sorting of that sublist. This eliminates many recursive calls.

Choose the pivot value more carefully. For example, find three distinct values in the current sublist and take their median value. If there are not three distinct values, then the sublist is nearly sorted and can be efficiently finished off using another sort, such as insertion.

4. [10 points] Indicate whether each of the following statements is true or false:

a) $f(n) = 5n^2 + 3n + 2$ is $\Theta(n)$ TRUE **FALSE**

b) $f(n) = 5n^2 + 3n + 2$ is $\Omega(n)$ **TRUE** FALSE

c) $g(n) = 3n + 100 \log n$ is $O(n^2)$ **TRUE** FALSE

d) $g(n) = 3n + 100 \log n$ is $\Omega(n)$ **TRUE** FALSE

e) $g(n) = 3n + 100 \log n$ is $O(\log n)$ TRUE **FALSE**

5. You must keep track of some data. Your options are:

   1) a binary search tree of records (assume it is well balanced)
   2) a linked-list of records stored in order of insertion
   3) an array-based list of records maintained in sorted order

   a) [12 points] Suppose that you must first build a data structure holding 2^{10} given elements, and then you must perform 2^{20} searches on that data structure. For each option, use the average case big-Θ **time** complexity results of each data structure to determine the costs associated with that data structure in this situation. Since you know exactly how many elements are stored, and how many searches must be performed, the cost values should be numbers, not expressed in terms of N.

   Note that space cost is not a consideration.

Option	Cost of construction	Cost of 2^{20} searches	Total Cost
BST	$10x2^{10}$	$10x2^{20}$	
unsorted linked list	2^{10}	2^{29}	
sorted array	2^{20} or $11x2^{10}$	$10x2^{20}$	

BST: Building costs n log n, which would be 2^{10} log 2^{10} = $10x2^{10}$.
Assuming a balanced BST, each search costs on average log n, which would be log 2^{10} = 10, so the total cost of 2^{20} searches would be $10x2^{20}$.

LL: Building costs n or 2^{10}, since each insertion has cost 1.
Search, on average costs n/2 or 2^9, and so the total cost of 2^{20} searches would be 2^{29}.

SA: Building cost depends on the method you assume. Inserting the elements one by one in sorted order would have cost n^2 or 2^{20}. Inserting the elements into an array and then sorting efficiently would cost n + n log n, which would be 2^{10} + 2^{10} log 2^{10} or $10x2^{10}$ + 2^{10} = $11x2^{10}$. Each search would have cost log n, or log 2^{10} = 10. So the total cost of all the searches would be $10x2^{20}$.

   b) [3 points] Based on your analysis above, which of the data structures would be the worst choice in the given situation?

The linked list has the worst overall cost.

6. [10 points] Use limits to prove that f(n) = n log n is NOT $\Theta(n^2)$.

By a theorem, f(n) is Θ(g(n)) if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ where c is a positive, finite constant.

Now, $\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1/(n\ln 2)}{1} = \lim_{n\to\infty} \frac{1}{n\ln 2} = 0$ so by the cited theorem, n log n is NOT Θ(n^2).

7. [10 points] Suppose that T is a full binary tree.

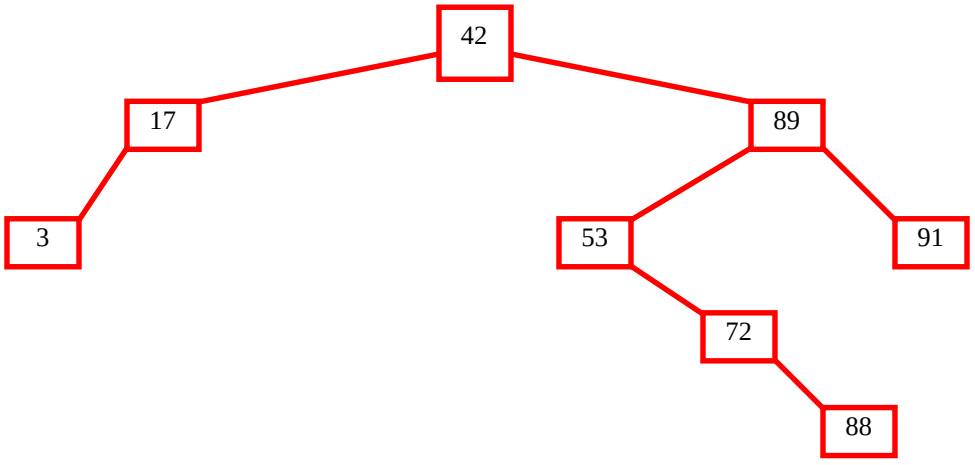
The Full Binary Tree Theorem says that an FBT with k internal nodes has k+1 leaf nodes. Applying that fact:

a) If T has 100 internal nodes then the number of leaves in T is _____**101**____.

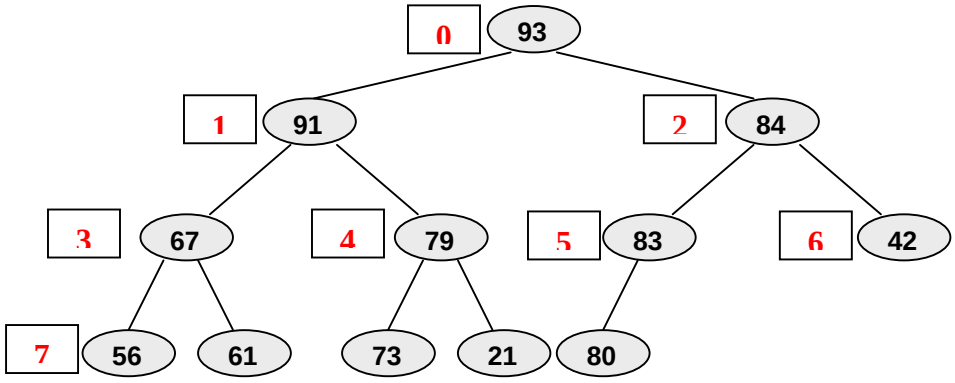
b) If T has a total of 1001 nodes, then _____**500**____ of them must have children.

8. [10 points] Draw the BST that results from inserting the following data values in the order given:

42 17 89 53 72 91 3 88



9. [5 points] Consider the heap given below. If the heap elements were stored in an array, as usual, what element would be stored at the index 7?



10. [10 points] Consider the `BST` and `NodeT` template interfaces given at the bottom of this page. Write the body of the member function `Find()`, which returns true if its parameter occurs in the BST and false otherwise.

```
template <class Data> bool BST<Data>::Find(const Data& toFind) {

   return FindHelper(Root, toFind);
}

template <class Data>
bool BST<Data>::FindHelper(NodeT<Data>* sRoot,
                           const Data& toFind) {

   if (sRoot == NULL) return false;

   Data currentData = sRoot->getData();

   if (currentData == toFind) return true;

   if (toFind < currentData)
      return FindHelper(sRoot->getLeft(), toFind);
   else
      return FindHelper(sRoot->getRight(), toFind);

}
```

```
template <class Data> class NodeT {
private:
   Data         Element;
   NodeT<Data>* Left;
   NodeT<Data>* Right;

public:
   // irrelevant members omitted
   Data getData() const;
   NodeT<Data>* getLeft() const;
   NodeT<Data>* getRight() const;
};
```

```
template <class Data> class BSTreeT {
private:
   NodeT<Data>* Root;

   // irrelevant members omitted
public:
   // irrelevant members omitted
   bool Find(const Data& D);
};
```

11. Consider the `BinaryTreeT` template interface given at the bottom of this page.

```
template <class Data>
void BinaryTreeT<Data>::ClearHelper(NodeT<Data>* sRoot) {

   if (sRoot == NULL) return;

   ClearHelper(sRoot->getLeft());
   delete sRoot;
   ClearHelper(sRoot->getRight());
}

template <class Data> BinaryTreeT<Data>::~BinaryTreeT() {

   ClearHelper(Root);
}
```

a) [2 points] What type of traversal does `ClearHelper()` perform on the tree? **inorder**

b) [8 points] Will the given destructor correctly delete all of the tree nodes? **No.**

If not, write any modifications that are necessary to fix the problem below:

```
template <class Data>
void BinaryTreeT<Data>::ClearHelper(NodeT<Data>* sRoot) {

   if (sRoot == NULL) return;

   ClearHelper(sRoot->getLeft());
   ClearHelper(sRoot->getRight());
   delete sRoot;
}
```

```
template <class Data> class BinaryTreeT {
protected:
   NodeT<Data>* Root;
   NodeT<Data>* Current;
   // . . . irrelevant protected member functions omitted
   void ClearHelper(NodeT<Data>* sRoot);

public:
   BinaryTreeT();
   BinaryTreeT(Data newData);

   // . . . irrelevant public member functions omitted

   ~BinaryTreeT();
};
```