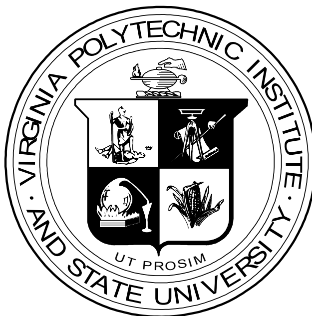


Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. 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 10 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 either failing to return this test, or discussing 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. [9 points] Complete the statement of the theorem below:

   Theorem Suppose that f and g are functions with domain [1, ∞) and that: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$

   Then: (a) f(n) is strictly O(g(n)) if C is **0**.

   (b) f(n) is strictly Ω(g(n)) if C is ∞.

   (c) f(n) is Θ(g(n)) if C is **positive and finite**.

2. [8 points] Use the theorem you completed above to either prove the statement below is true, or that it is false. (Be sure to state your conclusion.)

$$f(n) = n + 3n^2 \log n \text{ is } \Theta\left(n^2\right)$$

$$\lim_{n\to\infty} \frac{n + 3n^2 \log n}{n^2} = \lim_{n\to\infty}\left(\frac{1}{n} + 3\log n\right) = \infty$$

Therefore, the statement is false; in fact, f is strictly $\Omega(n^2)$.

---

3. [10 points] A programmer must choose a data structure to store N records, which will be supplied to the program in random order, and to support random searches on those records. Suppose the programmer decides to use a sorted array. As each element is received, it will be placed in its (currently) proper location in the array.

   a) What is the expected Θ-complexity for putting all N elements into the array? Explain.

   As described, each element must be put into the array in its proper location relative to the previous contents of the array. That means that some of the elements must be shifted; on average half of them would be shifted. So, when the i-th element is inserted, the average number of elements to be shifted would be about (i – 1)/2.

   So the total cost of inserting all N elements would be given by the following formula, which is $\Theta(N^2)$:

$$T(N) = \sum_{i=1}^{N} \frac{i-1}{2} = \frac{1}{2}\left(\frac{n(n+1)}{2}\right) - \frac{1}{2}N$$

   b) If $N = 2^{20}$, estimate the cost of performing 2^{30} searches on the data in the array? Your answer should be a number, perhaps expressed using powers of 2.

   It's a sorted array, so the cost of a single search would be Θ(log N), which is 20. So the total cost of all those searches would be:

$$20 \times 2^{30} = 5 \times 2^{32}$$

4. [12 points] Assuming that each assignment, arithmetic operation, comparison, and array index costs one unit of time, analyze the complexity of the body of the following code fragment that computes an approximation of the number π, and give a simplified exact count complexity function T(N) and state its big-Θcategory:

```
double Pi(unsigned int N) {

   double Approx;
   Approx = 1.0;                                   // 1

   for (unsigned int It = 1; It <= N; It++) {   // 1 before, 2 during, 1 after

      double Term = 1.0 /(2.0 * It + 1);          // 4

      if ( It % 2 == 0 )                          // 2
         Sign = 1.0;                              // 1
      else
         Sign = -1.0;                             // 1

      Approx = Approx + Sign * Term;              // 3
   }

   Approx = 4.0 * Approx;                         // 2
   return Approx;                                 // 1
}
```

The total cost function would be:

$$T(N) = 1 + 1 + \sum_{It=1}^{N} (2 + 4 + 2 + 1 + 3) + 1 + 2 + 1 = 6 + \sum_{It=1}^{N} (12) = 6 + 12N$$

This is Θ(N).

5. [10 points] Referring to the binary tree shown below, write down the values in the order they would be visited if an enumeration was done using:

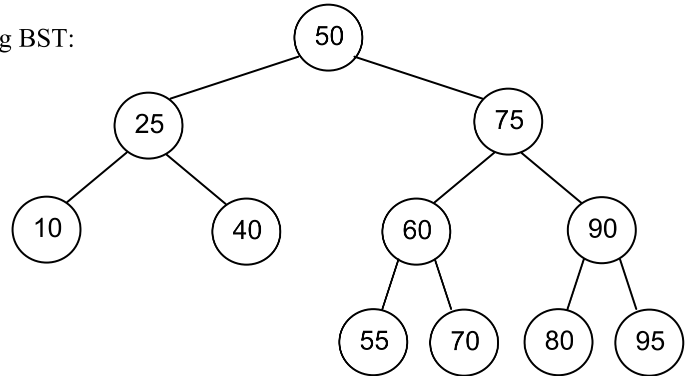
a) a preorder traversal

50 25 10 40 75 60 66 70 90 80 95

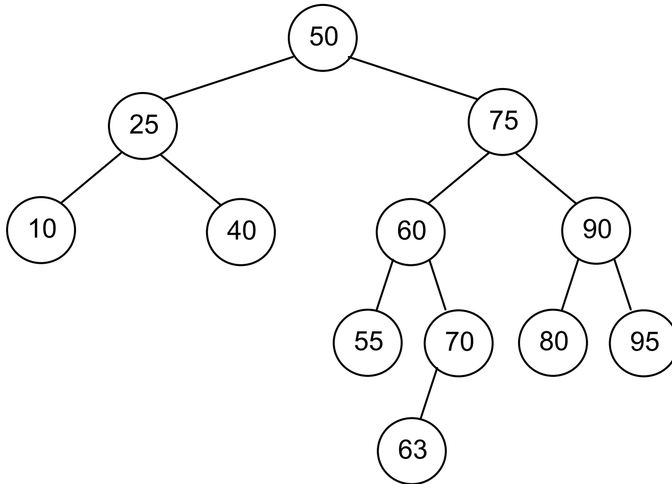
b) a postorder traversal

10 40 25 55 70 60 80 95 90 75 50

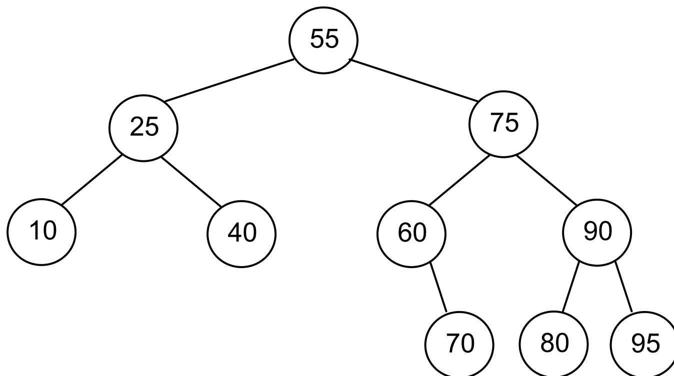
For <u>each</u> of the questions 5 and 6, <u>start</u> with the following BST:



6. [8 points] Draw the resulting BST if 63 is inserted.



7. [8 points] Draw the resulting BST if 50 is deleted.



For question 8, assume the following template declarations for an implementation of a doubly-linked list:

```
// DNodeT.h
//
. . .
template <typename T> class DNodeT {
public:
   T           Element;
   DNodeT<T>* Prev;
   DNodeT<T>* Next;
   // irrelevant members not shown
};
. . .
```

```
// DListT.h
//
. . .
template <typename T> class DListT {
private:
   DNodeT<T>* Head, Tail;    // pointers to first and last data nodes, if any
   DNodeT<T>* Fore, Aft;     // pointers to leading and trailing sentinels
public:
   // irrelevant members not shown
   iterator begin();              // return iterator to first data node (or end())
   iterator end();                //                   one-past-end
   const_iterator begin() const; // return const_iterator objects similarly
   const_iterator end() const;
   ~DListT();                     // destroy all dynamic content of the list
};
. . .
```

8. [10 points] Write an implementation of the copy constructor for the `DListT` template. You may not call any other member functions of the template in your solution.

```
template <typename T> DListT<T>::DListT(const DListT<T>& Source) {

   Head = Tail = Null;
   Fore = new DNodeT<T>();
   Aft  = new DNodeT<T>();

   Fore->Next = Aft;
   Aft->Prev  = Fore;

   DNodeT<T>* Curr = Source.Fore.Next;

   while ( Curr != Source.Aft ) {
      DNodeT<T>* nextNode = new DNodeT<T>( Curr->Element, Aft->Prev, Aft );
      Aft->Prev->Next = nextNode;
      Aft->Prev       = nextNode;
      Curr = Curr->Next;
   }

   if ( Fore->Next != Aft ) {
      Head = Fore->Next;
      Tail = Aft->Prev;
   }
}
```

9. [10 points] Prove: for all $L \geq 0$, if T is a binary tree with L levels, the maximum number of nodes T can have is $2^L - 1$.

Hint: a nonempty binary tree consists of a root node and two subtrees; use Strong Induction.

proof: Let $S = \{ L \geq 0 \mid$ if T is a binary tree with L levels, then T has no more than $2^L – 1$ nodes$\}$.

Suppose that T is a binary tree with 0 levels. Then T must be empty, and since $2^0 – 1 = 0$, 0 is in S.

Suppose that for some $K \geq 0$, for every L such that $0 \leq L \leq K$, L is in S. That is, for every L from 0 to K, if T is a binary tree with L levels, then T has no more than $2^L – 1$ nodes.

We must show that if T is a binary tree with L + 1 nodes, then T can have no more than $2^{L+1} – 1$ nodes.

Suppose that T is a binary tree with L + 1 nodes. Now T is not empty, so it consists of a root node and two (possibly empty) subtrees. Now each of the subtrees has fewer levels than T; therefore, each of the subtrees has no more than $2^L – 1$ nodes.

Hence, the total number of nodes in T is no more than $1 + 2^L – 1 + 2^L – 1 = 2*2^L – 1 = 2^{L+1} – 1$. Therefore, L + 1 is in S.

So, by mathematical induction, $S = \{L \geq 0\}$.

For question 10, assume the following template declarations for an implementation of a binary tree:

```
template <typename T> class BinNodeT {
public:
   T             Element;
   BinNodeT<T>* Left;
   BinNodeT<T>* Right;
   // irrelevant members not shown
};
```

```
template <typename T> class BST {
protected:
   BinNodeT<T>* Root;
   // irrelevant members not shown

public:
   // irrelevant members not shown
};
```

10. [15 points] Write an implementation for the new `BST` member function described below. Your implementation should not need to call any other template member functions, except for any helper member functions you may wish to write.

```
// Range() uses a (modified) inorder traversal pattern, and prints all the
// values it finds that are strictly between the parameters Min and Max to cout.
// The function should not visit any tree nodes unnecessarily.
//
template <typename T> void BST<T>::Range(const T& Min, const T& Max) const {

   rangeHelper( Root, Min, Max );
}

template <typename T>
void BST<T>::rangeHelper(BinNodeT<T>* sRoot, const T& Min, const T& Max) const {

   if ( sRoot == NULL ) return;      // done, if we're not at a node

   if ( Min < sRoot->Element )       // no need to check left subtree if
                                     //    this element is too small
      rangeHelper( sRoot->Left, Min, Max);

   // check whether current element is in the range
   if ( Min < sRoot->Element && sRoot->Element < Max )
      cout << sRoot->Element << endl;

   if ( sRoot->Element < Max )       // no need to check left subtree if
                                     //    this element is too large
      rangeHelper( sRoot->Left, Min, Max);
}
```