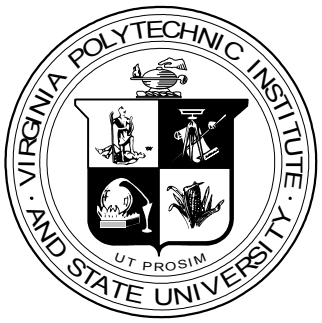


Instructions:

- 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. [6 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) if C = 0 then f(n) is **O**(g(n)).

(b) if C = ∞ then f(n) is **Ω**(g(n)).

(c) if 0 < C < ∞ then f(n) is **Θ**(g(n)).

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.)

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

First we need to find the relevant limit:

$$\lim_{n\to\infty} \frac{n^2 + 3n^2 \log n}{n^3} = \lim_{n\to\infty} \left(\frac{n^2}{n^3} + \frac{3n^2 \log n}{n^3} \right) = \lim_{n\to\infty} \left(\frac{1}{n} + \frac{3\log n}{n} \right) = 0 + \lim_{n\to\infty} \frac{\frac{3}{n \ln 3}}{1} = 0$$

So, from the theorem, the statement IS true.

2. [10 points] A programmer must choose a data structure to store N elements, which will be supplied to the program in random order. Give a big-Θ estimate for the number of operations required to create the structure if the programmer uses the following data structure and otherwise making intelligent decisions regarding efficiency:

   a) a doubly-linked list, inserting the N elements, in the order supplied, as they are supplied.

   Inserting each element, in the order supplied, is Θ(1), since they can simply be appended. Since there are N elements, the total cost is Θ(N).

   b) a sorted array, inserting the N elements as they are supplied and then sorting the array.

   Again, inserting each element would be Θ(1), and so inserting all N of them would be Θ(N). Sorting the array, efficiently, would be Θ(N log N) , so the total cost is Θ(N log N).

---

3. [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):

```
double Pi(unsigned int N) {

   double Approx;
   Approx = 1.0;                                        // 1

   for (unsigned int It = 1; It <= N; It++) {           // 1 before, 2 each pass,
                                                        // 1 to exit

         if ( It % 2 == 0 )                             // 2*
            Approx = Approx + 1.0 /(2.0 * It + 1);      // 5, if done
         else
            Approx = Approx - 1.0 /(2.0 * It + 1);      // 5, if done
   }

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

Given the statement analysis above, the total complexity is:

$$T(N) = 1 + 1 + \sum_{It=1}^{N} \left(2 + 2 + \max(5,5)\right) + 1 + 2 + 1 = 6 + \sum_{It=1}^{N} \left(9\right) = 6 + 9N$$

*** Consider this: what would you say if the statement was "if (foo)" where foo was a Boolean variable? You could argue that this should be 3 operations, in which case you should also argue that the for loop test costs 2 operations, not 1. I'd accept that analysis as well.**

4. [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:

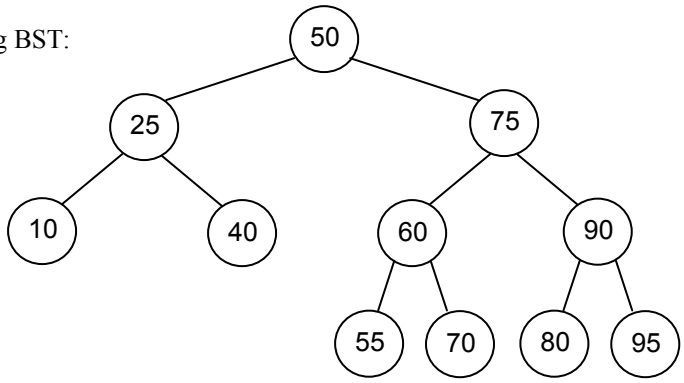
   a) an inorder traversal

   10 25 40 50 55 60 70 75 80 90 95

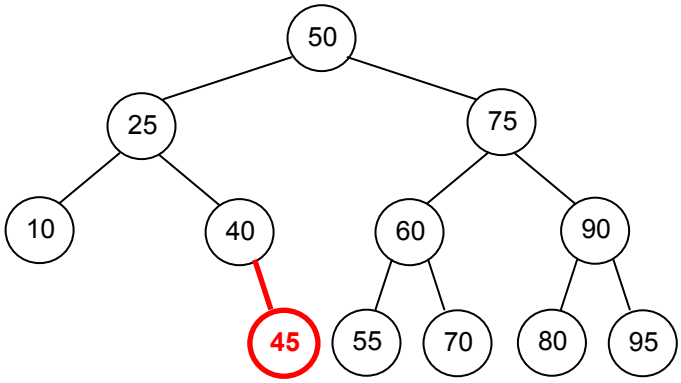
   b) a postorder traversal

   10 40 25 55 70 60 80 95 90 75 50

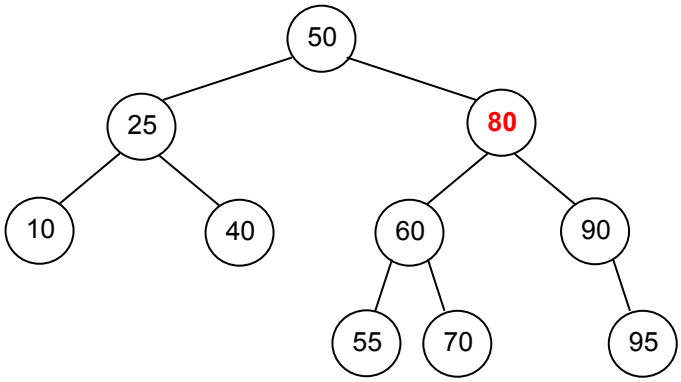
For each of the questions 5 and 6, start with the following BST:



5. [8 points] Draw the resulting BST if 45 is inserted.



6. [8 points] Draw the resulting BST if 75 is deleted.



For questions 7 and 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
};
. . .
```

7. [8 points] Write an implementation of the destructor for the `DListT` template. You may not call any other member functions of the template in your solution.

```
template <typename T> DListT<T>::~DListT() {

   DNodeT<T>* Current = Head;

   while ( Current != Aft ) {

      Head = Head->Next;
      delete Current;
      Current = Head;
   }

   delete Fore;
   delete Aft;
}
```

8. [10 points] Write an efficient implementation for the following new member function for the `DListT` template. Note that an efficient implementation will not allocate any new nodes or delete any existing ones..

```
// mtfFind() searches the DListT object for the first occurrence of a data
// object that is equal to the parameter Target.  If no match is found, the list
// is not modified and the iterator value end() is returned.  If a matching
// element is found, that element is moved to the head of the list (if necessary),
// and an iterator to that element is returned.
//
// For example, if mtfFind(17) is called on the list {35, 89, 23, 17, 45, 9},
// the resulting list would be {17, 35, 89, 23, 45, 9} and an iterator to 17
// would be returned.
//
template <typename T>
typename DListT<T>::iterator DListT<T>::mtfFind(const T& Target) {

   if ( Head == NULL )
      return end();

   DNodeT<T>* Current = Head;

   while ( Current != Aft ) {

      if ( Target == Current->Element ) {

         if ( Current != Head ) {
            Current->Prev->Next = Current->Next;  // reset surrounding nodes
            Current->Next->Prev = Current->Prev;

            if ( Current == Tail )       // reset Tail pointer if necessary
               Tail = Current->Prev;

            Current->Next = Head;        // attach node to old head node,
            Current->Prev = Fore;        //   and Fore sentinel

            Fore->Next = Current;        // attach Fore sentinel to node
            Head->Prev = Current;        // attach old head node to node
            Head       = Current;        // attach Head pointer to node
         }
         return iterator( Current );
      }

      Current = Current->Next;
   }

   return end();
}
```

For question 9, 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 BinTreeT {
protected:
   BinNodeT<T>* Root;
   // irrelevant members not shown

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

9. [10 points] Write an implementation for the new `BinTreeT` member function below. Your implementation should not need to call any other template member functions, except for any helper member functions you may wish to write.

```
// PreOrderRepMax makes a preorder traversal of the binary tree.  When it
// "visits" an internal node, it replaces the value stored there with the larger
// of the of the values stored in the children of that node. Leaf nodes are
// unchanged.
template <typename T> void BinTreeT<T>::PreOrderRepMax() const {

   RepMaxHelper( Root );
}

template <typename T> void BinTreeT<T>::RepMaxHelper(BinNodeT<T>* sRoot) const {

   if ( sRoot == NULL ) return;

   if ( sRoot->Left == NULL && sRoot->Right == NULL )
      return;

   if ( sRoot->Left == NULL )
      sRoot->Element = sRoot->Right->Element;
   else if ( sRoot->Right == NULL )
      sRoot->Element = sRoot->Left->Element;
   else {
      if ( sRoot->Left->Element >= sRoot->Right->Element )
         sRoot->Element = sRoot->Right->Element;
      else
         sRoot->Element = sRoot->Left->Element;
   }

   RepMaxHelper( sRoot->Left );

   RepMaxHelper( sRoot->Right );
}
```

10. [10 points] Let T be a binary tree, and number the levels of the tree starting at zero, so we would say the root node is in level 0. Prove: for all $n \geq 0$, the maximum number of nodes T can have in level n is equal to 2^n.

proof: Let T be a binary tree. Level 0 contains, at most, the root node, so the maximum number of nodes that can occur in level 0 would indeed be 1 (which is 2^0).

Assume that for some integer $k \geq 0$, the maximum number of nodes in level k is 2^k.

Consider level k+1 of T. Each node in level k+1 must be the child of a node in level k. Each node in a binary tree can have no more than 2 children. Therefore, the maximum number of nodes in level k+1 is exactly twice the number of nodes in level k. Since the latter is 2^k, the maximum number of nodes T can have in level k+1 would be $2 \times 2^k = 2^{k+1}$.