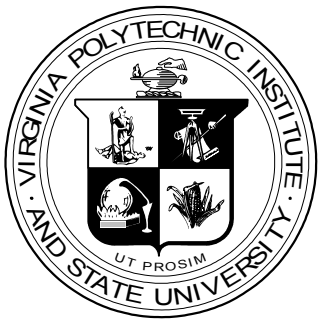


Instructions:

- 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. [14 points] For each part, either prove the statement is true, or that it is false. You may use any of the definitions or theorems covered in class, but the recommended approach is to use limits.

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

   A little algebra and a limit theorem or two show that:

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

   This implies that f(n) is $\Theta(n^2 \log n)$.

   b) $g(n) = \dfrac{n^2 + 2}{3n^2 + 4}$ is $\Theta(n)$

   Similarly:

$$\lim_{n\to\infty} \frac{\frac{n^2 + 2}{3n^2 + 4}}{n} = \lim_{n\to\infty} \frac{n^2 + 2}{3n^3 + 4n} = \lim_{n\to\infty} \frac{2n}{6n^2 + 4} = \lim_{n\to\infty} \frac{2}{12n} = 0$$

   However, since the limit is zero, this implies that g(n) is not $\Theta(n)$.

---

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 ascending order, as they are supplied.

   Inserting the i-th element requires a search, which will on average cost (i-1)/2 comparisons, and so inserting N elements would be $\Theta(N^2)$.

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

   Inserting the i-th element would cost only 1 operation, since it can be placed at the end. Sorting efficiently, using quicksort for example, would cost $\Theta(n \log n)$.

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, Sign;
   Approx = 1.0;                                          // 1
   Sign   = 1.0;                                          // 1

   for (int It = 1; It <= N; It++) {                      // 1 before loop,
                                                          // 2 per pass,
                                                          // 1 on exit
         Sign = -1.0 * Sign;                              // 2 per pass
         Approx = Approx + Sign * 1.0 /(2.0 * It + 1);    // 6 per pass
   }

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

So the complexity function would be:

$$T(n) = 1+1+1+\sum_{It=1}^{N}(2+2+6)+1+2 = 6+\sum_{It=1}^{N}10 = 6+10N$$

4. [8 points] Referring to the binary tree shown on the next page, write down the values in the order they would be visited if an enumeration was done using:

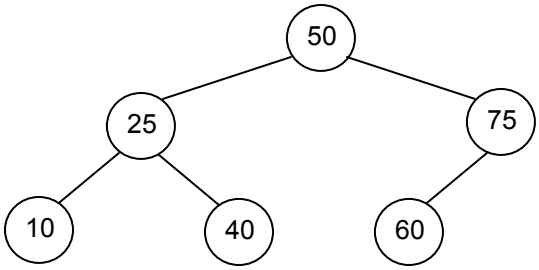
   a) an inorder traversal

   10 25 40 50 60 75

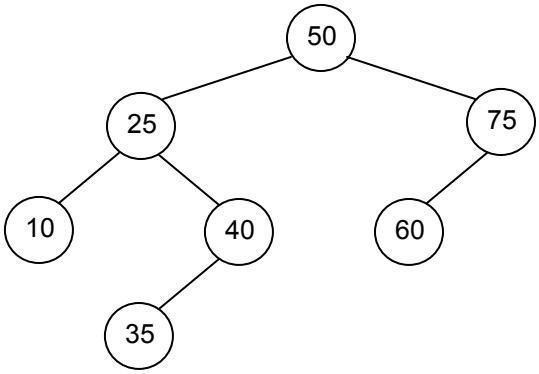
   b) a postorder traversal

   10 40 25 60 75 50

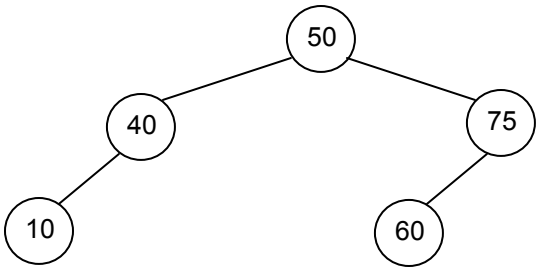
For each of the questions 5 through 7, start with the following BST:



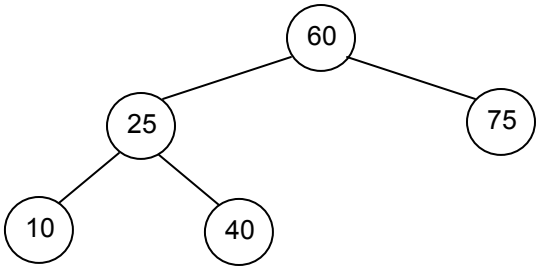
5. [8 points] Draw the resulting BST if 35 is inserted.



6. [8 points] Draw the resulting BST if 25 is deleted.



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



For questions 8 and 9, 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;

   DNodeT();
   DNodeT(const T& Elem, DNodeT<T>* P = NULL, DNodeT<T>* N = NULL);
   ~DNodeT();
};
. . .
```

```
// DListT.h
//
. . .
template <typename T> class DListT {
      friend class Javert;
private:
   DNodeT<T>* Head;
   DNodeT<T>* Tail;

public:
   // irrelevant members not shown
   T*   Insert(iterator It, const T& Elem);  // insert before target of iterator,
                                             //   if any; if iterator is NULL,
                                             //   append value to list
};
. . .
```

8. [10 points] Write an implementation for a copy constructor for the `DListT` template. Your solution should not call any member functions of the linked list template (although the node functions may be used).

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

   Head = Tail = NULL;
   DListT<T>::const_iterator Current = Source.begin();

   while ( Current != Source.end() ) {

      DNodeT<T>* Next = new DNodeT<T>(Current->Element);
      if ( Head == NULL ) {
         Head = Tail = Next;
      }
      else {
         Tail->Next = Next;
         Next->Prev = Tail;
         Tail = Next;
      }

      Current++;
   }
}
```

9. [10 points] Write an implementation for the following member function for the `DListT` template. You should call any useful other member functions of either template that will simplify your solution.

```
// Reverse() modifies the list object so the data values it stores are in
// reverse order.  You may assume the data type T supports the usual relational
// operators, assignment, etc.
//
template <typename T> void DListT<T>::Reverse() {

   DNodeT<T>* Moving = Head;
   DNodeT<T>* tailMarker = Head;

   while ( Head != Tail ) {

         Head = Head->Next;

         Moving->Prev = Tail;
         Moving->Next = Tail->Next;
         Tail->Next = Moving;

      Moving = Head;
   }

   Tail = tailMarker;
}
```

10. [12 points] Consider the recursive function definition: $S(n) = \begin{cases} 1 & n = 1 \\ S(n-1) + n & n > 1 \end{cases}$

Use induction to <u>prove</u> that for all n ≥ 1, $S(n) = \frac{n(n+1)}{2}$.

Let: $T = \left\{ n \geq 1 \middle| S(n) = \frac{n(n+1)}{2} \right\}$

Basis: if n = 1 then S(1) = 1 by definition, and $\frac{n(n+1)}{2} = 1$. **So, 1 is in T.**

Inductive hypothesis: assume that for some k ≥ 1, $S(k) = \frac{k(k+1)}{2}$

Then, $S(k+1) = S(k) + k + 1 = \frac{k(k+1)}{2} + (k+1) = (k+1)\left(\frac{k}{2} + 1\right) = (k+1)\frac{(k+2)}{2} = \frac{(k+1)(k+2)}{2}$

Thus, k+1 is in T.

So, by the Principle of Induction, every integer n ≥1 is in T.