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 9 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. [10 points] Circle TRUE or FALSE according to whether the statement is true or false:

a) $1000 - 7n + 2n^2$ is $\Theta(n)$ TRUE [✓ FALSE]

b) $1000 - 7n + 2n^2$ is $\Omega(n)$ [✓ TRUE] FALSE

c) $1000 - 7n + 2n^2$ is $O(n^3)$ [✓ TRUE] FALSE

d) $17n \log n + n^2$ is $\Theta(n^2)$ [✓ TRUE] FALSE

e) $\sum_{i=1}^{n} \left(7 + \sum_{k=1}^{i} k \right)$ is $\Theta(n^3)$ [✓ TRUE] FALSE

2. [15 points] The following algorithm prints an integer as a function of an input integer n, assumed to be positive.

```
read n;                // line 1:     ______1
x = 7 * n;             //      2:     ______2
y = x + 5 * n;         //      3:     ______3
while (x + y > 0) {    //      4:     ______2
    x = x - 3;         //      5:     ______2
    y = y + 1;         //      6:     ______2
}
print x * y;           //      7:     ______2
```

We charge for operations at a rate of one arbitrary time unit for each assignment, arithmetic operation, comparison, integer input, and integer output. Assume that n is a positive integer.

a) Indicate the time cost of each of the numbered lines in the code fragment above.

b) Compute the total time complexity function T(n) for the above algorithm.

T(n) = 1 + 2 + 3 + [Sum(2 + 2 + 2) from 1 to 19n/2] + 2 + 2 = 10 + 57n

c) Determine the asymptotic (big-Θ) of T(n) as a simple function of n.

The function above is clearly Theta(n).

3. [10 points] A programmer must provide a data structure to store N elements, which will be supplied to the program in random order. Give a big-Θ bound for the number of operations required to create the structure if the programmer uses:

   a) a sorted array of dimension N, inserting the N elements at the appropriate index as they are supplied.

   Inserting the k-th element will, on average, require searching a list of k-1 elements, taking log(k-1) operations, and then shifting half of the elements already in the array. So the cost of inserting elements 1 through N is dominated by the shifting and would be about:

$$T(N) = \sum_{k=1}^{N} \frac{k-1}{2} \approx \frac{1}{4} N^2 \in \Theta(N^2)$$

   b) an AVL tree, inserting the N elements as they are supplied.

   Inserting the k-th element will, on average, have a search cost of log(k-1) and then a constant cost for creating and inserting the new node and rebalancing if necessary. So the total cost would be about:

$$T(N) = \sum_{k=1}^{N} \log(k) \approx N \log(N) \in \Theta(N \log N)$$

---

4. [12 points] Consider the AVL tree structure.

   a) What property, relating to the tree's balance, does an AVL tree guarantee at each internal node?

   The heights of the left and right subtrees of the node differ by no more than 1.

   b) When a value is inserted or deleted in an AVL tree, it may be necessary to rearrange part of the tree to restore the AVL balance property. The rearrangement is accomplished by performing one or more rotations. In big-Θ terms, how many operations are required to perform a single rotation?

   Each rotation requires resetting a small number of pointers, an a small number of balance factors. This is bounded by a small constant, and so it is Theta(1).

   c) Theoretically, if a binary tree has N nodes, what is the minimum number of levels that the tree could have?

   Either from a theorem in the notes, or from examples, it is quickly obvious that if a binary tree contains N nodes then the number of levels must be at least 1 + log N.

5. [16 points] Consider the heap structure.

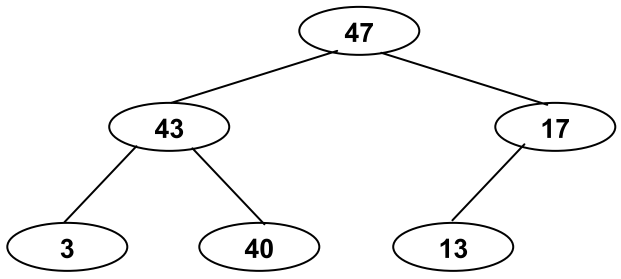
a) Is a heap containing N nodes guaranteed to have the minimum possible height? If so, why? If not, draw a max-heap that does not have minimum height.

Yes. In a complete tree, all the levels except (possibly) the bottom one are full. Since that leaves no "gaps" in the upper levels to which leaves could be moved, it would be impossible to make the tree shorter by moving nodes up.

b) What property of a heap makes it possible to easily store it in an array?

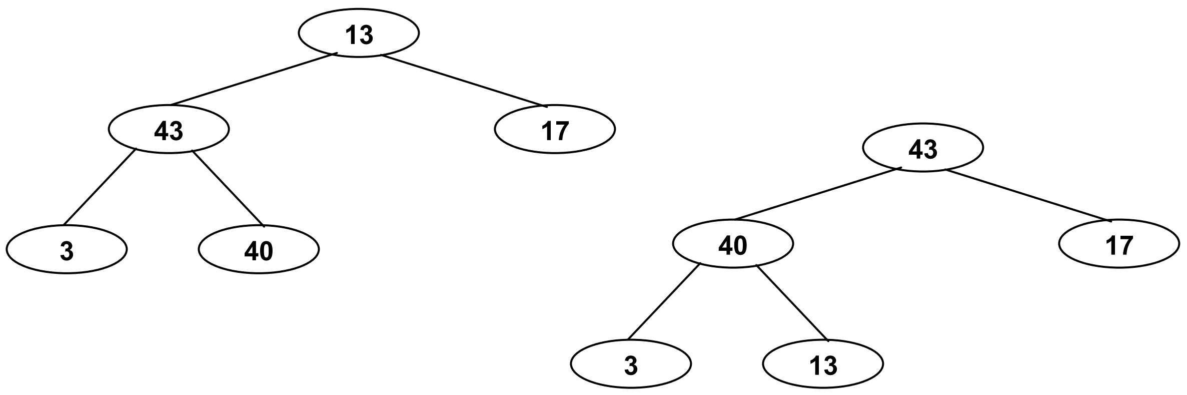
Again, the fact that it is complete. That means that when the left and right children are stored using the given index formulas, there will not be any "gaps" in the array due to unused cells corresponding to missing child nodes. The presence of such "gaps" would require that some sort of marker field also be stored to keep track of whether each cell was actually used.

c) Show the array representation of the max-heap shown below:



0	1	2	3	4	5
47	43	17	3	40	13

d) Draw (as a tree, not an array) the resulting tree if the root of the max-heap shown above is deleted:

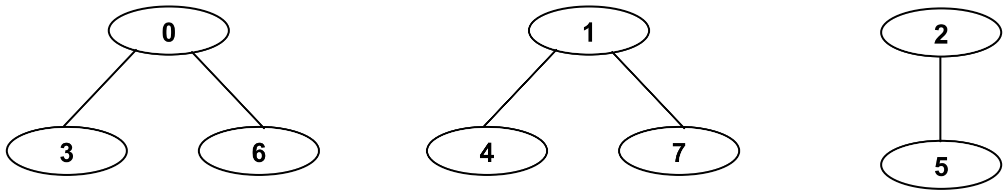


6. The relation on the set {0, 1, 2, 3, 4, 5, 6, 7} defined by "x ~ y if and only if x % 3 = y % 3" is an equivalence relation with three equivalence classes:

[0] = {0, 3, 6} [1] = {1, 4, 7} [2] = {2, 5}

a) [5 points] Show how this equivalence relation should be represented, using the notion of a general tree structure.

As discussed in class, the operations needed for an equivalence relation can all be supported via nodes storing only a data value and a parent pointer. So, the situation above could be represented as:



b) [4 points] Write a possible template interface for the tree node that would be used.

```
template <typename T> class pNode {
public:
   T         Data;
   pNode<T>* Parent;
   pNode();
   pNode(const T& D, pNode<T>* Par);
};
```

7. [6 points] Using the relationship between big-Θ and limits, prove that T(N) = 7N + N log N is Θ(N log N).

From a theorem in the notes, if the limit of f(N)/g(N) is a positive constant, then f(N) is Theta(g(N)). So:

$$\lim_{N\to\infty}\frac{7N+N\log N}{N\log N}=\lim_{N\to\infty}\left(\frac{7}{\log N}+1\right)=1$$

8. [10 points] Consider the LinkedList and NodeT template interfaces given below:

```
template <class Data> class NodeT {
public:
   Data          Element;
   NodeT<Data>* Next;

   NodeT(Data Elem = Data(),
         NodeT<Data>* N = NULL);
};
```

```
template <class Data> class LinkedList {
private:
   NodeT<Data>* Head;
   NodeT<Data>* Tail;
public:
   bool  Prefix(const Data&);
   bool  Append(const Data&);
   bool  Delete(const Data&);
   LinkedList SubList(const Data&);
   Data  Find(const Data&);
   ~LinkedList();
};
```

Write the body of the public function `Delete()`, which searches the list for a node containing the specified data element and deletes that node, returning the data element, or a default `Data` object if there is no such node. Your implementation must conform to the template interfaces given above. You may assume that the implementation of `Data` provides overloads for any relational operators you need.

```
template <class Data> Data LinkedList<Data>::Delete(const Data& Target)
{

   if ( Head == NULL ) return Data();   // handle empty list

   NodeT<Data>* Curr = Head;
   if ( Target == Curr->Element ) {     // head node holds target
      Head = Head->Next;
      if ( Head == NULL )
         Tail = NULL;
      Data Temp = Curr->Element;
      delete Curr;
      return Temp;
   }

   while ( Curr->Next != NULL && Target != Curr->Next->Element )
      Curr = Curr->Next;

   if ( Curr->Next == NULL ) return Data();  // target not found

   Data Temp = Curr->Next->Element;     // removing node, not head
   NodeT<Data>* toKill = Curr->Next;
   Curr->Next = Curr->Next->Next;
   if ( Curr->Next = Tail )
      Tail = Curr;
   delete toKill;
   return Temp;
}
```

9. [12 points] Consider the partial BST and binary node template interfaces given below:

```
template <typename T> class BinNodeT {
public:
   Data         Element;
   BinNodeT<T>* Left;
   BinNodeT<T>* Right;

   BinNodeT();
   BinNodeT(const T& newData,
            BinNodeT<T>* newLeft,
            BinNodeT<T>* newRight);
   ~BinNodeT();
};
```

```
template <typename T> class BST {
protected:
   BinNodeT<T> *Root;
   . . .

public:
   BST();
   T* Insert(const T& Elem);
   T* Find(const T& D, int& Level);
   T  Delete(const T& D);
   ~BST();
   . . .
};
```

Write a BST member function subTree() which conforms to the interface below. It is OK to use a recursive helper function as long as you show the implementation of that as well.

```
// If the value Target occurs in the tree, the function copies the subtree whose
// root stores Target and returns the copy.  If the value does not occur in the
// tree, the function returns an empty BST.  The implementation may assume that
// the value Target does not occur more than once.
//
template <typename T> BST<T> BST<T>::subTree(const T& Target) {

   BST<T> subTree;
   if ( Root == NULL ) return subTree;   // handle empty tree

   BinNodeT<T>* Curr = Root;
   while ( Curr != NULL && Target != Curr->Element ) {
      if ( Target < Curr->Element )
         Curr = Curr->Left;
      Else
         Curr = Curr->Right;
   }

   if ( Curr == NULL ) return subTree;    // target not found

   copyHelper(subTree.Root, Curr);
   return subTree;
}

template <typename T> void copyHelper(BinNodeT<T>*& copyRoot,
                                      const BinNodeT<T>* const sRoot) {

   if ( sRoot == NULL ) return;
   copyRoot = new BinNodeT<T>(sRoot->Element);
   copyHelper(copyRoot->Left, sRoot->Left);
   copyHelper(copyRoot->Right, sRoot->Right);
}
```