CSE 101
Midterm 2 Review Problems
Solutions

1. Rank the following functions from lowest to highest asymptotic growth rate.

   1) 2^n
   2) $n \ln(n)$
   3) n
   4) $2^{\ln(n)}$
   5) ln(ln(n))
   6) $n\sqrt{n}$
   7) n^2
   8) $\ln(n^2)$
   9) $\sqrt{n}$

   Write your answer as a permutation of the set $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, giving the corresponding line numbers of the above functions in the required order (left to right, slowest growing function to fastest growing function.) No justifications are required.

   Solution: 5 8 9 4 3 2 6 7 1

2. Consider the List ADT from pa5 but *without* the `cleanup()` function. Write a C++ client function with heading

```
void RemoveDuplicates(List& L)
```

that does the same thing as `cleanup()`, except that it does not matter where the cursor ends up. In other words, the call `RemoveDuplicates(L)` will alter List L so that it contains only the first occurrence of each of its data items. To do this, you may use all ADT operations in List.h *except* `cleanup()`.

Solution:

```
void RemoveDuplicates(List& L){

   int p, x, y;

   L.moveFront();
   p = 0;
   while( p<L.length() ){
      x = L.moveNext();
      while(L.position()<L.length()){
         y = L.moveNext();
         if( y==x ){
            L.eraseBefore();
         }
      }
      p++;
      while(L.position()>p){
         L.movePrev();
      }
   }
}
```

Alternate Solution:

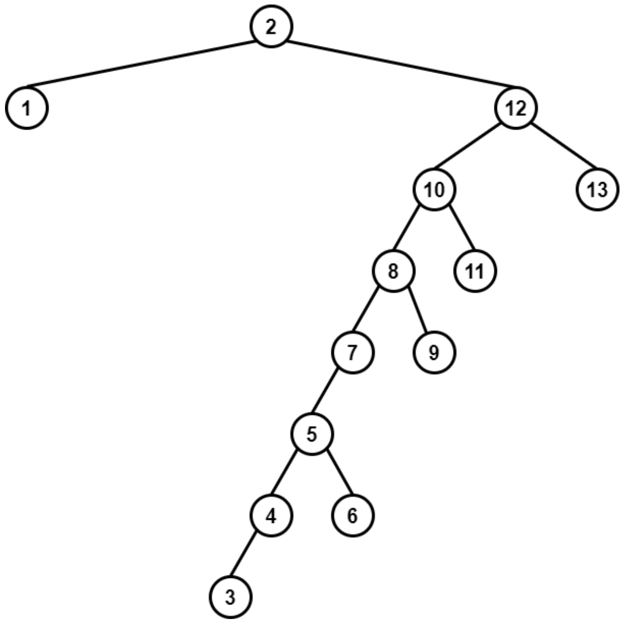
```
void RemoveDuplicates2(List& L){

   int p, x;

   L.moveFront();
   p = 0;
   while( p<L.length() ){
      x = L.moveNext();
      p = L.findNext(x);
      while(p>=0){
         L.eraseBefore();
         p = L.findNext(x);
      }
      L.moveFront();
      p = L.findNext(x);
   }
}
```

3. Let T be a Binary Search Tree containing the keys $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$. Suppose that a **pre-order tree walk** prints the keys in order: $2, 1, 12, 10, 8, 7, 5, 4, 3, 6, 9, 11, 13$, and that a post-order tree walk prints the keys in order: $1, 3, 4, 6, 5, 7, 9, 8, 11, 10, 13, 12, 2$. Determine the structure of T. (Note: only one of the two tree walks is really necessary since each of them uniquely determines the structure of T.) Present your solution either by drawing a picture of the tree, or by constructing a table giving the parent of each Node.

Solution1 (Picture):



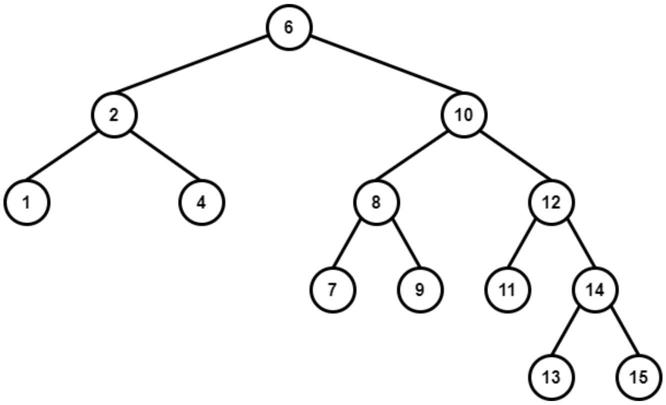
Solution2 (Table):

Node	**Parent**
1	**2**
2	**Nil**
3	**4**
4	**5**
5	**7**
6	**5**
7	**8**
8	**10**
9	**8**
10	**12**
11	**10**
12	**2**
13	**12**

4. Use the `TreeInsert()` algorithm to insert the following keys: 6, 2, 1, 4, 10, 8, 7, 9, 12, 11, 14, 13, 15 (in order) into an initially empty BST.

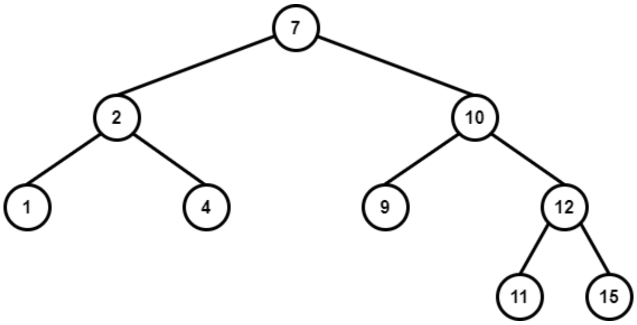
   a. (10 Points) Draw the resulting BST

   Solution:



   b. (10 Points) Use the `Delete()` algorithm to delete the following keys: 8, 6, 13, 14 (in order) from the BST you drew in part (a), then draw the resulting tree.

   Solution:



5. Suppose we alter the List ADT from pa5 by doing

```
typedef char ListElement;
```

at the beginning of `List.h`, making it a list of `char` instead of `int`. Assume a List L consists entirely of parenthesis characters `'('` and `')'`. The List L is called a *Well Formed Formula* (WFF) iff all parentheses can be matched in pairs (open and close). For instance `"(()(()))"` and `"()()(())"` are WFFs, while `"(()()"` and `"(()))"` are not. The empty List is considered to be a WFF. Write a client function with heading

```
bool isWFF(List L)
```

that returns `true` or `false`, according to whether `L` is or is not a WFF. (Hint: search for adjacent matching pairs and delete them. If `L` becomes empty, then return true.)

Solution:

```
bool isWFF(List L){

   int p;

   // delete matching pairs
   L.moveFront();
   while( L.length()>0 ){

      p = L.findNext(')');

      // p==-1 if and only if ')' was not found. p==1 if and only if
      // ')' was found, but has no matching '(' on its left. In both
      // cases we break since no matching pair can be deleted. Note
      // that p==0 is not possible from the specs of findNext().
      if( p<2 ){
         break;
      }

      // delete a matching pair "()"
      L.eraseBefore();  // delete ')'
      L.eraseBefore();  // delete '('
   }

   return ( L.length()==0 );
}
```