CSE 101
Midterm 2 Review Problems
Solutions

1. Use limits to show that the following statements are true.

   a. $n^2 = \omega\left(n\sqrt{n}\right)$

   Solution: $\frac{n^2}{n\sqrt{n}} = n^{1/2} \longrightarrow \infty$ as $n \to \infty$.

   b. $2^n = o(3^n)$

   Solution: $\frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n \longrightarrow 0$ as $n \to \infty$, since $\frac{2}{3} < 1$.

   c. $3^{(2^n)} = o\left(2^{(3^n)}\right)$ Hint: use the result of part (b)

   Solution: First take the natural log of the fraction $\frac{3^{2^n}}{2^{3^n}}$ to get

   $$\ln\left(\frac{3^{2^n}}{2^{3^n}}\right) = 2^n \ln(3) - 3^n \ln(2) = 3^n\left(\frac{2^n}{3^n}\ln(3) - \ln(2)\right) \longrightarrow -\infty.$$

   Now take the exponential of both sides of this limit to get

   $$\lim_{n\to\infty}\left(\frac{3^{2^n}}{2^{3^n}}\right) = e^{-\infty} = 0.$$

   d. If $h(n) = o\big(f(n)\big)$, then $f(n) + h(n) = \Theta\big(f(n)\big)$

   Solution: We have $\lim_{n\to\infty}\left(\frac{h(n)}{f(n)}\right) = 0$ since $h(n) = o\big(f(n)\big)$. Therefore

   $$\lim_{n\to\infty}\frac{f(n)+h(n)}{f(n)} = \lim_{n\to\infty}\left(1 + \frac{h(n)}{f(n)}\right) = 1 + 0 = 1,$$

   and hence $f(n) + h(n) = \Theta\big(f(n)\big)$.

2. 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) $\sqrt{n}$

    Solution: 5 8 9 4 3 2 6 7 1

3. (20 Points) Consider the integer List ADT from pa6. Write a C++ client function with the heading

```
List Replace(List L, int x, int y)
```

that returns a new List containing the same elements as L, in the same order, but with all occurances of the target x replaced by y. For instance, if $L = (3, 2, 9, 2, 5, 7, 2, 6, 4, 2, 7, 8, 2, 9)$, $x = 2$ and $y = 1$, then the returned List would be $(3, 1, 9, 1, 5, 7, 1, 6, 4, 1, 7, 8, 1, 9)$. The returned List will have its cursor placed at its back. If L does not contain the element x, then the returned List will be a copy of L. This function has no preconditions.

Solution1:

```
List Replace(List L, int x, int y){

   List R = L;
   R.moveFront();
   int p = R.findNext(x); // find first occurance of x

   while( p>0 ){
      R.setBefore(y);     // overwrite it with y
      p = R.findNext(x);  // find next occurance of x
   }
   return R;
}
```

Solution2:

```
List Replace(List L, int x, int y){

   List R = L;
   R.moveFront();
   int p = R.findNext(x); // find first occurance of x

   while( p>0 ){
      R.eraseBefore();    // erase x
      R.insertBefore(y);  // replace it by y
      p = R.findNext(x);  // find next occurance of x
   }
   return R;
}
```

Solution3:

```
List Replace(List L, int x, int y){

   int z;
   List R;
   L.moveFront();

   while( L.position()<L.length() ){
      z = L.moveNext();
      if( z==x ) R.insertBefore(y);
      else       R.insertBefore(z);
   }
   return R;
}
```

4. (20 Points) Consider the integer List ADT from pa6 again. Let us say that two Lists A and B are *similar* iff they contain the same elements with the same frequencies (though possibly in different orders). For instance, the Lists $A = (1, 2, 1, 1, 2)$ and $B = (2, 1, 2, 1, 1)$ are similar, since they both contain three $1's$ and two $2's$. On the other hand, Lists $A = (1, 2, 1, 1, 2)$ and $C = (1, 1, 2, 1, 1)$ are not similar since C contains four $1's$ and one 2. Note that similar Lists necessarily have the same length, and that two empty Lists are deemed to be similar. Write a C++ client function with heading

```
bool Similar(List A, List B)
```

that returns `true` if A and B are similar Lists, and `false` otherwise. This function has no preconditions.

Solution:

```
bool Similar(List A, List B){

   int x;
   bool is_similar = ( A.length()==B.length() );

   A.moveFront();
   while( is_similar && A.position()<A.length() ){

      x = A.moveNext();
      B.moveFront();
      is_similar = ( B.findNext(x)>0 );
      if( is_similar ){      // if we found x in B
         B.eraseBefore();    // erase x from B
      }else{                 // otherwise
         break;              // stop checking
      }
   }

   return is_similar;
}
```

Remark:
Note that the above code alters the input list B by calling eraseBefore(). Even if the specifications had called for the input lists to be unchanged, this would be fine, since calling this function entails an implicit call to the copy constructor. The changes are only made to the local copy.

5. Suppose we alter the List ADT from pa6 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)
```

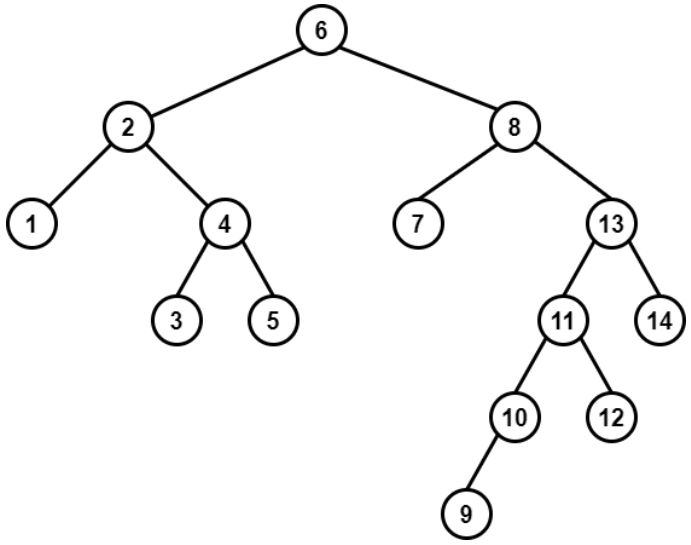
that returns `true` or `false`, according to whether `L` is or is not a WFF. (Hint: search for adjacent matching pairs and delete them. If `L` becomes empty, then return true.)

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 );
}
```

6. (20 Points) Let T be a BST containing the keys $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$. Suppose that a **post-order tree walk** prints the keys in order: $1, 3, 5, 4, 2, 7, 9, 10, 12, 11, 14, 13, 8, 6$. Determine 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):



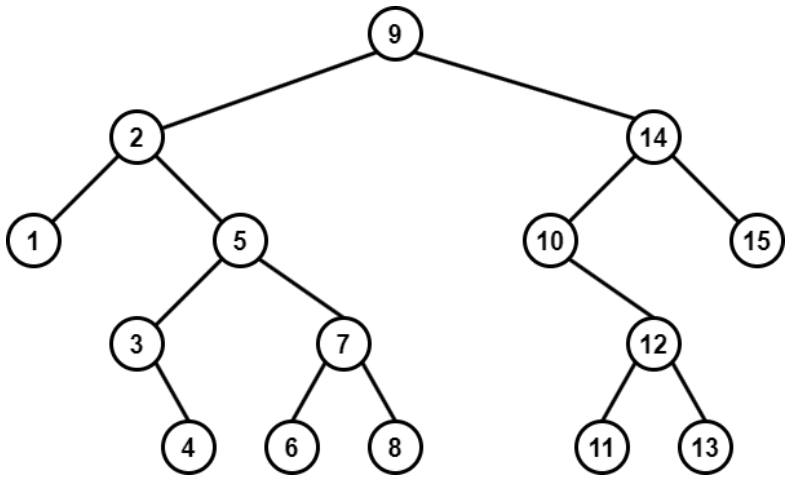
Solution2 (Table):

Node	Parent
1	2
2	6
3	4
4	2
5	4
6	nil
7	8
8	6
9	10
10	11
11	13
12	11
13	8
14	13

7. (20 Points) Use the `TreeInsert()` algorithm to insert the following keys: 9, 2, 1, 5, 3, 4, 7, 6, 8, 14, 10, 12, 11, 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: 9, 5, 2 (in order) from the BST you drew in part (a), then draw the resulting tree.

      Solution: