PROBLEM 1:  (True or False? (8 pts))

Assume the following problems apply to N data elements, where N is very large.

1. Mergesort has fewer comparisons than selection sort. TRUE or FALSE?
2. In the quicksort algorithm, after the call to partition (which returns the index of a pivot element), the elements to the left of the pivot in the array are in sorted order. TRUE or FALSE?
3. The bucket sort algorithm never compares the elements being sorted with each other. TRUE or FALSE?
4. The heap data structure organized as a max-heap (the maximum value is at the root) has the same worst case running time for the delete max operation as a sorted array. TRUE or FALSE?

PROBLEM 2:  (Analysis (Big-Oh) (8 pts))

Consider the following definitions to represent a list of CD’s.

```c
struct CDinfo
{
    string artist;       // name of artist
    string title;        // title of CD
    double price;        // price of CD
};
```

Vector <CDinfo *> ByArtist;       // CD’s sorted by artist
Vector <CDinfo *> ByTitle;        // CD’s sorted by title

Assume there are N CDs. ByArtist contains pointers to the CDs, sorted by artist, and ByTitle contains pointers to the same CDs, sorted by title.

Considering the most efficient algorithm, what is the best representation for the worst case running time (big-Oh) for each of the following?

1. Assume the titles are unique. Given a title, find its CD. __________
2. Print all the titles that contain a phrase. (For example, if the phrase is "for" then the title "Automatic for the People" would be printed in addition to all other titles that contained the phrase "for"). __________
3. Print the CD whose title is first in alphabetical order. __________
4. Suppose that an artist has $M$ CDs in the list of CDs. Given an artist, return the number of CDs by that artist (that is, calculate and return the value of $M$).

5. Print the name of the artist who has the most CD’s in the list.

**Problem 3:**  \textit{(Recursion (8 pts))} \\
Consider the following program (includes not shown).

```cpp
int Mystery(int n)
{
    if (n <= 0)
        return 0;
    else
        return 2 + Mystery(n-1);
}

void Something(int n)
{
    if (n > 0)
    {
        Something(n-1);
        cout << n << " ";
        Something(n-2);
    }
}

int main()
{
    cout << Mystery(4) << endl;
    Something(3);
    cout << endl;
}
```

1. What is the output of the program?
2. Give a recurrence relation for Mystery. (Do not solve it).
3. Give a recurrence relation for Something. (Do not solve it).

**Problem 4:**  \textit{(Stack it many ways: (11 pts))} \\
Consider the Stack class given in lecture and on the exam handout.

**Part A** \textit{(5 pts)}:
Write the Stack member function `Stack::max` whose header is shown below. The function `Stack::max` returns the maximum element in the stack.

For example, consider the stack `S` shown below.

```
stack S
myTop → 49
  52
  67
  43
  98
  37
```

The call `S.max()` would return 98.

Complete function `Stack::max` below.

```
template <class Type>
Type
Stack<Type>::max() const
  // precondition: stack is not empty
  // postcondition: returns the maximum element in the stack
{

PART B (6 pts):
Write the free (not a member function) function `FindMiddle` whose header is shown below. The function `FindMiddle` returns the middle element in the stack.

For example, consider the stack `S` shown below.

```
stack S
myTop → 49
  52
  67
  43
  98
  37
```

The call `FindMiddle(S)` should return 43. If there are two middle elements (as in this case), the one lower down on the stack is returned.

Complete function `FindMiddle` below. Note that `stk` is passed by const reference.

```
int FindMiddle(const Stack<int> & stk)
  // postcondition: returns the middle element in the stack,
  // returns 0 if the stack is empty
{

PROBLEM 5: (Number games: 16 pts))
Consider the *NumberList* class shown on the exam handout.

**PART A (4 pts):**
Write the *NumberList* member function *NumOccurrences* whose header is shown below. The function *NumOccurrences* returns the number of occurrences of *value* in the list of numbers.

For example, assume the *NumberList* N is represented by the list shown below

```
myList
-1 -> -4 -> 3 -> 8 -> -1 -> 10 -> 8 -> 8 -> 5
```

The call *N.NumOccurrences(8)* would return 3, and the call *N.NumOccurrences(-1)* would return 2.

Complete function *NumberList::NumOccurrences* below.

```
int NumberList::NumOccurences(int value)
// postcondition: returns the number of occurrences of value in list.
{
}
```

**PART B (6 pts):**
Write the *NumberList* member function *RemoveNegatives* whose header is shown below. The function *RemoveNegatives* removes all the negative numbers from the list.

For example, assume the *NumberList* N is represented by the list shown in Part A. After the call *N.RemoveNegatives()* the list is shown below.

```
myList
3 -> 8 -> 10 -> 8 -> 8 -> 5
```

Complete function *NumberList::RemoveNegatives* below.

```
void NumberList::RemoveNegatives()
// postcondition: all nodes with negative values have been removed from list
{
}
```

**PART C (6 pts):**
Write the *NumberList* member function *InsertAfterLarger* whose header is shown below. The function *InsertAfterLarger* inserts the value *num* after the first node whose value is larger than *num*, or at the end of the list if there is no value larger.

For example, assume the *NumberList* N is represented by the list shown in Part B. After the call *N.InsertAfterLarger(6)*, the list is shown below.

```
myList
3 -> 8 -> 10 -> 8 -> 8 -> 5
```

Complete function *NumberList::InsertAfterLarger* below.
void NumberList::InsertAfterLarger(int num)
    // postcondition: inserts num after the first node that
    // contains a larger value. If no node contains a larger value,
    // num is inserted at the end of the list.
{

PROBLEM 6:  \textit{(Bloomin Trees: (14 pts))}

Consider the following definition and constructor on the exam handout.

\textbf{PART A (3 pts):}

Write the function \textit{HasLeftOnly} whose header is shown below. The function \textit{HasLeftOnly} returns true if a node has only one child and that child is a left child.

For example, consider the tree T shown below.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure.png}
\end{figure}

The call \textit{HasLeftOnly}(T) returns false, the call \textit{HasLeftOnly}(T->left) returns true, and the call \textit{HasLeftOnly}(T->right->right) returns false.

Complete function \textit{HasLeftOnly} below.

\begin{verbatim}
bool HasLeftOnly(Node * T)
    // postcondition: returns TRUE if T has a left child and no right child,
    // otherwise returns FALSE
{

\end{verbatim}

\textbf{PART B (5 pts):}

Write the function \textit{CountLeftOnly} whose header is shown below. The function \textit{CountLeftOnly} returns the number of nodes that have a left child and no right child.

For example, consider the tree T shown in Part A.

The call \textit{CountLeftOnly}(T) returns 3, the call \textit{CountLeftOnly}(T->left) returns 2, and the call \textit{CountLeftOnly}(T->right) returns 1.

In writing function \textit{CountLeftOnly}, you may use the function \textit{HasLeftOnly} that you wrote in part A. Assume \textit{HasLeftOnly} works correctly, regardless of what you wrote in part A.

Complete function \textit{CountLeftOnly} below.

\begin{verbatim}
int CountLeftOnly(Node * T)
    // postcondition: returns the number of nodes in T that have a left child
{

\end{verbatim}
PART C (6 pts):
Write the function `Expand` whose header is shown below. The function `Expand` adds new nodes to the tree based on the following. For each node P in tree T, such that P has a left child and no right child, a new node is inserted between P and it's child. The value of the new node is the value of the node in P with an 'a' concatenated at the end.

For example, consider the tree T shown in Part A. After the call `Expand(T)`, T is the tree shown below.

In writing function `Expand`, you may use the function `HasLeftOnly` that you wrote in part A. Assume `HasLeftOnly` works correctly, regardless of what you wrote in part A.

Complete function `Expand` below.

```cpp
void Expand(Node * T) {
    // postcondition: For each node P in T that has a left child and no right child, a new node is inserted between P and it's child. The value of the new node is the value of the node in P with an 'a' concatenated at the end.

    (extra page for scratch, must be turned in)
}
```

CPS 100E Exam 2 Handout Spring 1999

Vector class

```cpp
// for a vector of Items use Vector<Item>, e.g., Vector<int> intvector;
```

```cpp
// note: Item must have a default constructor
```

```cpp
// constructors:
```
// Vector() -- default, vector of size 0 (no entries)
// Vector(int size) -- vector with size entries
// Vector(int size, Item fillValue) -- vector w/ size entries all == fillValue
// Vector(const Vector & vec) -- copy constructor

// int Length() -- returns size of vector (capacity)
// void SetSize(int newSize) -- resizes the vector to newSize elements
//                       (can result in losing elements if
//                       new size < old size)
// void resize(int newSize) -- synonym for SetSize

// void Fill(Item fillValue) -- set all entries == fillValue
// operator = -- assignment operator works properly
// operator [] -- indexes both const and non-const vectors

// examples of use:
// Vector<double> dlist(100);    // a list of 100 doubles
// Vector<double> dzlist(100,0.0); // initialized to 0.0
// Vector<String> slist(300);    // 300 strings
// Vector<int> ilist;            // has room for 0 ints

Stack class

template <class Type>
class Stack
{
    public:

    // constructors/destructor

    Stack( );                      // construct empty stack
    Stack( const Stack & s );     // copy constructor
    ~Stack( );                    // destructor

    // assignment

    const Stack & operator = ( const Stack & rhs );

    // accessors

    const Type & top( ) const;    // return top element (NO pop)
    bool isEmpty( ) const;       // return true if empty, else false
int length( ) const; // return number of elements in stack

// modifiers

void push( const Type & item ); // push item onto top of stack
void pop( ); // pop top element
void pop( Type & item ); // combines pop and top
void makeEmpty( ); // make stack empty (no elements)

private:

private:

int myTop; // index of top element
Vector<Type> myElements; // storage for stack

};

For Problem 5

struct Node
{
    int number;
    Node * next;

    Node(int newNumber, Node * newNext); // constructor
};

Node::Node(int newNumber, Node * newNext)
    : number(newNumber), next(newNext)
{
}

class NumberList
{
    public:

    NumberList( );
    int NumOccurences(int value);
    void InsertAfterLarger(int num);
    void RemoveNegatives( );

    // other public members not shown

    private:

    Node * myList; // list of numbers

};
NumberList N;

For Problem 6

struct Node
{
    string name;
    Node * left;
    Node * right;

    Node (string, Node *, Node *);
};

Node::Node (string newName, Node * lf, Node * rt)
    : name(newName), left(lf), right(rt)
{ }