1. (4 points) Consider the list of numbers

41 17 23 4 36 19 12 28 5

(a) After each of the first three passes of Insertion Sort, the list will be

Pass 1: 17 41 23 4 36 19 12 28 5
Pass 2: 17 23 41 4 36 19 12 28 5
Pass 3: 17 23 41 4 36 19 12 28 5

What will the list be after the next pass?

(b) After each of the first three passes of Selection Sort, the list will be

Pass 1: 5 17 23 4 36 19 12 28 41
Pass 2: 5 17 23 4 28 19 12 36 41
Pass 3: 5 17 23 4 12 19 28 36 41

What will the list be after the next pass?

2. (4 points) Given the ordered list of numbers

14 16 17 22 27 31 42

(a) What sequence of numbers will be examined in performing a sequential search for the target 28?

(b) What sequence of numbers will be examined in performing a binary search for the target 28?
3. (6 points) Complete the following C++ function which computes the binomial coefficient \( C(n, k) \). Use the following facts to write the function:

- \( C(n, 0) = C(n, n) = 1; \)
- If \( 0 < k < n \), then \( C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \).

```cpp
int C(int n, int k)
// Precondition: 0 <= k <= n
// Postcondition: Returns binomial coefficient "n choose k" --
//                 number of ways to choose a set of k out of n items
{
    if ( ) return 1;
    return C( , ) + C( , );
}
```

4. (4 points) Show the recursion tree (that is, the “box trace” as described in the text) for evaluating \( C(4, 2) \).

5. (10 points) Here is another (less obvious) C++ implementation of the binomial coefficient function:

```cpp
int C2(int n, int k)
{
    int result = 1;
    for (int i = 1; i <= k; i++) {
        result = result * (n+1-i) / i;
    }
    return result;
}
```

You should check that \( C(4, 2) \) and \( C2(4, 2) \) give the same result.

(a) What is the big-O running time to evaluate \( C2(n, n/2) \), expressed in terms of \( n \)?

(b) Suppose that \( C2(100, 50) \) takes 50 ms to run on a particular machine; estimate how long \( C2(102, 51) \) will take:

(Continued on next page)
(c) An approximate recurrence relation for the running time of the original, recursive version on \( C(n, n/2) \) is:

\[
\begin{align*}
T(0) &= 1 \\
T(n) &= 2 \cdot T(n - 1)
\end{align*}
\]

Solve this recurrence:

(d) Suppose that \( C(100, 50) \) takes 50 ms to run on a particular machine, using the recursive code; estimate how long \( C(102, 51) \) will take:

(e) Which implementation is likely to run faster for large values of \( n \) and \( k \), your recursive solution or this iterative version? Why?

6. (3 points) What is the output of the following sequence of operations if \( x \) is declared as a \texttt{stack<char>}?

```cpp
x.push('h'); x.push('e'); x.push('l');
cout << x.top(); x.pop();
cout << x.top(); x.pop();
x.push('l'); x.push('o');
cout << x.top(); x.pop();
cout << x.top(); x.pop();
cout << x.top(); x.pop();
```

What would the output be from the same sequence if \( x \) were instead declared as a \texttt{queue<char>}, where now \texttt{x.push(c)} inserts \( c \) into the queue (“enqueues” it), \texttt{x.top()} returns the character at the front of the queue (in the C++ standard library, this operation is actually called \texttt{x.front()}), and \texttt{x.pop()} removes the character from the front (“dequeues” it).

7. (3 points) Consider the stack ADT as discussed in class: it only provides the operations \texttt{push}, \texttt{pop}, \texttt{top}, and \texttt{empty} (which returns \texttt{true} if the stack is empty), plus the usual constructor and destructor. Suppose we add the following three operations:

- \texttt{int size()} — return the number of items on the stack
- \texttt{StackItemType retrieve(int i)} — return the item at position \( i \) from the top of the stack
- \texttt{void replace(int i, StackItemType x)} — replace the item at position \( i \) with \( x \)

Give one benefit and one disadvantage to using this modified stack ADT.
8. (6 points) What is the output from the following code?

```c++
struct node {
    int item;
    node *next;
};

int a, b;
int *p, *q;
ode *x, *y;

a = 25;
b = 17;
p = &a;
q = new int;
*p = a + b;
*q = a + b;
cout << "a = " << a << " b = " << b << endl;
cout << "p = " << *p << " q = " << *q << endl;
p = q;
q = 0;
cout << "p = " << *p << endl;

x = 0;
y = new node;
y->item = 5;
y->next = x;

x = new node;
x->item = 3;
x->next = y;
for (y = x; y != 0; y = y->next) {
    cout << y->item << endl;
}
```

Now write a few more lines to “clean up” after the above code by deleting all of the dynamically allocated storage: