Below, I am giving (hopefully helpful) clues for handling some of the problems that, according to my experence, students have difficulty with.

But don't forget the other material, such as loop invariants, logarithms, time complexity of algorithms, Huffman coding, and dynamic programming, and the very important Omega(n log n) lower bound for comparison based sorting.

Time Complexity of Code Fragments.

1. Think.
2. How many times can you double a number before you reach or exceed n? Approximately $\log n$ times. Starting at n, how many times can you halve that number before reaching 1? Approximately $\log n$
3. Starting at a number larger than 1, how many times can you square the number before you reach n? Aproximately $\log \log n$. Starting at n, how many times can you take the square root before you reach, or go below, 2? Again, approximately $\log \log n$.
4. Here is a method of handling those double loop examples. Assume that i is the index of the outer loop, j the index of the inner. (a) Start with your favorite value of n. (The best choice depends on the problem.) Write a column of all values of i that will occur. For each i, write a row consisting of all values of j that will occur for that value of i. Then count how many entries in the table. Here is an example:

```
for(int i = 1; i < n; i++)
 for(int j = i; j < n; j = 2*j)

Assume $n = 20$.

\begin{verbatim}
  i    j

  1    1 2 4 8 16
  2    2 4 8 16
  3    3 6 12
  4    4 8 16
  5    5 10
  6    6 12
  7    7 14
  8    8 16
  9    9 18
  10   10
  11   11
  12   12
  13   13
  14   14
  15   15
  16   16
```

```
17   17
18   18
19   19
```

The first column has 19, approximately n, entries. The second column has 16, less than between n and 2n, entries. So the total number of entries is between n and 3n, which is $\Theta(n)$

Next example.

```
for(int i = 1; i < n; i++)
 for(int j = 1; j < i; j++)
```

```
1
2    1
3    1 2
4    1 2
5    1 2 4
6    1 2 4
7    1 2 4
8    1 2 4
9    1 2 4 8
10   1 2 4 8
11   1 2 4 8
12   1 2 4 8
13   1 2 4 8
14   1 2 4 8
15   1 2 4 8
16   1 2 4 8
17   1 2 4 8 16
18   1 2 4 8 16
19   1 2 4 8 16
```

What is $\log 20$? between 4 and 5, closer to 4. There are 64 entries. If you divide that by n, what do you get? Between 3 and 4. That's sort of close to $n \log n$.

The Queue Example. Assume that N is at least as large as the number of time insert will be executed during the run of the program. False overflow will not happen.

Binary Tree. I expect you to be able to traverse a binary tree in preorder, postorder, or inorder, using recursion.

A Very Simple Dynamic Program. Write dynamic programm code to compute an array int F[N], where F[0] = 0, F[1] = 1, F[2] = 3, and F[n] = F[n-3] + 2*F[n-2] + 3*F[n-1] for any $n \geq 3$. Here is the solution I would want to see.

```
void computeF()
 {
  F[0] = 0;
```

```
 F[1] = 1;
 F[2] = 3;
 for(int i = 4; i < N; i++)
  F[i] = F[i-3]+2*F[i-2]+3*F[i-1];
}
```

An alternative approach is to write a recursive program. (This is still dynamic programming, though.)

```
int F(int i)
 {
  if(i == 0) return 0;
  else if(i == 1) return 1;
  else if(i == 2) return 3;
  else return F(0) + 2*F(i-2) + 3*F(i-1);
 }
```

However, this code is very wasteful: its execution time is an exponential function of N. Do you see why?