1. Find the asymptotic complexity of each of these code fragments.
   (a) for(int i = 0; i < n; i++)
       for(int j = 1; j < i; j = j * 2j)

   (b) for(int i = 0; i < n; i++)
       for(int j = i; j < n; j = j * 2j)

2. Give an asymptotic solution to each of these recurrences.
   (a) $F(n) = 2F(n/2) + n$

   (b) $F(n) = F(n - \log n) + \log n$

   (c) $F(n) = 4F(n/2) + n^3$

   (d) $F(n) = 4F(n/2) + n^2$

   (e) $F(n) = 4F(n/2) + n$

3. The loop invariant of a loop in a program is a condition that satisfies the following to statements:
   (a) The condition is true before the first iteration of the loop.
   (b) If the condition holds at the beginning of any iteration, it is true at the end of that iteration.
Given that a condition is a loop invariant of a loop, it is true after the last iteration of the loop. This fact, together with the loop condition, is useful during analysis of code.

A loop invariant has to be Boolean, that means that it’s either true or false. But it doesn’t have to be an equation – it could be simply a sentence. Here is an example.

```c
int A[N]; // assume that the values of this array have been given

int sumA()
{
    int i = 0;
    int sum = 0;
    // Loop invariant holds here
    while (i < N)
    {
        // start of loop
        sum = sum + A[i];
        i++;
        // end of loop
    }
    // Loop invariant holds here
    return sum;
}
```

The loop invariant is the following statement: “sum is the sum of the first i values of the array A”
If the invariant holds at the start of any iteration of the loop, it holds at the end of that iteration.

Here is another code fragment. The code is for selection sort and its purpose is to sort the array A in increasing order. There are two loops, and each has a loop invariant. Find those two loop invariants.

```c
void selectionsort() // sorts the array A[N]
{
    int i = 0;
    while(i < N)
    {
        int j = i+1;
        while(j < N)
        {
            j++;
        }
        i++;
    }
}
```

Outer loop invariant: the first i entries of A, namely A[0] ... A[i-1], have their final values.
Inner loop invariant: A[i] is less than or equal to A[k] for all i ≤ k ≤ j
4. Here is C++ code that implements binary search tree of character. It ran, but then I deleted most of the lines. No lines were deleted from procedure main, and the definitions of bstnode and bst are not changed.

```cpp
struct bstnode;
const int N = 20;
typedef bstnode*bst;
struct bstnode
{
    char kee;
    bst left = NULL;
    bst right = NULL;
};

bst mainroot;

void insert(bst&root,char newkee)
{

}

void preorder(bst root)
{

}

void inorder(bst root)
{

}

int kount(bst root)
    // returns number of nodes in the subtree rooted at root
{

}

int hite(bst root)
    // returns height of the subtree rooted at root
{

}
```
int main()
{
    char k;
    for(int i = 0; i < N; i++)
    {
        cin >> k;
        insert(mainroot,k);
    }
    inorder(mainroot);
    cout << endl;
    preorder(mainroot);
    cout << endl;
    cout << "The tree has " << kount(mainroot) << " nodes and height "
         << hite(mainroot) << endl;
    return 1;
}

(a) Rewrite the missing parts of the code.
(b) Here is an output of the original program, with my input.

ABEGHJLMNOPQRSTUVWXZ
TGEBARMJHLQONPSWUVZX
The tree has 20 nodes and height 6

You now have enough information to recover the binary search tree constructed by the program.
Sketch that tree.

5. The following is C++ code for a function that computes a floating point number to the power of a positive integer. Find a useful loop invariant for the loop in this function.

float power(float x, int n) // input condition: n > 0
{
    assert(n > 0);
    float z = 1.0;
    float y = x;
    int m = n;
    while(m > 0)
    {
        if(m%2) z = y*z; // that means m is odd
        y = y*y;
        m = m/2;
    }
    return z;
}