Joseph Steeb

1.
   a. Both programs have a runtime complexity of O(n) because they loop n times.
   b. int g(int n)

      {

         return n;

      }
2. g(n) has Olog(n) complexity, because there is division by 2 in the number of times it loops.
3.

```
#include <iostream>
bool checkArr(bool nums[10])
{
    for(int i = 0; i < 10; i++)
    {
        if(!nums[i])
            return false;
    }
    return true;
}

int findK(int n)
{
    int k = 1;
    bool nums[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
    while(!checkArr(nums))
    {
        int nMultiplied = k*n;
        int nCopy = nMultiplied;
        unsigned int nLength = 0;

        do
        {
            ++nLength;
            nCopy /= 10;
        }while(nCopy);

        nums[nMultiplied % 10] = true;

        for(int i = 1; i < nLength; i++)
        {
            int divisor = pow(10,i);
            nums[nMultiplied / divisor % 10] = true;
        }
        k++;
    }
    return k-1;
}
```

It would be very difficult to directly formalize a worst case for this algorithm, because there is no relationship between the size of the number, and the complexity of the algorithm. I estimate that the worst case is somewhere around O(n^2).

4.

a. O(1). requires only the use of modulo. if(!n%2){return true;}
b. O(logn) if the list is sorted, otherwise O(n). If the list is sorted binary search can be used which halfs the search every time, resulting in logarithmic complexity, otherwise the whole list must be iterated over, resulting in O(n).
c. O(1) if the list is sorted, O(n) if unsorted. If sorted, simply return the first or last element, otherwise the whole list must be iterated over, resulting in O(n).
d. O(n^2). For each element of one list, the entire second list must be iterated over.
e. O(n). Simply compare the values at each index.
f. O(log(n)). The equation to get the height of a BST is ceiling[log(n+1)], and If we are finding worst case complexity, the full height of the tree must be traversed.

5.

```
#include <iostream>

bool contains(char c, std::string s)
{
    for(char const &element : s)
    {
        if(element == c)
            return true;
    }
    return false;
}

bool isAnnogram(std::string s1, std::string s2)
{
    for(char const &element : s2)
    {
        if(!contains(element, s1))
            return false;
    }
    return true;
}
```