1. A. F() = **O(N)** G() = **O(N)**

   B.
   int h(int n)
   {
   return n;
   }

2. **O(logn^2)**

```
int findK(int value, int valSize)
{
    int arr[10] = { 0 };
    int k = 0;
    int currentDecimal;
    bool arrStatus = false;



    while (arrStatus == false)
    {
        k++;
        int tempVal = value;
        tempVal *= k;
        for (int i = 0; i < valSize; i++)
        {

            currentDecimal = tempVal % 10;
            tempVal /= 10;

            for (int j = 0; j < 10; j++)
            {
                if (currentDecimal == j)
                    arr[j] += 1;
            }
        }

        for (int i = 0; i < 10; i++)
        {
            arrStatus = true;
            if (arr[i] == 0)
            {
                arrStatus = false;
                break;
            }

        }



    }

    return k;

}
```

3. No you cannot, because if you enter in a value of 1234 and a value of 12345, it will still have the same time complexity, so size does not matter when it comes to formulating worst case.

4. A.
```
void h(int n)
{
    if (n % 2 == 0)
        cout << "Even" << endl;
    else
        cout << "Odd" << endl;
}
```
O(1)

B.
```
bool h(int n[], int key, int size)
{
    for (int i = 0; i < size; i++)
    {
        if (key == n[i])
            return true;
    }

    return false;
}
```
O(n)

C.
```
int smallest(int n[], int size)
{
    int smallNum = 0;

    for (int i = 0; i < size; i++)
    {
        if (i == 0)
            smallNum = n[i];
        else
        {
            if (n[i] < smallNum)
                smallNum = n[i];
        }
    }

    return smallNum;
}
```
O(n)

D.

```
bool sameVals(int arr1[], int arr2[], int size)
{

    int ifSame[] = { 0 };

    for (int i = 0; i < size; i++)
    {
        int j;
        for (j = 0; j < size; j++)
        {
            if (arr1[i] == arr2[j] && ifSame[j] == 0)
            {
                ifSame[j] == 1;
                break;
            }
        }

        if (j == size)
            return false;
    }

    return true;
}
```

O(n^2)

E.

```
bool sameVals(int arr1[], int arr2[], int sizeArr1, int sizeArr2)
{

    bool status = true;

    if (sizeArr1 != sizeArr2)
        return false;

    int size = sizeArr1;

    for (int i = 0; i < size; i++)
    {
        if (arr1[i] != arr2[i])
        {
            status = false;
            return status;
        }

    }

    return status;
}
```

O(n)

F.

```
bool findNode(struct Node* node, int x)
{
    if (node->data = x)
        return true;

    bool ifLeft = findNode(node->left, x);
    if (ifLeft)
        return true;

    bool ifRight = findNode(node->right, x);
    return ifRight;
}
```

O(logn)

5.
```
bool anagram(string s1, string s2)
{
    int s1Length = s1.length();
    int s2Length = s2.length();

    if (s1Length != s2Length)
        return false;

    sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());

    for (int i = 0; i < s1Length; i++)
    {
        if (s1[i] != s2[i])
            return false;

    }

    return true;
}
```
The time complexity for this algorithm is O(nlogn).