11. Prove and demonstrate that binary search takes O(log(n)).

```
public static int BinarySearch(int key, int[] A, int LI, int HI)
        {
                int mid = (LI+HI)/2;
                if(LI>HI)
                {
                        return -1;
                }

                if(key==A[mid])
                {
                        return mid;
                }
                else if(key<A[mid])
                {
                        return BinarySearch(key, A, LI, mid-1);
                }
                else
                {
                        return BinarySearch(key, A, mid+1, HI);
                }

        }
```

The algorithm of binary search splits the sorted array into half and search one of them first, and then determine if the key is inside.
To accomplish this, the algorithm call will need to keep dividing the size of the array by 2 until it reaches the base case, where either the left and right meet without finding the key, or the key is found at a certain index of the array. This means that n started from LI-HI+1, and each time after the iteration, it is divided into half and comes to a stop when it becomes one or zero. For example, it is n for the first iteration, n/2 for the second, and n/4 for the third.
To determine how many times the iteration will reach the base case, we can have: 2^x = n → x = log(n)
This calculation explains how we can divide log(n) times to reach the base case and find the key element, and this number of iterations proves the time complexity of this algorithm is O(log(n)).