What is the Big-O Time Complexity Analysis of Linear Search?

```
// Java code for linearly search x in arr[]. If x
// is present then return its location, otherwise
// return -1

class GFG
{
public static int search(int arr[], int x)
{
    int n = arr.length;
    for(int i = 0; i < n; i++)
    {
        if(arr[i] == x)
            return i;
    }
    return -1;
}

public static void main(String args[])
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;

    int result = search(arr, x);
    if(result == -1)
        System.out.print("Element is not present in array");
    else
        System.out.print("Element is present at index " + result);
}
}
```

Description=>
Here we are Finding only one element in the unsorted array of n element. So the length of array is n. if the element we are looking for is n'th elements then number of elements we are checking are n. if the number is in the middle of off somewhere then the index number of that element is the number of elements we checked.
The function has single loop. That's why Big-O Time complexity of Linear search is (it usually translates into a running time complexity of) O(1) or O(n).
In worst case scenario program will take n iterations