Let us discuss the complexity of linear search and binary search.

Linear Search Algo:→

1. Set Found equal to false.
2. Set Loc equal to 1.
3. while Loc <= n and not found do the following.
4. If Item = A[Loc] then.
5. set found to True.
6. else
   ~~Set~~ Increment Loc by 1.

The worst case is obvious that in which Item is not in the list.

So in the worst case we find the computing time for linear search algorithm as follows

Statement	# of times executed
1	1
2	1
3	n+1
4	n
5	0
6	n
TOTAL	3+3n

$T_L(n) = O(n)$

Binary Search Algo:→

1. Set found = false.
2. Set first = 1
3. Set last = n.
4. while first <= last and not found do the following.
5. Calculate mid = (first + last)/2
6. If Item < A[mid] then set
7. last = mid - 1
8. Else if Item > A[mid] then
9. Set first = mid + 1
10. else set found to True.

→ In this algorithm, Statemen 1, 2, and 3 are executed exactly once.

→ To calculate the worst Case Complexity, we must determine the number of passes (times) the loop composed of Statement 4 through 10 is executed.

→ Each pass through this loop reduces the size of the sublist still to be searched by at least one half.

2 10/4 = (2)

→ The last pass occurs when the sub list reaches size one.

→ Thus the total number of iterations of this loop is 1 plus the no. of k passes required to produce a sublist of size one.

→ Since the size of the sublist after k passes is at most $n/2^k$, we must have

$$\frac{n}{2^k} < 2$$

that is

n = 25.

① $\frac{25}{2^1} = 12$

$\frac{25}{2^2} = \frac{25}{4} = 6$

$\frac{25}{2^3} = \frac{25}{8} = 3$

$\frac{25}{2^4} = \frac{25}{16} = 1 <$