Best case time complexity, Average case time complexity, Worst case time complexity and Lower bound

We talk about the **Best case time complexity,** the **Average case time complexity** and the **Worst case time complexity** in the context of an algorithm **A** to solve a problem **P**.

Example 1.

Consider **Linear Search** algorithm to search an item in an (unsorted) array of size n.

Best case: The item is in the very first location. Time complexity is O(1)

Average case: The item is in the middle of the array. Time complexity is O(n/2) = O(n)

Worst case: The item is either at the last location or the item is not in the array. Time complexity is O(n).

We talk about **lower bound** in the context of a problem **P**. What we say applies to the worst case time complexity all algorithms **A** (present and future) to solve the problem **P**.

Example 2.

Consider the problem of determining whether an item is in the array. Can you determine that fact without checking all n items in the array?

No. Therefore, whatever the algorithm, it must check all items in the array at least once in the worst case to conclude whether or not an item is in the array. Therefore, the lower bound of the problem in this case is $\Omega(n)$.

We have the following relationship.

Let **P** be a problem and **A** be an algorithm to solve **P**. Then

The Worst-case time complexity A $\geq$ The Lower Bound of (P).

If Worst case time complexity A is equal to the Lower Bound of (P), then A is called an optimal algorithm to solve P.

Example: Linear search is an optimal algorithm to search an item in an unsorted array.

In other words, lower bound refers to "**minimum amount of work" any algorithm A must do to correctly solve the problem P in the worst case.**