**CS 311** California State University SAN MARCOS

## 1 Time Complexity

The Time Complexity of an Algorithm is the running time needed by an algorithm expressed as a function of the input size.

Input Size can be many elements to sort, the total number of bits, the number of edges of a graph, etc.

Running Time is the number of primitive operations or steps executed. Examples include arithmetic operators like add, subtract, modulus, floor, etc. A primitive operation takes a fixed amount of time to perform.

The amount of work an algorithm does depends on the Input Size and the nature of the Input.

The Nature of Input requires us to identify the worst and best-case scenarios.
1. Worst Case: maximum number of operations performed by an algorithm of the input size (Upper-Bound).
2. Best Case: minimum number of operations performed by an algorithm of the input size (Lower-Bound).
3. Average Case: average number of operations performed by an algorithm of the input size. It takes into account the probability of each type of input.

## 2 Notation

1. Big O Notation: expresses the time complexity of the upper bound of the function, as the input size goes to infinity. A function $f(n)$ is $O(g(n))$ if there exist positive numbers $c$ and $N$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq N$.In simpler terms, with a larger input $n$ the function $f(n)$ is at most $c \cdot g(n)$.

2. Big $\Omega$ Notation: expresses the time complexity of the lower bound of the function, we can say this due to symmetry. A function $f(n)$ is said to be $\Omega(g(n)) \iff$ there exist positive constants $c$ and $n_0$ such that for all $f(n) \geq c \cdot g(n)$. In simpler terms, with a larger input size $n$, the function $f(n)$ is at least $c \cdot g(n)$.

3. Big $\Theta$ Notation: expresses the time complexity of the upper and lower bound of the function. A function $f(n)$ is said to be $\Theta g(n)$ if there exist positive numbers $c_1$, $c_2$, and $N$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq N$. It tells us that $f(n)$ is bounded above and below $g(n)$, up to constant factors, for sufficiently large $n$ meaning the run time of the algorithm is sandwiched between multiples of $g(n)$ when $n$ is large enough.

## 3 Proving Time Complexity

Example:

Suppose $f(n)$ is a function which is defined as $f(n) = n^2 + \log_2(n)$. We want to express this function in Big $\Theta$ Notation.

1. Determine the tight lower-bound as a function $g(n)$:
   $g(n) = n^2$.
2. By Definition of Big $\Theta$ Notation:
   $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq N$
3. By Substitution:
   $c_1 \cdot n^2 \leq n^2 + \log_2(n) \leq c_2 \cdot n^2$
4. Prove Inequality:
   $c_1 \cdot n^2 \leq n^2 + \log_2(n) \leq c_2 \cdot n^2$
   $c_1 \leq 1 + \frac{\log_2(n)}{n^2} \leq c_2$
   Let $c_1 = 1$, $c_2 = 1$, and $n_0 = 1$, for $1 \leq n \leq N$.
   $1 \leq 1 + \frac{\log_2(1)}{1^2} \leq 1$
   $1 \leq 1 + \frac{0}{1} \leq 1$
   $1 \leq 1 + 0 \leq 1$
   $1 \leq 1 \leq 1$
   The inequality holds for all $1 \leq n \leq N$ for $f(n)$.
5. We can conclude that $f(n)$ is $\Theta(n^2)$.

## 4 Real World Example

1. Linear Search:

Suppose we have an unsorted array of $n$ integers. We know that the best case of this algorithm $O(1)$ which happens if the first integer searched is the element we are trying to find. The worst case of this algorithm is when the since we are searching for doesn't exist within the array resulting in O(n). The probability of the best case is $\frac{1}{n}$ which becomes a low probability as $n$ grows larger. The probability of the worst case is $\frac{n-1}{n}$, which becomes a higher probability as $n$ grows larger we can say the time complexity is $O(n)$, but let's prove it.

Proof:

Suppose the time complexity of the linear search algorithm is represented as a function $f(n) = n$.

By the definition of Big $O$:

$f(n) \leq c \cdot g(n)$ for all $n \geq N$.

We want to choose a function $g(n)$ that represents the tightest lower bound of the function $f(n)$.

Let $g(n) = n$ and $c = 1$.

By Substitution:

$n \leq 1 \cdot n$

$1 \leq 1$

Thus, the time complexity of $f(n)$ is $O(n)$.