KS091201
MATEMATIKA DISKRIT
(DISCRETE MATHEMATICS)

Complexity of Algorithm

Discrete Math Team

Outline

- Big-Ω Notation
- Big-θ Notation
- Complexity of Algorithm

Review Big-Oh

- Let f and g be functions. We say that $f(x)$ is $O(g(x))$ if there are constants c and k such that

  $$|f(x)| \le C\ |g(x)|$$

  whenever $x > k$
- Big-Oh notation is used to estimate the number of operations needed to solve a problem using a spesified procedure or algorithm.

Big-Ω and Big-θ Notation

- Big-O notation does not provide a lower bound for the size of *f(x)* for large x, for this we use big-Omega (Big-Ω) notation.

- When we want to give both, an upper and lower bound on the size of a function *f(x)*, relative to a reference function *g(x)*, we use big-Theta (Big-θ) notation.

Formal Definition of Big-Ω

- Let *f* and *g* be function from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are positive constants C and k such that

$$|f(x)| \geq C\ |g(x)| \text{ whenever } x > k$$

- This is read as "*f(x)* is big-Omega of *(g(x))*".
- Alternatively, we can say that:

$$\exists k \in \mathbf{R}, \exists c \in \mathbf{R}, \forall x \in \mathbf{R}, x > k \Rightarrow |f(x)| \geq c\ |g(x)|$$

Example Big-Ω Problem

- The function $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$, where $g(x)$ is the function $g(x) = x^3$.
- $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$.
  - since $8x^3 + 5x^2 + 7 > 8x^3$ for all $x > 0$.
- $f(x) = \Omega(g(x))$ is equivalent to $g(x) = O(f(x))$

Formal Definition of Big-θ

- Let f and g be function from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$.
- When $f(x)$ is $\theta(g(x))$ we say that "$f(x)$ is big-Theta of $(g(x))$" and we also say that $f(x)$ is of order $(g(x))$
- When $f(x)$ is $\theta(g(x))$, it is also the case that $g(x)$ is $\theta(f(x))$.
- Note that $f(x)$ is $\theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$.

Proof

- $f(x)$ is $\theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$.

  - If $f(x)$ is $\theta(g(x))$, then there exist constants C_1 and C_2 with $C_1|g(x)| \le |f(x)| \le C_2|g(x)|$.
  - It follows that $|f(x)| \le C_2|g(x)|$ and $|g(x)| \le 1/C_1|f(x)|$ for $x > k$.
  - Thus $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$.
  - Conversely, suppose that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$, then there exist constants C_1, C_2, k_1, k_2 such that $|f(x)| \le C_1|g(x)|$ for $x > k_1$ and $|g(x)| \le C_2\ |f(x)|$ for $x > k_2$.
  - We can assume that $C_2 > 0$ (we can always make C_2 larger). Then we have $1/C_2|g(x)| \le |f(x)| \le C_1|g(x)|$ for $x > \max(k_1, k_2)$. Hence $f(x)$ is $\theta(g(x))$.

Example Big-θ Problem

- Show that $3x^2 + 8x \log x$ is $\theta(x^2)$

- Solution:
  - $f(x)$ is $\theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$
  - We can find c_1 and c_2 such that $c_1 \, |g(x)| \le |f(x)| \le c_2 \, |g(x)|$.
  - $3x^2 + 8x \log x$ is $\theta(x^2)$
    - $3x^2 + 8x \log x < c.\, x^2$
    - $3x^2 + 8x^2 < 11x^2$ for $x > 1$
      - $\therefore\ 3x^2 + 8x \log x = O(x^2)$.
    - $3x^2 + 8x \log x > x^2$ for $x > 1$
      - $\therefore\ 3x^2 + 8x \log x = \Omega(x^2)$.

Complexity of Algorithm

- How can the efficiency of an algorithm be analyzed?
- One measure of efficiency is the time used by a computer to solve a problem using the algorithm when input values are of specified size → time complexity
- A second measure is the amount of computer memory required to implement the algorithm when input values are of specified size → space complexity
- In this section we will discuss the time complexity.

Comparison of running times

- Searches
  - Linear: n steps
  - Binary: 2 log n steps

- Sorts
  - Bubble: n^2 steps
  - Insertion: n^2 steps

Time Complexity of Max Element Algorithm

- The number of comparisons will be used as the measure of the time complexity since comparisons are the basic operations used.
- Two comparisons are used for each of the second through the nth elements and one more comparison to exit the loop when $i = n + 1$, exactly $2(n - 1) + 1 = 2n - 1$.
- Hence the algorithm for finding max element of a set of n elements has time complexity $\theta(n)$, measured in terms of the number of comparisons used.

Time Complexity of Linear Search Algorithm

- At each step of the loop, two comparisons are performed – one to see whether the end of the loop has been reached and one to compare the element x with a term in the list. One more comparison is made outside the loop.
- Consequently, if $x = a_i$, $2i + 1$ comparisons are used.
- The most comparison, $2n + 2$, are required when the element is not in the list – $2n$ comparisons are used to determine that x is not a_i, an additional comparison is used to exit the loop, and one more comparison outside the loop.
- Hence, a linear search algorithm requires at most $\theta(n)$. This is worst case complexity.
- Worst case analysis tells us how many operations an algorithm requires to guarantee that it will produce a solution.

Time Complexity of Binary Search Algorithm

- Binary search requires at most $2 \log n + 2$ comparisons when the list being searced has 2^k elements, where $k = \log n$.
- If n is not a power of 2, the original list is expanded with 2^{k+1} terms, where $k = \lfloor \log n \rfloor$ and the search requires at most $2 \lceil \log n \rceil + 2$ comparisons .
- Consequently, binary search requires at most $\theta(\log n)$ comparisons.
- This is average case complexity → the average number of operations used to solve the problem over all inputs of a given size.

Average Case Performance of Linear Search Algorithm

- If x is *i*th term in the list, 2*i* + 1 comparisons are needed.
- The average number of comparisons used equals:

$$\frac{3+5+7+\cdots+(2n+1)}{n} = \frac{2(1+2+3+\cdots+n)+n}{n}$$

$$1+2+3+\cdots+n = \frac{n(n+1)}{2}$$

- Hence, the average number of comparisons used by linear search algorithm (when x is known to be in the list) is $\frac{2[\frac{n(n+1)}{2}]}{n} + 1 = n + 2$, which is $\theta(n)$

Worst Case Complexity of Two Sorting Algorithms

- **Bubble sort:**
  - Total number of comparisons used by bubble sort to order a list of n elements is:

$$(n-1)+(n-2)+\cdots+2+1=\frac{(n-1)n}{2}$$

  - So it has $\theta(n^2)$ worst case complexity

- **Insertion sort:**
  - Total number of comparisons used by insertion sort to order a list of n elements is:

$$2+3+\cdots+n=\frac{n(n+1)}{2}-1$$

  - So it has $\theta(n^2)$ worst case complexity

Commonly Used Terminology for Complexity of Algorithms

Complexity	Terminology
$\theta(1)$	Constant complexity
$\theta(\log n)$	Logarithmic complexity
$\theta(n)$	Linear complexity
$\theta(n \log n)$	$n \log n$ complexity
$\theta(n^b)$	Polynomial complexity
$\theta(b^n)$, where $b > 1$	Exponential complexity
$\theta(n!)$	Factorial complexity