Algorithms

COMP 53 - DATA STRUCTURE

UNIVERSITY OF THE PACIFIC

Algorithm Run time

- It is ideal to design algorithms that aim to lower amount of time it needs to run.
- A single algorithm can always execute more quickly on a faster processor.
- The analysis of an algorithm does not describes runtime in terms of nanoseconds.
- Instead, it is described as the number of constant time operations.
- A constant time operation is an operation that, for a given processor, always operates in the same amount of time, regardless of input values.

Constant Time Operations

Constant time operations

```
x = 10        // assignment
y = 20        // assignment
a = 1000      // assignment
b = 2000      // assignment

z = x * y     // multiplication

c = a * b     // multiplication
```

Each multiplication completes in the same amount of time

Non-constant time operations

```
for (i = 0; i < x; i++)
   sum += y
```

```
// assume str1 is a string
// assume str2 is a string
str3 = concat(str1, str2)
```

Constant Time Operations

- The programming language being used, as well as the hardware running the code, both affect what is and what is not a constant time operation.

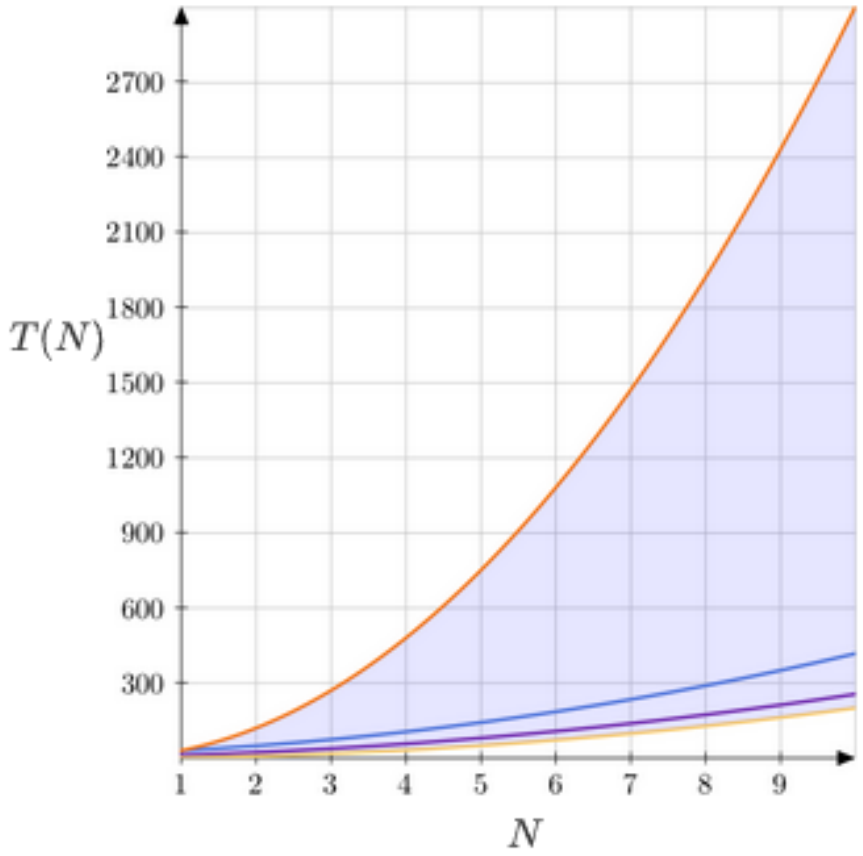
Operation	Example
Addition, subtraction, multiplication, and division of fixed size integer or floating point values.	w = 10.4 x = 3.4 y = 2.0 z = (w - x) / y
Assignment of a reference, pointer, or other fixed size data value.	x = 1000 y = x a = true b = a
Comparison of two fixed size data values.	a = 100 b = 200 **if** (b > a) { ... }
Read or write an array element at a particular index.	x = arr[index] arr[index + 1] = x + 1

Run time complexity

- An algorithm with runtime complexity T(N) has a lower bound and an upper bound.
- These lower and upper bounds provide a general picture of the runtime.

- Lower bound: A function f(N) that is ≤ the best case T(N), for all values of N ≥ 1.
  - Best case run time: $T(N) = 7N + 36$
  - Lower bound: $f(N) = 7N$
- Upper bound: A function f(N) that is ≥ the worst case T(N), for all values of N ≥ 1.
  - Worst case run time: $f(N) = 3N^2 + 10N + 17$
  - Upper bound: $f(N) = 30N^2$

Growth of complexity



Algorithm runtime complexities

- $3N^2 + 10N + 17$ (worst case)
- $2N^2 + 5N + 5$ (best case)

Algorithm runtime bounds

- $2N^2$ (lower)
- $30N^2$ (upper)

Examples on lower bound

- Let best case run time complexity be $T(N) = 3N^2 + 4N + 1$
  - Lower bound: $f(N) = 3$
  - Lower bound: $f(N) = 3N$
  - Lower bound: $f(N) = 3N^2$
  - Lower bound: $f(N) = 3N^2 + 4N$
  - Lower bound: $f(N) = 3N^2 + 4N + 1$

Examples on lower bound

- Let best case run time complexity be $T(N) = 3N^2 + 4N + 1$
  - Not a lower bound: $f(N) = 3N^2 + 4N + 2$ (try $N = 1$)
  - Not a lower bound: $f(N) = 5N^2$ (try $N = 3$)
  - Not a lower bound: $f(N) = N^3$ (try $N = 5$)
  - Not a lower bound: $f(N) = N^2 \log N$ (try $N = 10000$)

Examples on upper bound

- Let worst case run time complexity be $T(N) = 3N^2 + 4N + 1$
  - Upper bound: $f(N) = 3N^2 + 4N + 2$
  - Upper bound: $f(N) = 10N^2$
  - Upper bound: $f(N) = 3N^3 + 4N + 1$
  - Upper bound: $f(N) = 3N^4 + 4N^2 + 1$

Examples on upper bound

- Let worst case run time complexity be $T(N) = 3N^2 + 4N + 1$
  - Not an upper bound: $f(N) = 3N^2 + 4N$ (try $N = 1$)
  - Not an upper bound: $f(N) = 2N^2$ (try $N = 1$)
  - Not an upper bound: $f(N) = 2N^2 + 4N$ (try $N = 1$)
  - Not an upper bound: $f(N) = N^2 \log N$ (try $N = 1$)

Asymptotic Notations

- An additional simplification can factor out the constant from a bounding function.
  - Upper bound: $30N^2$
  - Growth rate: N^2

- O notation provides a growth rate for an algorithm's upper bound.
  - $T(N) = O\big(f(N)\big),\ a\ constant\ c\ exists\ such\ that\ for\ all\ N \geq 1, T(N) \leq c \times f(N)$

- Ω notation provides a growth rate for an algorithm's lower bound.
  - $T(N) = \Omega\big(f(N)\big),\ a\ constant\ c\ exists\ such\ that\ for\ all\ N \geq 1, T(N) \geq c \times f(N)$

- Θ notation provides a growth rate that is both an upper and lower bound.
  - $T(N) = \Theta\big(f(N)\big),\ when\ T(N) = O\big(f(N)\big)\ and\ T(N) = \Omega(f(N))$

Asymptotic Notations: O

- *O* notation provides a growth rate for an algorithm's upper bound.
  - $T(N) = O\big(f(N)\big),\ a\ constant\ c\ exists\ such\ that\ for\ all\ N \geq 1, T(N) \leq c \times f(N)$

- Example:
  - $2n = O(n)$, let $c = 3$
  - $\log n = O(n)$, let $c = 1$
  - $n^2 \neq O(n)$, there is no constant c such that for all $n \geq 1$, $n^2 \leq cn$
  - $n \log n \neq O(n)$, there is no constant c such that for all $n \geq 1$, $n \log n \leq cn$

  - $5n^2 = O(n^2)$, let $c = 6$
  - $3n = O(n^2)$, let $c = 3$
  - $4n \log n = O(n^2)$, let $c = 4$
  - $2n^3 \neq O(n^2)$, there is no constant c such that for all $n \geq 1$, $2n^3 \leq cn^2$
  - $n^2 \log n \neq O(n^2)$, there is no constant c such that for all $n \geq 1$, $n^2 \log n \leq cn^2$

Asymptotic Notations: Ω

- Ω notation provides a growth rate for an algorithm's lower bound.
  - $T(N) = \Omega\big(f(N)\big),\ a\ constant\ c\ exists\ such\ that\ for\ all\ N \geq 1, T(N) \geq c \times f(N)$

- Example:
  - $n^2 = \Omega(n)$, let $c = 1$
  - $2n = \Omega(n)$, let $c = 1$
  - $n \log n + 2 = \Omega(n)$, let $c = 1$
  - $\log n \neq \Omega(n)$, there is no constant c such that for all $n \geq 1$, $\log n \geq cn$

  - $5n^2 = \Omega(n^2)$, let $c = 4$
  - $n^2 \log n + 2 = \Omega(n^2)$, let $c = 1$
  - $n^3 = \Omega(n^2)$, let $c = 1$
  - $4n \neq \Omega(n^2)$, there is no constant c such that for all $n \geq 1$, $4n \geq cn^2$
  - $n \log n \neq \Omega(n^2)$,there is no constant c such that for all $n \geq 1$, $n \log n \geq cn^2$

Asymptotic Notations: Θ

- Θ notation provides a growth rate that is both an upper and lower bound.
  - $T(N) = \Theta\big(f(N)\big),\ when\ T(N) = O\big(f(N)\big)\ and\ T(N) = \Omega(f(N))$

- Example:
  - $n = \Theta(n)$
  - $2n = \Theta(n)$
  - $n^2 \neq \Theta(n)$, because $n^2 \neq O(n)$
  - $\log n \neq \Theta(n)$, because $\log n \neq \Omega(n)$

  - $n^2 = \Theta(n^2)$
  - $5n^2 = \Theta(n^2)$
  - $n \neq \Theta(n^2)$, because $n \neq \Omega(n^2)$
  - $n^3 \neq \Theta(n^2)$, because $n^3 \neq O(n^2)$

Big O

- How running time of an algorithm generally behaves in relation to the input size.
- All functions with same growth rate are characterized using the same Big O notation.
- All functions with same growth rate are considered equivalent in Big O notation.

- If f(N) is a sum of several terms, the highest order term (the one with the fastest growth rate) is kept and others are discarded.

- If f(N) has a term that is a product of several factors, all constants (those that are not in terms of N) are omitted.

Big O

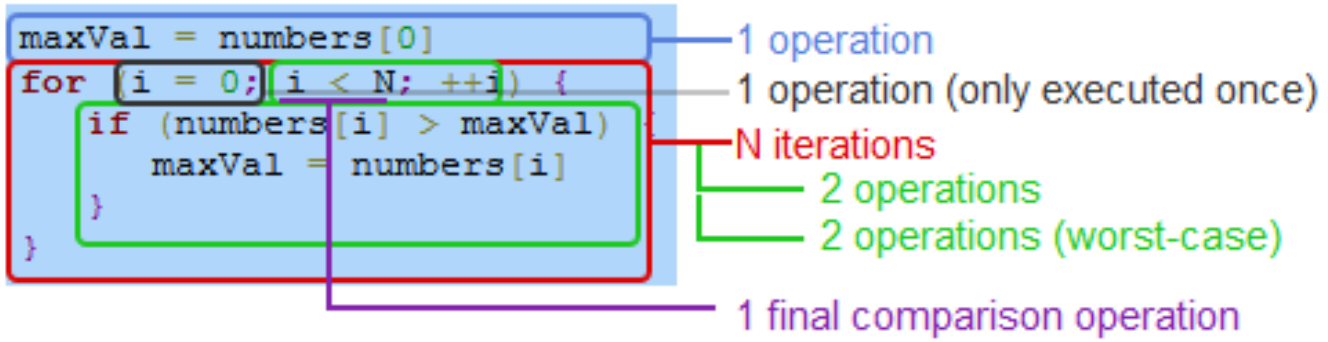
Composite function	Big O notation
$c \cdot O(f(N))$	$O(f(N))$
$c + O(f(N))$	$O(f(N))$
$g(N) \cdot O(f(N))$	$O(g(N) \cdot f(N))$
$g(N) + O(f(N))$	$O(g(N) + f(N))$

Big O

Function	N = 10	N = 50	N = 100	N = 1000	N = 10000	N = 100000
$\log_2 N$	3.3 μs	5.65 μs	6.6 μs	9.9 μs	13.3 μs	16.6 μs
N	10 μs	50 μs	100 μs	1000 μs	10 ms	100 ms
$N \log_2 N$	.03 ms	.28 ms	.66 ms	.099 s	.132 s	1.66 s
N^2	.1 ms	2.5 ms	10 ms	1 s	100 s	2.7 $hours$
N^3	1 ms	.125 s	1 s	16.7 min	11.57 $days$	31.7 $years$
2^n	.001 s	35.7 $years$	$>$ 1000 $years$			

Worst-Case Algorithm Analysis

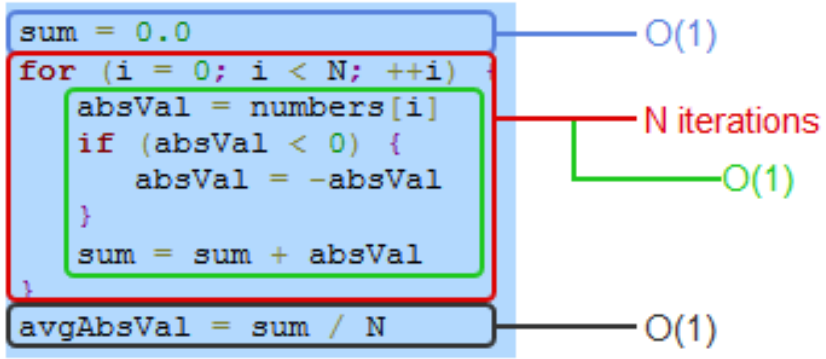
- How runtime of an algorithm scales as the input size increases?
- Determine how many operations the algorithm executes for a specific input size, N.



$$f(N) = 1 + 1 + 1 + N(2 + 2)$$
$$= 3 + 4N$$
$$= O(N)$$

Worst-Case Algorithm Analysis

- Since constants are omitted in big-O notation, any constant number of constant time operations is O(1).



$$O(1) + NO(1) + O(1) = O(N)$$

Worst-Case Algorithm Analysis

```
for (i = 0; i < N; ++i) {
   indexSmallest = i

   for (j = i + 1; j < N; ++j) {
      if (numbers[j] < numbers[indexSmallest])
         indexSmallest = j
      }
   }

   temp = numbers[i]
   numbers[i] = numbers[indexSmallest]
   numbers[indexSmallest] = temp
}
```

4 operations, generalized as constant c

Each of the N outer loop iterations executes 5 operations, generalized as constant d

$$f(N) = c\left((N-1) + (N-2) + \cdots + 2 + 1 + 0\right) = c\left(\frac{N(N-1)}{2}\right) + d \cdot N$$

$$O(f(N)) = O\left(\frac{c}{2}(N^2 - N) + d \cdot N\right) = O(N^2 + N) = O(N^2)$$