Running Time Analysis

• Units of “time”
  • Operations
    • Whichever operations we pick

• How do we express running time?
  • Function
    • Domain (input): size of the input
    • Range: count of operations
Worst Case Running Time Analysis

• If an algorithm has a worst case **running time** of $f(n)$
  • Among all possible size-$n$ inputs, the “worst” one will do $f(n)$ “operations”
  • $f(n)$ gives the maximum count of operations needed from among all inputs of size $n$
Analysis Process From 123/143

• Count the number of “primitive chosen operations”
  • +, -, compare, arr[i], arr.length, etc.
  • Select the operation(s) which:
    • Is/are done the most
    • Is/are the most “expensive”
    • Is/are the most “important”

• Write that count as an expression using \( n \) (the input size)

• Put that expression into a “bucket” by ignoring constants and “non-dominant” terms, then put a \( O() \) around it.
  • \( 4n^2 + 8n - 10 \) ends up as \( O(n^2) \)
  • \( \frac{1}{2} n + 80 \) ends up as \( O(n) \)
Searching in a Sorted List

```java
public static boolean contains(List<Integer> a, int k){
    for(int i=0; i< a.size(); i++){
        if (a.get(i) == k)
            return true;
    }
    return false;
}
```

$O(n)$
Faster way?

Can you think of a faster algorithm to solve this problem?
Binary Search

```java
public static boolean contains(List<Integer> a, int k){
    int start = 0;
    int end = a.size();
    while(start < end){
        int mid = start + (end-start)/2;
        if(a.get(mid) == k)
            return true;
        else if(a.get(mid) < k)
            start = mid+1;
        else
            end = mid;
    }
    return false;
}
```

\[ 2^x = n \]

\[ \log_2 n \]
Why is this \( \log_2 n \)?

• In the beginning: \( \text{end} - \text{start} = n \)
• After 1 iteration: \( \text{end} - \text{start} = \frac{n}{2} \)
  • \( \text{mid} - \text{start} = (\text{start} + (\text{end} - \text{start})/2) - \text{start} \)
  • \( \text{end} - \text{mid} = \text{end} - (\text{start} + (\text{end} - \text{start})/2) \)
• Each iteration cuts the “gap” in half!
• We stop when the gap is 1

\[ n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots, \frac{n}{2^i} \]
Comparing
Comparing Running Times

• Suppose I have these algorithms, all of which have the same input/output behavior:
  • Algorithm A’s worst case running time is $10n + 900$
  • Algorithm B’s worst case running time is $100n - 50$
  • Algorithm C’s worst case running time is $\frac{n^2}{100}$

• Which algorithm is best?
\[ f(n) = O(g(n)) \]

\[ f(n) = \Theta(g(n)) \]

\[ f(n) = \Omega(g(n)) \]
Asymptotic Notation

• $O(g(n))$
  - The set of functions with asymptotic behavior less than or equal to $g(n)$
  - Upper-bounded by a constant times $g$ for large enough values $n$
  - $f \in O(g(n)) \equiv \exists c > 0. \exists n_0 > 0. \forall n \geq n_0. f(n) \leq c \cdot g(n)$

• $\Omega(g(n))$
  - The set of functions with asymptotic behavior greater than or equal to $g(n)$
  - Lower-bounded by a constant times $g$ for large enough values $n$
  - $f \in \Omega(g(n)) \equiv \exists c > 0. \exists n_0 > 0. \forall n \geq n_0. f(n) \geq c \cdot g(n)$

• $\Theta(g(n))$
  - “Tightly” within constant of $g$ for large $n$
  - $\Omega(g(n)) \cap O(g(n))$
Idea of $\Theta$

• $x = y$
  - $x \leq y \land x \geq y$
Asymptotic Notation Example

• Show: $10n + 100 \in O(n^2)$
  • **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n > n_0. 10n + 100 \leq c \cdot n^2$
  • **Proof:**

$$c = 10$$
$$n_0 = 1$$

$$10n + 100 \leq c \cdot n^2$$
$$10n + 100 \leq 10n^2$$
$$0 \leq 10n^2 - 10n - 100$$
$$0 \leq 10(n^2 - n - 10)$$
$$0 \leq n^2 - n - 10 \leq n^2$$
Asymptotic Notation Example

• Show: $10n + 100 \in O(n^2)$
  
  • **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. \ 10n + 100 \leq c \cdot n^2$
  
  • **Proof:** Let $c = 10$ and $n_0 = 6$. Show $\forall n \geq 6. 10n + 100 \leq 10n^2$

    
    $10n + 100 \leq 10n^2$
    
    $\equiv n + 10 \leq n^2$
    
    $\equiv 10 \leq n^2 - n$
    
    $\equiv 10 \leq n(n - 1)$

    This is True because $n(n - 1)$ is strictly increasing and $6(6 - 1) > 10$
Asymptotic Notation Example

• Show: $13n^2 - 50n \in \Omega(n^2)$
  • **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. 13n^2 - 50n \geq c \cdot n^2$
  • **Proof:**
    • $c =$
    • $n_0 =$
Asymptotic Notation Example

• Show: $13n^2 - 50n \in \Omega(n^2)$
  • **Technique:** find values $c > 0$ and $n_0 > 0$ such that $\forall n \geq n_0. 13n^2 - 50n \geq c \cdot n^2$
  • **Proof:** let $c = 12$ and $n_0 = 50$. Show $\forall n \geq 50. 13n^2 - 50n \geq 12n^2$
    
    $13n^2 - 50n \geq 12n^2$
    
    $\equiv n^2 - 50n \geq 0$
    
    $\equiv n^2 \geq 50n$
    
    $\equiv n \geq 50$
    
    This is certainly true $\forall n \geq 50$. 
Asymptotic Notation Example

• Show: \( n^2 \notin O(n) \)

• Want to show that there does not exist a pair of \( c \) and \( n_0 \) such that \( \forall n_0 > n. \ n^2 \leq c \cdot n \)
  • No matter what value of \( c \) we choose, eventually \( n^2 > c \cdot n \)
  • Find a value of \( n \) such that \( n^2 > c \cdot n \) that we define in terms of \( c \)
    • \( n^2 > c \cdot n \)
    • \( n > c \)
  • No pair of \( c \) and \( n_0 \) exists, because the inequality fails whenever \( n > c \)
Asymptotic Notation Example

• To Show: $n^2 \notin O(n)$
  • Technique: Contradiction
  • Proof: Assume $n^2 \in O(n)$. Then $\exists c, n_0 > 0$ s.t. $\forall n > n_0, n^2 \leq cn$

Let us derive constant $c$. For all $n > n_0 > 0$, we know:

$cn \geq n^2$,
$c \geq n$.

Since $c$ is lower bounded by $n$, $c$ cannot be a constant and make this True.
Contradiction. Therefore $n^2 \notin O(n)$.

Proof by Contradiction!
Gaining Intuition

• When doing asymptotic analysis of functions:
  • If multiple expressions are added together, ignore all but the “biggest”
    • If $f(n)$ grows asymptotically faster than $g(n)$, then $f(n) + g(n) \in \Theta(f(n))$
  • Ignore all multiplicative constants
    • $f(n) + c \in \Theta(f(n))$ for any constant $c \in \mathbb{R}$
  • Ignore bases of logarithms
  • Do NOT ignore:
    • Non-multiplicative and non-additive constants (e.g. in exponents, bases of exponents)
    • Logarithms themselves

• Examples:
  • $4n + 5$
  • $0.5n \log n + 2n + 7$
  • $n^3 + 2^n + 3n$
  • $n \log(10n^2)$
More Examples

• Is each of the following True or False?
  • $4 + 3n \in O(n)$
  • $n + 2 \log n \in O(\log n)$
  • $\log n + 2 \in O(1)$
  • $n^{50} \in O(1.1^n)$
  • $3^n \in \Theta(2^n)$
Common Categories

• $O(1)$  “constant”
• $O(\log n)$ “logarithmic”
• $O(n)$  “linear”
• $O(n \log n)$ “log-linear”
• $O(n^2)$ “quadratic”
• $O(n^3)$ “cubic”
• $O(n^k)$ “polynomial”
• $O(k^n)$ “exponential”
Defining your running time function

- **Worst-case complexity:**
  - max number of steps algorithm takes on “most challenging” input

- **Best-case complexity:**
  - min number of steps algorithm takes on “easiest” input

- **Average/expected complexity:**
  - avg number of steps algorithm takes on random inputs (context-dependent)

- **Amortized complexity:**
  - max total number of steps algorithm takes on M “most challenging” consecutive inputs, divided by M (i.e., divide the max total sum by M).
Beware!

- Worst case, Best case, amortized are ways to select a function
- $O$, $\Omega$, $\Theta$ are ways to compare functions
- You can mix and match!
- The following statements totally make sense!
  - The worst case running time of my algorithm is $\Omega(n^3)$
  - The best case running time of my algorithm is $O(n)$
  - The best case running time of my algorithm is $\Theta(2^n)$