Computational Complexity

Time and Space Complexity
Big O Notation
Examples
Practical Study: Sets
Computational Complexity

Context

Key computational resources:

- Time
- Space
- Energy

Computational complexity is the study of how problem size affects resource consumption for a given implementation.

- Worst case
  - the complexity of solving the problem for the worst input of size $n$

- Average case
  - is the complexity of solving the problem on an average.
(Computational) Scaling

1. Identify $n$, the number that characterizes the problem size.
   - Number of pixels on screen
   - Number of elements to be sorted
   - etc.

2. Study the algorithm to determine how resource consumption changes as a function of $n$. 
Big O Notation

Suppose we have a problem of size $n$ that takes $g(n)$ time to execute in the average case.

We say:

$$g(n) \in O(f(n))$$

if and only if there exists a constant $c > 0$ and a constant $n_0 > 0$ such that for all $n > n_0$:

$$g(n) \leq c \times f(n)$$
Simple Examples

• Constant $O(1)$
  – Time to perform an addition

• Logarithmic $O(\log(n))$
  – Time to find an element in a (balanced) BST

• Linear $O(n)$
  – Time to find an element within a list

• $O(n \log(n))$
  – Average time to sort using mergesort

• Quadratic $O(n^2)$
  – Time to compare $n$ elements with each other
Time Complexity: Counting Statements

Time complexity can be estimated by simply counting the number of statements to be executed.

- **Traps**
  - Simple statements are constant time
  - Library calls may have arbitrary complexity
Concrete Examples

Consider hashing into a table of $n$ elements…

```java
public int hash(Integer key, int buckets) {
    return key % buckets;
}
```

Constant time, $O(1)$
Concrete Examples

Consider summing a list of size $n$...

```java
public int sum(ArrayList<Integer> list) {
    int rtn = 0;
    for (Integer i : list) {
        rtn += i;
    }
    return rtn;
}
```

Linear time, $O(n)$
Concrete Examples

```java
public int minDiff(ArrayList<Integer> values) {
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < values.size(); i++) {
        for (int j = i + 1; j < values.size(); j++) {
            int diff = values.get(i) - values.get(j);
            if (Math.abs(diff) < min) {
                min = Math.abs(diff);
            }
        }
    }
    return min;
}
```

\[
S(N) = 1 + n + 4 \left(\frac{(n-1)n}{2}\right) = 1 + n + 2n \cdot \frac{n^2 - 2n}{2} = 2n^2 - n + 1 \in O(n^2)
\]

Note: \( n-1 + n - 2 + \ldots 2 + 1 = (n-1) \frac{n}{2} \)