Introduction to Algorithm Analysis
AP Computer Science
Imagine yourself at the grocery store. You would like to purchase 15 bottles of soda. There are two likely scenarios here for the cashier to ring your soda up…

Option 1 – She rings each one up separately…
  Can 1 – Total bill increases by 50 cents
  Can 2 – Total bill increases by 50 cents
  Can 3 - Total bill increases by 50 cents
  .............
  Can 15 – Total bill increases by 50 cents

Option 2 – She scans one can and multiplies the price by 15 since there are 15 cans.
In the grocery store, we do not want to wait around for a cashier to inefficiently ring up each can of soda one-by-one.

In Computer Science, we do not want to wait around while a program inefficiently performs calculations in a manner that could have been done more quickly.

We need a formal method of classifying the run-time of code (that is not dependent on the constantly changing speeds of computers).
The O stands for “order”. Order refers to the growth rate of a function. It represents the running time needed to solve the computational steps.
Take a few minutes to write a method that inputs an integer N and calculates the sum of all integers in the range of 1 to N.
How many have a solution that would run in linear time?

```java
public int calculateSum(int N)
{
    int totalSum = 0;
    for(i=1; i<=N; i++)
        totalSum += i;
    return (totalSum);
}
```

This solution is comparable to ringing up each can of soda separately.
A solution that would run in constant time would use the formula: \( \frac{N(N+1)}{2} \)

```java
public static int calculateSum(int N) {
    int totalSum = 0;
    totalSum = (N * (N+1))/2;
    return (totalSum);
}
```

This solution is comparable to ringing up one can of soda to get the price and multiplying the number of cans by that price.
We are always concerned about the dominating term of a function.

When we see one calculation, we classify the Big-O notation as $O(c)$, where the “c” stand for constant.

When we see one loop that performs calculations inside of it, we call that $O(n)$, or linear.

Other examples will later include:
- $O(n^2)$ – Quadratic
- $O(\log(n))$ – Logarithmic
Operations converge to really large n-values
Iterations (loops) contain most operations
Consider the following:
```
for(int i = 0; i < n; i++)
for(int i = 0; i < 10 * n; i++)
```
Both loops above are of order \( n \) (\( n, 10n \))
All first order \( n \) functions converge at approximately the same time
The preceding constant may be ignored
### Nth Degree Functions

Assuming one operation per microsecond, compare approximate run times for input of size 1000 and 100000

<table>
<thead>
<tr>
<th>Order</th>
<th>( n = 1000 )</th>
<th>( n = 100000 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(\log_2 n) )</td>
<td>0.00001 sec</td>
<td>0.000017 sec</td>
</tr>
<tr>
<td>( O(n) )</td>
<td>0.001 sec</td>
<td>0.1 sec</td>
</tr>
<tr>
<td>( O(n \log_2 n) )</td>
<td>0.01 sec</td>
<td>1.7 sec</td>
</tr>
<tr>
<td>( O(n^2) )</td>
<td>1 sec</td>
<td>3 hours</td>
</tr>
<tr>
<td>( O(n^3) )</td>
<td>17 minutes</td>
<td>3200 years</td>
</tr>
<tr>
<td>( O(2^n) )</td>
<td>( 10^{287} ) years</td>
<td>( 10^{10000} ) years</td>
</tr>
</tbody>
</table>
Consider Polynomials

Given the equations:
- \( n^2 + n + 4 \)
  - \( n^2 \) dominates, \( O(n^2) \)
- \( n \log n + n^3 + n \)
  - \( n^3 \) dominates, \( O(n^3) \)
- \( 5n + 2^n + n^2 \)
  - \( 2^n \) dominates, \( O(2^n) \)
for(int i = 1; i < n; i++)
for(int j = 1; j < m; j++)
  .
  .
  .

- $O(n^2)$
- Nested loops with $n \times m$ iterations possible
- Outer loop is $n$; inner loop is $m$ (nearly $n$)
for (k = 1; k <= n; k++)
{
    j = n;
    while (j > 0)
    {
        ...
        j = j / 2;
    }
}

\(O(n \log n)\)

- Outside loop is \(n\), inside loop is halved (\(\log_2 n\))
O(n^3)
- Outer loop is n; inner loop is n*n or n^2
- The step value of 5 is relatively insignificant
Sort and Search Analysis

Average case scenarios for various sorts

<table>
<thead>
<tr>
<th>Sort Type</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Insertions Sort</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Quick Sort</td>
<td>$O(n\log n)$</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>$O(n\log n)$</td>
</tr>
<tr>
<td>Sequential Search</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Binary Search</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>