Algorithm Performance Prediction ("big O")

• In this subdiscipline of Computer Science, we are interested in predicting (not measuring) the variation in running time (or sometimes, space requirements) of an algorithm as the amount of data varies.

• Here we are not interested in the absolute running time (in seconds, say), but rather in the variation of running time.

• Suppose an algorithm takes K seconds to treat 1000 data items. If we double the data to 2000, will the running time be 2K, or 4K, or 10K, or what?

• This variation is sometimes called the growth rate of an algorithm.

• "Big O" stands for "order of magnitude". We are interested in predictive estimation, not exact measurement.
Common Algorithm Growth Rates

• $O(1)$, or constant
• $O(\log N)$, or logarithmic (the logarithm is usually taken to the base 2)
• $O(N)$, or linear (directly proportional to $N$)
• $O(N \log N)$, (usually just called $N \log N$)
• $O(N^2)$, or quadratic (proportional to the square of $N$)
<table>
<thead>
<tr>
<th>$N$</th>
<th>1 (constant)</th>
<th>$\log N$</th>
<th>$N \log N$</th>
<th>$N^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>24</td>
<td>64</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>4</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>5</td>
<td>160</td>
<td>1024</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>6</td>
<td>384</td>
<td>4096</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>7</td>
<td>896</td>
<td>16384</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>8</td>
<td>2048</td>
<td>65536</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
<td>9</td>
<td>4608</td>
<td>262144</td>
</tr>
<tr>
<td>1024</td>
<td>1</td>
<td>10</td>
<td>10240</td>
<td>1048576</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
<td>11</td>
<td>22528</td>
<td>4194304</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
<td>12</td>
<td>49152</td>
<td>16777216</td>
</tr>
<tr>
<td>8192</td>
<td>1</td>
<td>13</td>
<td>106496</td>
<td>67108864</td>
</tr>
<tr>
<td>16384</td>
<td>1</td>
<td>14</td>
<td>229376</td>
<td>268435456</td>
</tr>
<tr>
<td>32768</td>
<td>1</td>
<td>15</td>
<td>491520</td>
<td>1073741824</td>
</tr>
</tbody>
</table>
Estimating Algorithm Growth Rates

• Sequence, or writing one statement below another
• Decision, or the well-known if-then or if-then-else
• Loop, including counting loops and while loops
• Subprogram call
Some Java Control Structures

(a) Sequence

    Temp = A;
    A = B;
    B = Temp;

(b) Decision

    if (x > Max)
    {
        Max = x;
    }

(c) If-then-else

    if (x = y)
    {
        Max = x;
    }
    else
    {
        Max = y;
    }
(d) Counting loop

```c
int x = 0;
for (int i = p; i <= q; i++)
{
    x = x + i;
}
```

(e) While loop

```c
while (x > 0)
{
    y = y + 3;
    x = x / 2;
}
```
Two Simple Counting Loops

(a) Trip count is constant

```java
for (int counter = 1; counter < 5; counter++)
{
    // something with O(1) performance
}
```

(b) Trip count depends on N

```java
for (int counter = 1; counter < N; counter++)
{
    // something with O(1) performance
}
```
Three Doubly Nested Loops

(a) A double counting loop

```java
for (int outerCounter = 1; outerCounter < N; outerCounter++)
{
    for (int innerCounter = 1; innerCounter < N; innerCounter++)
    {
        // something with O(1) performance
    }
}
```

(b) Another double counting loop

```java
for (int outerCounter = 1; outerCounter < N; outerCounter++)
{
    for (int innerCounter = 1; innerCounter < outerCounter; innerCounter++)
    {
        // something with O(1) performance
    }
}
```
(c) Yet another double counting loop

```c
for (int outerCounter = 1; outerCounter < N; outerCounter++)
{
    for (int innerCounter = outerCounter;
         innerCounter < N; innerCounter++)
    {
        // something with O(1) performance
    }
}
```
Two Multiplicatively Controlled Loops

(a) Control is multiplied by 2

```c
int Control = 1;
while (Control <= N)
{
    // something with O(1) performance
    Control = 2 * Control;
}
```

(b) Control is divided by 2

```c
int Control = N;
while (Control >= 1)
{
    // something with O(1) performance
    Control = Control / 2;
}
```
Two N log N Loop Structures

```c
int Control = 1;
for (int Counter = 1; Counter <= N; Counter++)
{
    while (Control <= N)
    {
        // something with O(1) performance
        Control = 2 * Control;
    }
}

int Control = N;
while (Control >= 1)
{
    for (int Counter = 1; Counter <= N; Counter++)
    {
        // something with O(1) performance
    }
    Control = Control / 2;
}