Growth of Functions and Asymptotic Notation

- When we study algorithms, we are interested in characterizing them according to their efficiency.
- We are usually interested in the order of growth of the running time of an algorithm, not in the exact running time. This is also referred to as the asymptotic running time.
- We need to develop a way to talk about rate of growth of functions so that we can compare algorithms.
- Asymptotic notation gives us a method for classifying functions according to their rate of growth.
Big-O Notation

- **Definition:** \( f(n) = O(g(n)) \) iff there are two positive constants \( c \) and \( n_0 \) such that
  \[
  |f(n)| \leq c |g(n)| \text{ for all } n \geq n_0
  \]
- If \( f(n) \) is nonnegative, we can simplify the last condition to
  \[
  0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0
  \]
- We say that “\( f(n) \) is big-O of \( g(n) \).”
- As \( n \) increases, \( f(n) \) grows no faster than \( g(n) \). In other words, \( g(n) \) is an *asymptotic upper bound* on \( f(n) \).
**Example:** $n^2 + n = O(n^3)$

**Proof:**

- Here, we have $f(n) = n^2 + n$, and $g(n) = n^3$
- Notice that if $n \geq 1$, $n \leq n^3$ is clear.
- Also, notice that if $n \geq 1$, $n^2 \leq n^3$ is clear.
- **Side Note:** In general, if $a \leq b$, then $n^a \leq n^b$ whenever $n \geq 1$. This fact is used often in these types of proofs.
- Therefore,
  $$n^2 + n \leq n^3 + n^3 = 2n^3$$
- We have just shown that
  $$n^2 + n \leq 2n^3 \text{ for all } n \geq 1$$
- Thus, we have shown that $n^2 + n = O(n^3)$
  (by definition of Big-$O$, with $n_0 = 1$, and $c = 2$.)
**Big-Ω notation**

- **Definition:** $f(n) = \Omega(g(n))$ iff there are two positive constants $c$ and $n_0$ such that
  
  $$|f(n)| \geq c|g(n)| \text{ for all } n \geq n_0$$

- If $f(n)$ is nonnegative, we can simplify the last condition to
  
  $$0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0$$

- We say that “$f(n)$ is omega of $g(n)$.”

- As $n$ increases, $f(n)$ grows no slower than $g(n)$. In other words, $g(n)$ is an asymptotic lower bound on $f(n)$. 
Example: \( n^3 + 4n^2 = \Omega(n^2) \)

Proof:

- Here, we have \( f(n) = n^3 + 4n^2 \), and \( g(n) = n^2 \).
- It is not too hard to see that if \( n \geq 0 \),
  \[
  n^3 \leq n^3 + 4n^2
  \]
- We have already seen that if \( n \geq 1 \),
  \[
  n^2 \leq n^3
  \]
- Thus when \( n \geq 1 \),
  \[
  n^2 \leq n^3 \leq n^3 + 4n^2
  \]
- Therefore,
  \[
  1n^2 \leq n^3 + 4n^2 \text{ for all } n \geq 1
  \]
- Thus, we have shown that \( n^3 + 4n^2 = \Omega(n^2) \)
  (by definition of Big-\(\Omega\), with \( n_0 = 1 \), and \( c = 1 \)).
Big-$\Theta$ notation

- **Definition:** $f(n) = \Theta(g(n))$ iff there are three positive constants $c_1$, $c_2$ and $n_0$ such that
  
  $$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)| \text{ for all } n \geq n_0$$

- If $f(n)$ is nonnegative, we can simplify the last condition to
  
  $$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

- We say that “$f(n)$ is theta of $g(n)$.”

- As $n$ increases, $f(n)$ grows at the same rate as $g(n)$. In other words, $g(n)$ is an asymptotically tight bound on $f(n)$. 

![Graph showing the relationship between $f(n)$, $c_1 g(n)$, $c_2 g(n)$, and $n_0$.](image-url)
Example: $n^2 + 5n + 7 = \Theta(n^2)$

Proof:

- When $n \geq 1$,
  \[ n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2 \]
- When $n \geq 0$,
  \[ n^2 \leq n^2 + 5n + 7 \]
- Thus, when $n \geq 1$
  \[ 1n^2 \leq n^2 + 5n + 7 \leq 13n^2 \]

Thus, we have shown that $n^2 + 5n + 7 = \Theta(n^2)$  
(by definition of Big-Θ, with $n_0 = 1$, $c_1 = 1$, and $c_2 = 13$.)
Arithmetic of Big-O, $\Omega$, and $\Theta$ notations

• Transitivity:
  
  - $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
  
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  
  - $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

• Scaling: if $f(n) = O(g(n))$ then for any $k > 0$, $f(n) = O(kg(n))$

• Sums: if $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then 
  $(f_1 + f_2)(n) = O(\max(g_1(n), g_2(n)))$
Strategies for Big-O

- Sometimes the easiest way to prove that \( f(n) = O(g(n)) \) is to take \( c \) to be the sum of the positive coefficients of \( f(n) \).

- We can usually ignore the negative coefficients. Why?

- **Example:** To prove \( 5n^2 + 3n + 20 = O(n^2) \), we pick \( c = 5 + 3 + 20 = 28 \). Then if \( n \geq n_0 = 1 \),
  \[
  5n^2 + 3n + 20 \leq 5n^2 + 3n^2 + 20n^2 = 28n^2,
  \]
  thus \( 5n^2 + 3n + 20 = O(n^2) \).

- This is not always so easy. How would you show that \((\sqrt{2})^{\log n} + \log^2 n + n^4\) is \( O(2^n) \)? Or that \( n^2 = O(n^2 - 13n + 23) \)? After we have talked about the relative rates of growth of several functions, this will be easier.

- In general, we simply (or, in some cases, with much effort) find values \( c \) and \( n_0 \) that work. This gets easier with practice.
Strategies for $\Omega$ and $\Theta$

- Proving that a $f(n) = \Omega(g(n))$ often requires more thought.
  - Quite often, we have to pick $c < 1$.
  - A good strategy is to pick a value of $c$ which you think will work, and determine which value of $n_0$ is needed.
  - Being able to do a little algebra helps.
  - We can sometimes simplify by ignoring terms if $f(n)$ with the positive coefficients. Why?

- The following theorem shows us that proving $f(n) = \Theta(g(n))$ is nothing new:
  - **Theorem:** $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
  - Thus, we just apply the previous two strategies.

- We will present a few more examples using a several different approaches.
Show that $\frac{1}{2}n^2 + 3n = \Theta(n^2)$

Proof:

• Notice that if $n \geq 1$,
  
  \[ \frac{1}{2}n^2 + 3n \leq \frac{1}{2}n^2 + 3n^2 = \frac{7}{2}n^2 \]

• Thus,
  
  \[ \frac{1}{2}n^2 + 3n = O(n^2) \]

• Also, when $n \geq 0$,
  
  \[ \frac{1}{2}n^2 \leq \frac{1}{2}n^2 + 3n \]

• So
  
  \[ \frac{1}{2}n^2 + 3n = \Omega(n^2) \]

• Since $\frac{1}{2}n^2 + 3n = O(n^2)$ and $\frac{1}{2}n^2 + 3n = \Omega(n^2)$,

  \[ \frac{1}{2}n^2 + 3n = \Theta(n^2) \]
**Show that** \( (n \log n - 2n + 13) = \Omega(n \log n) \)

**Proof:** We need to show that there exist positive constants \( c \) and \( n_0 \) such that

\[
0 \leq c \, n \log n \leq n \log n - 2n + 13 \quad \text{for all } n \geq n_0.
\]

Since \( n \log n - 2n \leq n \log n - 2n + 13 \), we will instead show that

\[
c \, n \log n \leq n \log n - 2n,
\]

which is equivalent to

\[
c \leq 1 - \frac{2}{\log n}, \quad \text{when } n > 1.
\]

If \( n \geq 8 \), then \( 2/(\log n) \leq 2/3 \), and picking \( c = 1/3 \) suffices. Thus if \( c = 1/3 \) and \( n_0 = 8 \), then for all \( n \geq n_0 \), we have

\[
0 \leq c \, n \log n \leq n \log n - 2n \leq n \log n - 2n + 13.
\]

Thus \( (n \log n - 2n + 13) = \Omega(n \log n) \).
Show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Proof:

- We need to find positive constants $c_1$, $c_2$, and $n_0$ such that
  \[0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ for all } n \geq n_0\]

- Dividing by $n^2$, we get
  \[0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2\]

- $c_1 \leq \frac{1}{2} - \frac{3}{n}$ holds for $n \geq 10$ and $c_1 = 1/5$
- $\frac{1}{2} - \frac{3}{n} \leq c_2$ holds for $n \geq 10$ and $c_2 = 1$.

- Thus, if $c_1 = 1/5$, $c_2 = 1$, and $n_0 = 10$, then for all $n \geq n_0$,
  \[0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \text{ for all } n \geq n_0.\]

Thus we have shown that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$. 
Asymptotic Bounds and Algorithms

- In all of the examples so far, we have assumed we knew the exact running time of the algorithm.
- In general, it may be very difficult to determine the exact running time.
- Thus, we will try to determine a bounds without computing the exact running time.
- **Example:** What is the complexity of the following algorithm?

```
for (i = 0; i < n; i ++)
    for (j = 0; j < n; j ++)
        a[i][j] = b[i][j] * x;
```

**Answer:** $O(n^2)$
- We will see more examples later.
Summary of the Notation

- \( f(n) = O(g(n)) \Rightarrow f \preceq g \)
- \( f(n) = \Omega(g(n)) \Rightarrow f \succeq g \)
- \( f(n) = \Theta(g(n)) \Rightarrow f \approx g \)

- It is important to remember that a Big-O bound is only an upper bound. So an algorithm that is \( O(n^2) \) might not ever take that much time. It may actually run in \( O(n) \) time.

- Conversely, an \( \Omega \) bound is only a lower bound. So an algorithm that is \( \Omega(n \log n) \) might actually be \( \Theta(2^n) \).

- Unlike the other bounds, a \( \Theta \)-bound is precise. So, if an algorithm is \( \Theta(n^2) \), it runs in quadratic time.
Common Rates of Growth

In order for us to compare the efficiency of algorithms, we need to know some common growth rates, and how they compare to one another. This is the goal of the next several slides.

Let \( n \) be the size of input to an algorithm, and \( k \) some constant. The following are common rates of growth.

- **Constant**: \( \Theta(k) \), for example \( \Theta(1) \)
- **Linear**: \( \Theta(n) \)
- **Logarithmic**: \( \Theta(\log_k n) \)
- **\( n \log n \)**: \( \Theta(n \log_k n) \)
- **Quadratic**: \( \Theta(n^2) \)
- **Polynomial**: \( \Theta(n^k) \)
- **Exponential**: \( \Theta(k^n) \)

We’ll take a closer look at each of these classes.
Classification of algorithms - $\Theta(1)$

- Operations are performed $k$ times, where $k$ is some constant, independent of the size of the input $n$.
- This is the best one can hope for, and most often unattainable.
- **Examples:**

```c
int Fifth_Element(int A[], int n) {
    return A[5];
}

int Partial_Sum(int A[], int n) {
    int sum = 0;
    for (int i = 0; i < 42; i++)
        sum = sum + A[i];
    return sum;
}
```
Classification of algorithms - $\Theta(n)$

- Running time is linear
- As $n$ increases, run time increases in proportion
- Algorithms that attain this look at each of the $n$ inputs at most some constant $k$ times.

**Examples:**

```c
void sum_first_n(int n) {
    int i,sum=0;
    for (i=1;i<=n;i++)
        sum = sum + i;
}

void m_sum_first_n(int n) {
    int i,k,sum=0;
    for (i=1;i<=n;i++)
        for (k=1;k<7;k++)
            sum = sum + i;
}
```
Classification of algorithms - $\Theta(\log n)$

- A logarithmic function is the inverse of an exponential function, i.e. $b^x = n$ is equivalent to $x = \log_b n$.
- Always increases, but at a slower rate as $n$ increases. (Recall that the derivative of $\log n$ is $\frac{1}{n}$, a decreasing function.)
- Typically found where the algorithm can systematically ignore fractions of the input.
- **Examples:**

```c
int binarysearch(int a[], int n, int val)
{
    int l=1, r=n, m;
    while (r>=1) {
        m = (l+r)/2;
        if   (a[m]==val) return m;
        if   (a[m]>val) r=m-1;
        else l=m+1; }
    return -1;
}
```
Classification of algorithms - $\Theta(n \log n)$

- Combination of $O(n)$ and $O(\log n)$
- Found in algorithms where the input is recursively broken up into a constant number of subproblems of the same type which can be solved independently of one another, followed by recombining the sub-solutions.
- **Example:** Quicksort is $O(n \log n)$.

Perhaps now is a good time for a reminder that when speaking asymptotically, the base of logarithms is irrelevant. This is because of the identity

$$\log_a b \log_b n = \log_a n.$$
Classification of algorithms - \( \Theta(n^2) \)

- We call this class quadratic.
- As \( n \) doubles, run-time quadruples.
- However, it is still polynomial, which we consider to be good.
- Typically found where algorithms deal with all pairs of data.

- **Example:**

  ```c
  int *compute_sums(int A[], int n) {
    int M[n][n];
    int i, j;
    for (i=0; i<n; i++)
      for (j=0; j<n; j++)
    return M;
  }
  ```

- More generally, if an algorithm is \( \Theta(n^k) \) for constant \( k \) it is called a polynomial-time algorithm.
Classification of algorithms - $\Theta(2^n)$

- We call this class exponential.
- This class is, essentially, as bad as it gets.
- Algorithms that use brute force are often in this class.
- Can be used only for small values of $n$ in practice.
- **Example:** A simple way to determine all $n$ bit numbers whose binary representation has $k$ non-zero bits is to run through all the numbers from 1 to $2^n$, incrementing a counter when a number has $k$ nonzero bits. It is clear this is exponential in $n$. 
# Comparison of growth rates

<table>
<thead>
<tr>
<th>log n</th>
<th>n</th>
<th>n log n</th>
<th>n²</th>
<th>n³</th>
<th>2ⁿ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6931</td>
<td>2</td>
<td>1.39</td>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>1.099</td>
<td>3</td>
<td>3.30</td>
<td>9</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>1.386</td>
<td>4</td>
<td>5.55</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>1.609</td>
<td>5</td>
<td>8.05</td>
<td>25</td>
<td>125</td>
<td>32</td>
</tr>
<tr>
<td>1.792</td>
<td>6</td>
<td>10.75</td>
<td>36</td>
<td>216</td>
<td>64</td>
</tr>
<tr>
<td>1.946</td>
<td>7</td>
<td>13.62</td>
<td>49</td>
<td>343</td>
<td>128</td>
</tr>
<tr>
<td>2.079</td>
<td>8</td>
<td>16.64</td>
<td>64</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>2.197</td>
<td>9</td>
<td>19.78</td>
<td>81</td>
<td>729</td>
<td>512</td>
</tr>
<tr>
<td>2.303</td>
<td>10</td>
<td>23.03</td>
<td>100</td>
<td>1000</td>
<td>1024</td>
</tr>
<tr>
<td>2.398</td>
<td>11</td>
<td>26.38</td>
<td>121</td>
<td>1331</td>
<td>2048</td>
</tr>
<tr>
<td>2.485</td>
<td>12</td>
<td>29.82</td>
<td>144</td>
<td>1728</td>
<td>4096</td>
</tr>
<tr>
<td>2.565</td>
<td>13</td>
<td>33.34</td>
<td>169</td>
<td>2197</td>
<td>8192</td>
</tr>
<tr>
<td>2.639</td>
<td>14</td>
<td>36.95</td>
<td>196</td>
<td>2744</td>
<td>16384</td>
</tr>
<tr>
<td>2.708</td>
<td>15</td>
<td>40.62</td>
<td>225</td>
<td>3375</td>
<td>32768</td>
</tr>
<tr>
<td>2.773</td>
<td>16</td>
<td>44.36</td>
<td>256</td>
<td>4096</td>
<td>65536</td>
</tr>
<tr>
<td>2.833</td>
<td>17</td>
<td>48.16</td>
<td>289</td>
<td>4913</td>
<td>131072</td>
</tr>
<tr>
<td>2.890</td>
<td>18</td>
<td>52.03</td>
<td>324</td>
<td>5832</td>
<td>262144</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>log log m</th>
<th>log m</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## More growth rates

<table>
<thead>
<tr>
<th>$n$</th>
<th>$100n$</th>
<th>$n^2$</th>
<th>$11n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>4</td>
<td>44</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>9</td>
<td>99</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>16</td>
<td>176</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>25</td>
<td>275</td>
<td>125</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>600</td>
<td>36</td>
<td>396</td>
<td>216</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>700</td>
<td>49</td>
<td>539</td>
<td>343</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>800</td>
<td>64</td>
<td>704</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>900</td>
<td>81</td>
<td>891</td>
<td>729</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>100</td>
<td>1100</td>
<td>1000</td>
<td>1024</td>
</tr>
<tr>
<td>11</td>
<td>1100</td>
<td>121</td>
<td>1331</td>
<td>1331</td>
<td>2048</td>
</tr>
<tr>
<td>12</td>
<td>1200</td>
<td>144</td>
<td>1584</td>
<td>1728</td>
<td>4096</td>
</tr>
<tr>
<td>13</td>
<td>1300</td>
<td>169</td>
<td>1859</td>
<td>2197</td>
<td>8192</td>
</tr>
<tr>
<td>14</td>
<td>1400</td>
<td>196</td>
<td>2156</td>
<td>2744</td>
<td>16384</td>
</tr>
<tr>
<td>15</td>
<td>1500</td>
<td>225</td>
<td>2475</td>
<td>3375</td>
<td>32768</td>
</tr>
<tr>
<td>16</td>
<td>1600</td>
<td>256</td>
<td>2816</td>
<td>4096</td>
<td>65536</td>
</tr>
<tr>
<td>17</td>
<td>1700</td>
<td>289</td>
<td>3179</td>
<td>4913</td>
<td>131072</td>
</tr>
<tr>
<td>18</td>
<td>1800</td>
<td>324</td>
<td>3564</td>
<td>5832</td>
<td>262144</td>
</tr>
<tr>
<td>19</td>
<td>1900</td>
<td>361</td>
<td>3971</td>
<td>6859</td>
<td>524288</td>
</tr>
</tbody>
</table>
## More growth rates

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^2 - n$</th>
<th>$n^2 + 99$</th>
<th>$n^3$</th>
<th>$n^3 + 234$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>103</td>
<td>8</td>
<td>242</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>30</td>
<td>135</td>
<td>216</td>
<td>450</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>90</td>
<td>199</td>
<td>1000</td>
<td>1234</td>
</tr>
<tr>
<td>14</td>
<td>196</td>
<td>182</td>
<td>295</td>
<td>2744</td>
<td>2978</td>
</tr>
<tr>
<td>18</td>
<td>324</td>
<td>306</td>
<td>423</td>
<td>5832</td>
<td>6066</td>
</tr>
<tr>
<td>22</td>
<td>484</td>
<td>462</td>
<td>583</td>
<td>10648</td>
<td>10882</td>
</tr>
<tr>
<td>26</td>
<td>676</td>
<td>650</td>
<td>775</td>
<td>17576</td>
<td>17810</td>
</tr>
<tr>
<td>30</td>
<td>900</td>
<td>870</td>
<td>999</td>
<td>27000</td>
<td>27234</td>
</tr>
<tr>
<td>34</td>
<td>1156</td>
<td>1122</td>
<td>1255</td>
<td>39304</td>
<td>39538</td>
</tr>
<tr>
<td>38</td>
<td>1444</td>
<td>1406</td>
<td>1543</td>
<td>54872</td>
<td>55106</td>
</tr>
<tr>
<td>42</td>
<td>1764</td>
<td>1722</td>
<td>1863</td>
<td>74088</td>
<td>74322</td>
</tr>
<tr>
<td>46</td>
<td>2116</td>
<td>2070</td>
<td>2215</td>
<td>97336</td>
<td>97570</td>
</tr>
<tr>
<td>50</td>
<td>2500</td>
<td>2450</td>
<td>2599</td>
<td>125000</td>
<td>125234</td>
</tr>
<tr>
<td>54</td>
<td>2916</td>
<td>2862</td>
<td>3015</td>
<td>157464</td>
<td>157698</td>
</tr>
<tr>
<td>58</td>
<td>3364</td>
<td>3306</td>
<td>3463</td>
<td>195112</td>
<td>195346</td>
</tr>
<tr>
<td>62</td>
<td>3844</td>
<td>3782</td>
<td>3943</td>
<td>238328</td>
<td>238562</td>
</tr>
<tr>
<td>66</td>
<td>4356</td>
<td>4290</td>
<td>4455</td>
<td>287496</td>
<td>287730</td>
</tr>
<tr>
<td>70</td>
<td>4900</td>
<td>4830</td>
<td>4999</td>
<td>343000</td>
<td>343234</td>
</tr>
<tr>
<td>74</td>
<td>5476</td>
<td>5402</td>
<td>5575</td>
<td>405224</td>
<td>405458</td>
</tr>
</tbody>
</table>
Polynomial Functions

Asymptotic Notation
Slow Growing Functions

- $\log(x)$
- $x$
- $x\log(x)$
- $x^2$
Fast Growing Functions Part 1

- \(x\)
- \(x^3\)
- \(x^4\)
- \(2^x\)
Why Constants and Non-Leading Terms Don’t Matter

- $1000000x$
- $300000x^2 + 300x$
- $2^x$
Classification Summary

We have seen that when we analyze functions asymptotically:

- Only the leading term is important.
- Constants don’t make a significant difference.
- The following inequalities hold asymptotically:

\[ c < \log n < \log^2 n < \sqrt{n} < n < n \log n \]
\[ n < n \log n < n^{(1.1)} < n^2 < n^3 < n^4 < 2^n \]

- In other words, an algorithm that is \( \Theta(n \log(n)) \) is more efficient than an algorithm that is \( \Theta(n^3) \).