CS 2230
CS II: Data structures

Meeting 11: Asymptotic analysis
Brandon Myers
University of Iowa
Today’s big ideas

• Algorithms can be evaluated based on efficiency in time and space

• Time for an algorithm to finish can be expressed as a function $f$ of input size $n$

• We usually focus on the big picture by looking at the growth rate of $f(n)$
  • so...we don’t always need to run timing experiments!
Let’s start with an experiment

**Question:** how long does it take to insert an integer into a sorted array?

**Hypothesis:** the average time to insert an integer will increase with the size of the array

**Methodology:**
1. create random sorted array of size N
2. insert T integers into the sorted array, timing each insert
3. repeat experiment for more values of N
Code to run the experiment

// create an sorted array of N random integers; N given by command line argument
System.out.println("creating and sorting array");
final int N = Integer.parseInt(args[0]);
final Random r = new Random();
final int[] array = r.ints(N).toArray();
Arrays.sort(array);

// generate 1000 random integers to insert into the sorted array
System.out.println("creating test inputs");
final int T = 1000;
final int[] trialElements = r.ints(T).toArray();
final long[] results = new long[T];

// run 1000 experiments
System.out.println("running experiments");
for (int t=0; t<T; t++) {
    long trial_start = System.nanoTime();
    // always just insert at the end
    insertSorted(array, N-1, trialElements[t]);
    long trial_end = System.nanoTime();
    results[t] = trial_end - trial_start;
}
Results

- Time (sec) to insert 1 element
- N: the size of sorted array
- worst, avg, best

Graph showing the time taken to insert 1 element into a sorted array of various sizes, with markers for worst, avg, and best cases.
What is the **main cause** of the difference between worst, average, and best case points for a given value of N?

a) the integer inserted in a trial of the experiment may take longer or shorter depending on its value
b) larger integers take up more storage space in the computer so they take longer to swap through the array
c) variability in the computer while it executes each trial causes some trials to take different amounts of time
d) longer arrays take longer to insert elements into

[https://b.socrative.com/login/student/](https://b.socrative.com/login/student/) room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Can we find an analytical model for worst/avg/best case time as a function of N?

\[ \text{worstTime}(N) = ? \]
\[ \text{avgTime}(N) = ? \]
\[ \text{bestTime}(N) = ? \]

\( c = \text{time it takes to do 1 swap} \)
Can we find an analytical model for worst/avg/best case time as a function of N?

\[
\begin{align*}
\text{worstTime}(N) &= c \times N \\
\text{avgTime}(N) &= ? \\
\text{bestTime} &= ?
\end{align*}
\]

\(c = \text{time it takes to do 1 swap}\)
Can we find an analytical model for worst/avg/best case time as a function of N?

\[ \text{worstTime}(N) = c \times N \]
\[ \text{avgTime}(N) = \frac{c}{2} \times (N + 1) \]
\[ \text{bestTime}(N) = ? \]

\( c \) = time it takes to do 1 swap
Can we find an analytical model for worst/avg/best case time as a function of N?

\[ \text{worstTime}(N) = c \times N \]

\[ \text{avgTime}(N) = \frac{c}{2} \times (N + 1) \]

\[ \text{bestTime}(N) = c \times 1 \]
Common functions in algorithm analysis

<table>
<thead>
<tr>
<th>constant</th>
<th>logarithm</th>
<th>linear</th>
<th>n log n</th>
<th>quadratic</th>
<th>cubic</th>
<th>polynomial</th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>log ( n )</td>
<td>( n )</td>
<td>( n \log n )</td>
<td>( n^2 )</td>
<td>( n^3 )</td>
<td>( n^d )</td>
<td>( a^n )</td>
</tr>
</tbody>
</table>

\( a > 1 \)

from Goodrich, Tamassia, Goldwasser
Common functions in algorithm analysis

<table>
<thead>
<tr>
<th>constant</th>
<th>logarithm</th>
<th>linear</th>
<th>n log n</th>
<th>quadratic</th>
<th>cubic</th>
<th>polynomial</th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>log (n)</td>
<td>(n)</td>
<td>(n \log n)</td>
<td>(n^2)</td>
<td>(n^3)</td>
<td>(n^d)</td>
<td>(a^n)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{bestTime}(N) &= c \times 1 \\
\text{worstTime}(N) &= c \times N \\
\text{avgTime}(N) &= \frac{c}{2} \times (N + 1)
\end{align*}
\]

classifying the running time models from our experiment from Goodrich, Tamassia, Goldwasser
**Growth rate**

<table>
<thead>
<tr>
<th>Function</th>
<th>Value for N=1</th>
<th>Value for N=20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x) = 50x$</td>
<td>50</td>
<td>1000</td>
</tr>
<tr>
<td>$f(x) = x^3$</td>
<td>1</td>
<td>8000</td>
</tr>
<tr>
<td>$f(x) = 2^x$</td>
<td>2</td>
<td>1,048,576</td>
</tr>
</tbody>
</table>

![Exponential Growth Chart](https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/Exponential.svg/200px-Exponential.svg.png)
Order these functions from slowest to fastest growing:

A. $4n^2 + n\log(n^2)$
B. $2n^3 + n^2 + n\log(n)$
C. $3n^2 + 500n$
D. $n! + n^2$
E. $2^n + 4n^3$
F. $n^2\log(n)$
G. $n\log^2(n)$
Examples

• Problems with running time **linear** in N
  • search an unsorted array of size N for a value

• Problems with running time **quadratic** in N
  • sort N students using our `insertSorted` algorithm (put new element at end of array and swap into sorted position)

• Problems with running time **exponential** in N
  • find the quickest route for a UPS truck with N packages to deliver to the destinations
Today’s big ideas

• Algorithms can be evaluated based on efficiency in time and space

• Time for an algorithm to finish can be expressed as a function $f$ of input size $n$

• We usually focus on the big picture by looking at the growth rate of $f(n)$