ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN JAVA, GOODRICH, TAMASSIA AND GOLDWASSER (WILEY 2016)
ANALYSIS OF ALGORITHMS (CH 4.2-4.3)
RUNTIME ANALYSIS
BIG-OH

• Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants $c$ and $n_0$ such that $f(n) \leq cg(n)$ for $n \geq n_0$
  • $f(n)$ is the real (measured) time

• We need to know how to determine $f(n)$, $c$, and $n_0$
  • This is all done through experiments
DETERMINING $f(n)$

- Vary the size of the input and then determine runtime using `System.nanoTime()`

```java
1. for(int n = 2; n < MAX; n*=2) {
2.   int r = max(10, MAX/n); //number of repetitions
3.   long start = System.nanoTime();
4.   for(int k = 0; k < r; ++k)
5.     executeFunction();
6.   long stop = System.nanoTime();
7.   double elapsed = (stop - start)/1.e9/r;
8. }
```
DETERMINE $c$ AND $n_0$

• First plot $f(n)$ – time vs size

• Second plot $\frac{f(n)}{g(n)}$ or $\frac{\text{time}}{\text{theoretical time}}$ vs size

• Look for where the data levels off. This will be $n_0$

• Look for the largest value to the right of $n_0$, this will be $c$
TOGETHER – TIME LINEAR SEARCH

• We will download and modify Timing.java for this activity (see Programming Assignment 3)
WHY GO THROUGH THIS ANALYSIS?

• If two algorithms have the same theoretical analysis, we must compare them experimentally!
  • The algorithm with a smaller $c$ value is more efficient

• Determining the $n_0$ informs us:
  • When the theoretical complexity begins holding true

• If you reach the memory limit of the machine, you will see "odd" effects…
ACTIVITY

• Determine big-oh constants for Arrays.sort();
• Theoretical complexity will be $O(n \log n)$