ALGORITHM ANALYSIS

cs2420 | Introduction to Algorithms and Data Structures | Spring 2015

administrivia…

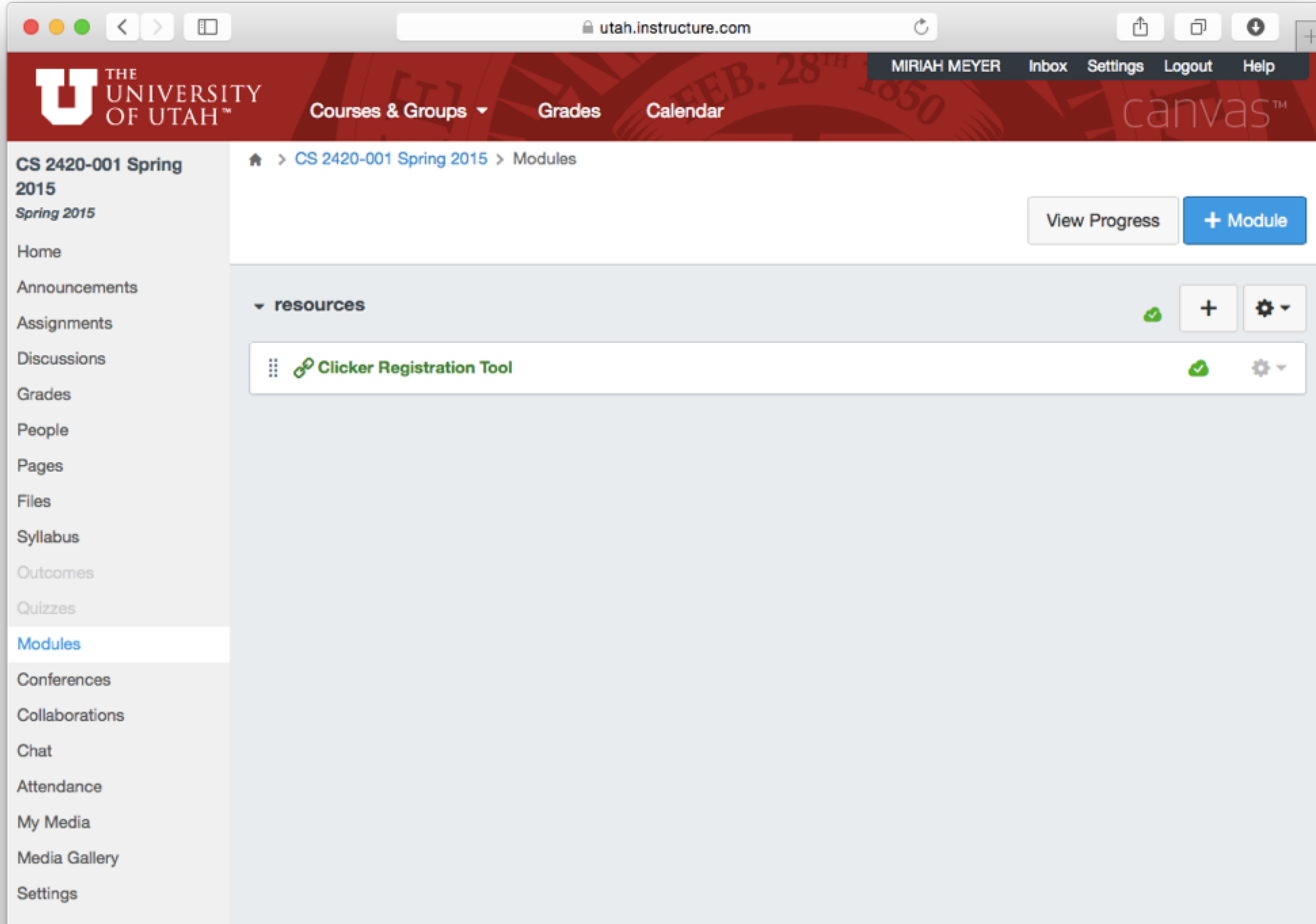
-assignment 2 is due Friday at midnight

  -note change in due date, and time

-tutoring experiment

HTTP://DOODLE.COM/89CBB4U5N5ACY9AG

CLICKERS!!!


utah.instructure.com
THE UNIVERSITY OF UTAH
Courses & Groups
Grades
Calendar
MIRIAH MEYER
Inbox
Settings
Logout
Help
canvas
CS 2420-001 Spring 2015
Spring 2015
Home
Announcements
Assignments
Discussions
Grades
People
Pages
Files
Syllabus
Outcomes
Quizzes
Modules
Conferences
Collaborations
Chat
Attendance
My Media
Media Gallery
Settings
CS 2420-001 Spring 2015 > Modules
View Progress
+ Module
resources
Clicker Registration Tool


Connection Info
Channels 40, 26
Session ID 427738

ARE YOU HERE?

1) yes
2) no

last time…

generics

-we always see `<>` associated with `ArrayList`…

```
ArrayList<Shape> list = new ArrayList<Shape>();
```

-`ArrayList` is a **generic class** — we can create any version of it that we want

-**generic programming:** algorithms are written in terms of types to-be-specified-later

- -algorithms instantiated when needed for specific types defined by parameters

-here’s what the code actually looks like:

```
public class ArrayList<T> {
  T storage[];
  int capacity, numItems;

  public void add(T item)
  { … }
}
```

-the placeholder `T` is replaced with the real type when you instantiate an `ArrayList` with `<>`

-`T` can be used as a type anywhere in `ArrayList` class

-generics allow for type-checking at compile time instead of run-time

-can detect type mismatch **BEFORE** your code runs

BEFORE GENERICS:

```
ArrayList l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // crash
```

ALTERNATIVE:

```
ArrayList<String> l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // compile error
```

COMPILE-TIME ERRORS ARE ALWAYS BETTER THAN RUN-TIME!

-`static` methods can have their own generic types

-declare the generic type before the return type:

```
public static <T> boolean doWork(…){…}
```

-we can refer to `T` as a type within that method only!

-example:

```
public static <T> boolean contains(T[] array, T item)
{
  for(int i=0; i < array.length; i++)
    if(array[i].equals(item))
      return true;

  return false;
}
```

today…

-algorithm analysis

-complexity growth rate

-big-O notation

algorithm analysis

-correctness is only half the battle

-programs are expected to terminate in a reasonable amount of time

-running time of a program is strongly correlated to the choice of algorithms used in problem solving

-**how much time and space does an algorithm require?**

example…

finding a word in a dictionary

ALGORITHM 1:

1) start on the **first** page, **first** entry
2) if word not found, move to the next entry
3) if very end of dictionary reached, word not found

IS THIS ALGORITHM CORRECT?

1) yes
2) no

CAN WE DO BETTER?

1) yes
2) no

finding a word in a dictionary

ALGORITHM 2:

1) guess which page the entry is on

2) did we go too far?
    - go back some pages

3) did we not go far enough?
    - go forward some pages

4) continue narrowing

WHAT DOES THIS ALGORITHM ASSUME ABOUT THE DICTIONARY?

-ALGORITHM 1: LINEAR SEARCH

-running time directly related to size of dictionary

-assume 180K words, and 0.25s to check one word

-12 hours to complete!

-ALGORITHM 2: BINARY SEARCH

-more like what humans do

-4 seconds to complete!

TIME * 2 ⟶ **O(N)**

TIME + 0.25 ⟶ **O(logN)**

-what if the dictionary doubles?

ALGORITHM 1 RUN-TIME?

1) time * 0.5
2) time * 2
3) time + 0.25
4) time + 10

ALGORITHM 2 RUN-TIME?

1) time * 0.5
2) time * 2
3) time + 0.25
4) time + 10

a note on logarithms

-a **logarithm** is an exponent indicating the power to which a base is raised to produce a given number

$\log_B N = X$ $B^X = N$

HOW MANY BITS DOES IT TAKE TO REPRESENT A NUMBER?

-by default the base is 2 …we'll come back to this

-the logarithm grows slowly

$9 < \log 1000 < 10$
$19 < \log 1{,}000{,}000 < 20$
$29 < \log 1{,}000{,}000{,}000 < 30$

-**$N \log N$** is closer to **N** than **N^2**

why is binary search O(log N)?

why is the default base 2?

finding a word in a dictionary

-binary search will **always** win for large dictionaries
  -as **N** increases, the gap between the algorithms becomes larger

-linear search has linear growth rate
  -graph is a _______ line
  -run time for **N = T** units of time
  -run time for **2N = 2T** units of time

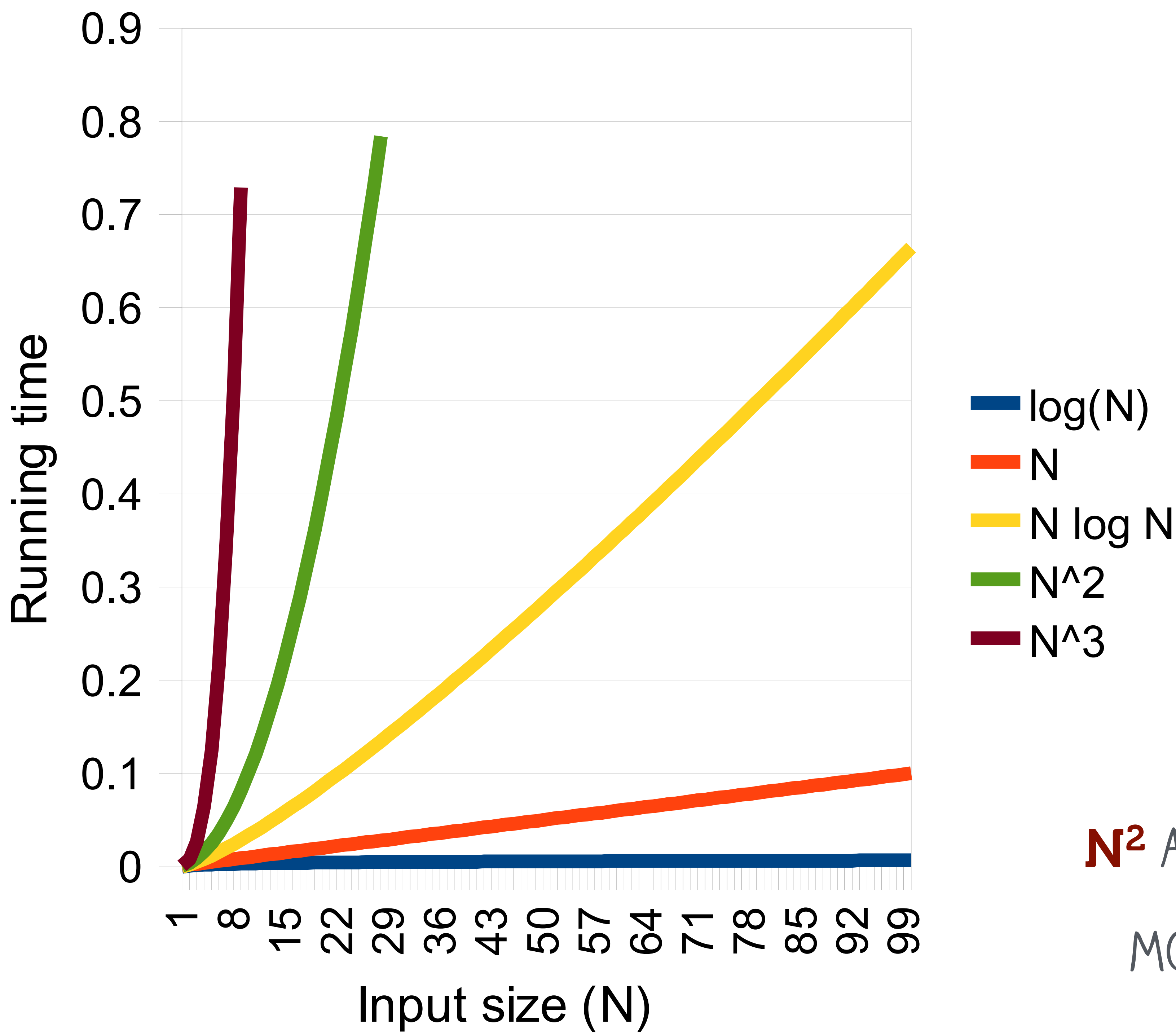
1) exponential
2) straight
3) negative-slope

-binary search has logarithmic growth rate
  -run time for **N = T** units of time
  -run time for **2N = T+_** units of time

1) 1
2) 2
3) 10

growth rate

typical run-time complexities



N^2 AND N^3 ARE TYPICALLY NOT ACCEPTABLE FOR MODERATE INPUT SIZES!

-knowing that **$F(N) < G(N)$** for a particular **N** is not very useful

-instead, we measure the functions' growth rates

-for sufficiently large **N**, a function's growth rate is determined by its dominate term

$10N^2 + 40N + 760$ → WHAT IS THE DOMINATE TERM?

c	constant
$\log N$	logarithmic
N	linear
$N \log N$	linearithmic
N^2	quadratic
N^3	cubic

increasing growth rate ↓

how to get log growth?

-how many bits are needed to represent **N** consecutive integers?

-starting at **x=1**, how many iterations of **x*2** before **x>=N**?

-the *repeated doubling* principle

-starting at **x=N**, how many iterations of **x/2** before **x<=1**?

-the *repeated halving* principle

big-O notation

-**big-O notation** (**O**) is used to capture the dominate term in an algorithm

- -assuming large **N**!

-for example, the running time of a quadratic algorithm is **N^2** is specified **$O(N^2)$**

- -pronounced “order N squared”

-this notation allows us to establish a relative order among algorithms

- -**$O(N \log N)$** is better than **$O(N^2)$**

what's code got to do, got to do with it…

-**$O(N^2)$** and **$O(N^3)$** are impractical for most **N**

-clever programming tricks **CANNOT** make an inefficient algorithm fast

- -a poorly coded linear algorithm trumps a quadratic algorithm in a highly efficient machine language

TAKE AWAY:

OPTIMIZING THE ALGORITHM (OR CHOOSING THE BEST ONE) WILL GET YOU MUCH FURTHER THAN OPTIMIZING THE CODE

worst, average, best

- **worst-case** is a guarantee on all inputs — it will never be worse than this

- **average-case** is the common case, measured over all possible inputs
  - this is the most useful!

- **best-case** is the absolute fastest that an algorithm can terminate
  - we don't care about this because it rarely happens

example...

finding the maximum item in an array

ALGORITHM?

1) initialize `max` to the first element

2) scan through each item in the array
   - if the item is greater than `max`, update `max`

WHAT IS THE BIG-O COMPLEXITY OF THIS ALGORITHM?

1) **c**
2) **log N**
3) **N**
4) **N log N**
5) **N^2**
6) **N^3**

finding the smallest difference

ALGORITHM?

```
diff = MAX_INTEGER;
for(int i=0; i<array.length-1; i++)
{
  num1 = array[i];
  for(int j=i+1; j<array.length; j++)
  {
    num2 = array[j];
    if (abs(num1-num2) < diff)
      diff = abs(num1-num2);
  }
}
return diff;
```

1) **c**
2) **log N**
3) **N**
4) **N log N**
5) **N^2**
6) **N^3**

WHAT IS THE BIG-O COMPLEXITY OF THIS ALGORITHM?

next time…

-reading

  -chapters 5 & 6

-homework

  -assignment 2 due Friday at 11:59pm

    -must complete with a partner!