CS 146, Data Structures and Algorithms, SJSU: Doug Case

Program 3:
Big Theta, n squared vs n cubed. Where is n0?

Following is a brief reminder from the Cormen book about big Theta and n0.

Note for this program you only have to change one line of code for the first part! (And then just temporarily comment out some code for the rest).

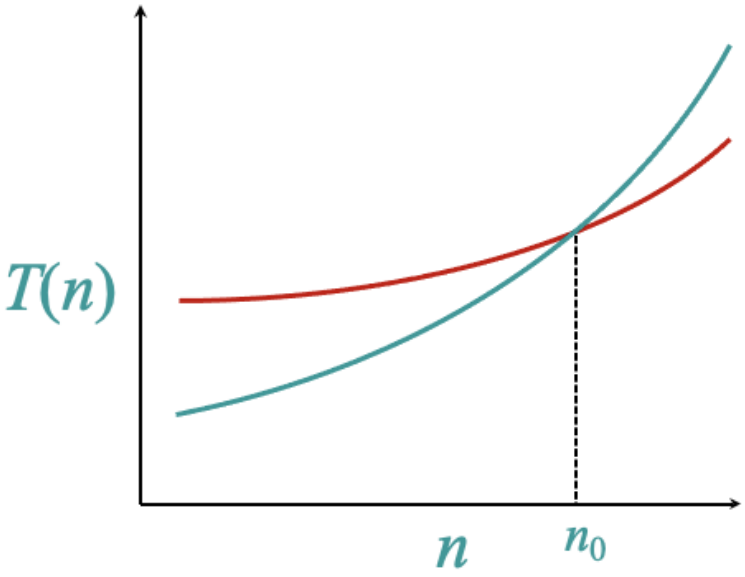
Θ-notation

Math:

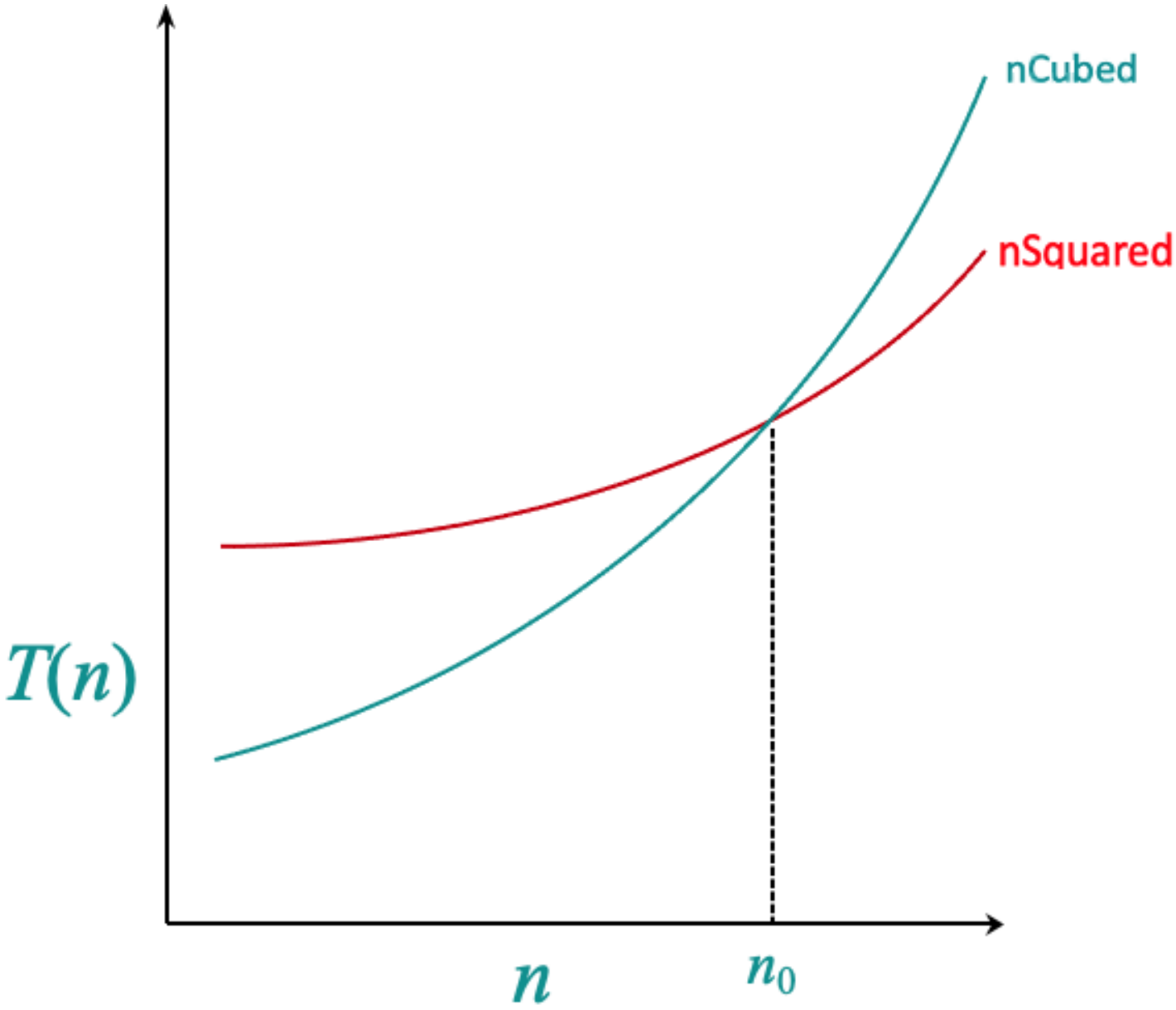
$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \}$$

Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.



1) Copy the given code below into an IDE:

```
package package1;  // NOTE: Students might not need this, or might need to change this
import java.util.Random;

// Java program for analyzing performance
public class BigO
{
   /* This does a bit of random useless processing num ** 3 times, and returns how long it takes in ms */
   static long nCubed(int num)
```

```
  {
    long startTime = System.currentTimeMillis();
    long temp = 0;
    Random rand = new Random();
    // Setting upper bound to generate random numbers in specific range
    int upperbound = 1000;
    for (long i = 0; i < num; i++)
    {
      for (long j = 0; j < num; j++)
      {
        for (long k = 0; k < num; k++)
        {
          temp = i * j * k + rand.nextInt(upperbound);
        }
      }
    }
    return System.currentTimeMillis() - startTime;
  }

  /* This does a lot of random useless processing num ** 2 times, and returns how long it takes in ms */
  static long nSquared(int num)
  {
    long startTime = System.currentTimeMillis();
    Random rand = new Random();
    // Setting upper bound to generate random numbers in specific range
    int upperbound = 1000;
    String s = "I am a big fat hungry walrus who lives in Greenland and eats a lot of prawns on Tuesdays.";
    for (int i = 0; i < num; i++)
    {
      for (int j = 0; j < num; j++)
      {
        // NOTE: This starts the top half of the inner loop
        String sub =
s.substring(12).toLowerCase().toUpperCase().toLowerCase().toUpperCase().toLowerCase().toUpperCase();
        sub =
s.substring(12).toLowerCase().toUpperCase().toLowerCase().toUpperCase().toLowerCase().toUpperCase();
        sub =
s.substring(12).toLowerCase().toUpperCase().toLowerCase().toUpperCase().toLowerCase().toUpperCase();
        char c = sub.charAt(5);
        String sChar = Character.toString(c);
        sChar = null;

        // NOTE: this starts the bottom half of the inner loop
        // TODO for the next question, comment out the top, then the bottom, to see which is slower
        int myRandomInt = rand.nextInt(upperbound);
        myRandomInt = 0;
        int yourRandomInt = rand.nextInt(upperbound);
        yourRandomInt = 0;
        int herRandomInt = rand.nextInt(upperbound);
        herRandomInt = 0;
        int hisRandomInt = rand.nextInt(upperbound);
```

```
            hisRandomInt = 0;
            int theirRandomInt = rand.nextInt(upperbound);
            theirRandomInt = 0;
            int ourRandomInt = rand.nextInt(upperbound);
            ourRandomInt = 0;
            int studentRandomInt = rand.nextInt(upperbound);
            studentRandomInt = 0;
          }
        }
        return System.currentTimeMillis() - startTime;
    }

    // main driver program
    public static void main(String args[])
    {
        // TODO first run the code asis to see the performance of the nSquared and nCubed methods.
        // TODO adjust the next line to see where n0 is.
        // TODO that is, in the "for" statement, play with any/all of the 3 numbers as needed to find
        // TODO the approximate value of n0 where
        for (int n = 10; n <= 1000; n *= 10) {   // TODO change this line of code!
            long nSquaredTime = nSquared(n);
            System.out.println("\nFor " + Integer.toString(n) + " items, n**2 time in ms is: " + Long.toString(nSquaredTime));
            long nCubedTime = nCubed(n);
            System.out.println("For " + Integer.toString(n) + " items, n**3 time in ms is: " + Long.toString(nCubedTime));
        }
    }
}
```

2) Adjust any or all 3 numbers in the for statement in main as needed to find n0, that is the approximate value of n where the time taken by nSquared and nCubed are about the same. Also feel free to change from *= to += or change that line of code in any way. This will take some trial and error.
3) Figure out which is the slow part of the nSquared method. You can do that by trial and error, commenting out the top half, and then the bottom half of the nSquared method, and running it.
4) Submit as text four things:
    a. The one line of code you changed in main, now it is "for (int n = 10; n <= 1000; n *= 10)"
    b. The output of your program
    c. The approximate value of n0 on your computer. That is, the approximate value of n where the time taken for nSquared and nCubed are about the same. (The numbers will vary slightly with different runs, so you don't have to be very precise.
    d. Describe which is slower, the top half or the bottom half of the nSquared method. Is it a little slower or a lot slower? Is that a surprise to you?