Fundamental Algorithms, Assignment 4
Solutions

1. Consider the recursion \( T(n) = 9T(n/3) + n^2 \) with initial value \( T(1) = 1 \). Calculate the precise values of \( T(3), T(9), T(27), T(81), T(243) \). Make a good (and correct) guess as to the general formula for \( T(3^i) \) and write this as \( T(n) \). (Don’t worry about when \( n \) is not a power of three.) Now use the Master Theorem to give, in Thetaland, the asymptotics of \( T(n) \). Check that the two answers are consistent.

Solution: \( T(3) = 9(1) + 3^3 = 18 = 2(9), \)
\( T(9) = 9(18) + 9^2 = 243 = 3(81), \)
\( T(27) = 9(243) + 729 = 2916 = 4(729), \)
\( T(81) = 32805 = 5(6561), \)
\( T(243) = 354294 = 6(59049). \) In general,
\( T(3^i) = (i + 1)3^i. \)

With \( n = 3^i \) we have \( 3^2i = n^2 \) and \( i = \log_3 n \) so the formula is \( T(n) = n^2(1 + \log_3 n) \). In Thetaland, \( T(n) = \Theta(n^2 \lg n) \). With the Master Theorem, as \( \log_3 9 = 2 \) we are in the special case which gives indeed \( T(n) = \Theta(n^2 \lg n) \).

Another approach is via the auxilliary function \( S(n) \) discussed in class. Here \( S(n) = T(n)/n^2 \). Dividing the original recursion by \( n^2 \) gives
\[
\frac{T(n)}{n^2} = \frac{T(n/3)}{(n/3)^2} + 1
\]
so that
\[
S(n) = S(n/3) + 1 \text{ with initial value } S(1) = T(1)/1^2 = 1
\]
so that
\[
S(n)1 + \log_3 n \text{ and so } T(n) = n^2(1 + \log_3 n)
\]

2. Use the Master Theorem to give, in Thetaland, the asymptotics of these recursions:

(a) \( T(n) = 6T(n/2) + n\sqrt{n} \)
Solution: As \( \log_2 6 = \frac{\log_2 6}{\log_2 2} = 2.58 \ldots > 3/2 \) we have Low Overhead and \( T(n) = \Theta(n^{\log_2 6}). \)

(b) \( T(n) = 4T(n/2) + n^5 \)
Solution: \( \log_2 4 = 2 < 5 \) so we have High Overhead and \( T(n) = \Theta(n^5). \)

(c) \( T(n) = 4T(n/2) + 7n^2 + 2n + 1 \)
Solution: \( \log_2 4 = 2 \) and the Overhead is \( \Theta(n^2) \) so \( T(n) = \Theta(n^2 \lg n). \)
3. Toom-3 is an algorithm similar to the Karatsuba algorithm discussed in class. (Don’t worry how Toom-3 really works, we just want an analysis given the information below.) It multiplies two \( n \) digit numbers by making five recursive calls to multiplication of two \( n/3 \) digit numbers plus thirty additions and subtractions. Each of the additions and subtractions take time \( O(n) \). Give the recursion for the time \( T(n) \) for Toom-3 and use the Master Theorem to find the asymptotics of \( T(n) \). Compare with the time \( \Theta(n \log_3 5) \) of Karatsuba. Which is faster when \( n \) is large?

Solution: \( T(n) = 5T(n/3) + O(n) \) as the thirty is absorbed into the big oh \( n \) term. From the master theorem \( T(n) = \Theta(n \log_3 5) \). As \( \log_3 5 = \frac{\ln 5}{\ln 3} = 1.46\cdots < 1.58\cdots = \log_2 3 \)

it is better that the \( \Theta(n \log_2 3) \) of Karatsuba. (In practice unless \( n \) is really large Karatsuba does better because Toom-3 has large constant factors.)

4. Write the following sums in the form \( \Theta(g(n)) \) with \( g(n) \) one of the standard functions. In each case give reasonable (they needn’t be optimal) positive \( c_1, c_2 \) so that the sum is between \( c_1g(n) \) and \( c_2g(n) \) for \( n \) large.

(a) \( n^2 + (n + 1)^2 + \ldots + (2n)^2 \)
   \[ \text{Solution:} \Theta(n^3) \]
   There are \( \sim n \) terms all between \( n^2 \) and \( 4n^2 \) so the sum is between \( n^3(1 + o(1)) \) and \( 4n^3(1 + o(1)) \).

(b) \( \lg^2(1) + \lg^2(2) + \ldots + \lg^2(n) \)
   \[ \text{Solution:} \Theta(n \lg^2 n) \]
   There are \( n \) terms all at most \( \lg^2(n) \) so an upper bound is \( n \lg^2(n) \). Lopping off the bottom half of the terms we still have \( n/2 \) terms and each is at least \( \lg^2(n/2) = (\lg(n) - 1)^2 \sim \lg^2 n \) so the lower bound is \( (1 + o(1))(\frac{n}{2})\lg^2 n \).

(c) \( 1^3 + \ldots + n^3 \)
   \[ \text{Solution:} T(n) = \Theta(n^4) \]
   Upper bound \( n^4 \) as \( n \) terms, each at most \( n \). Lopping off bottom half yields \( n/2 \) terms, each at least \( (n/2)^3 \) giving a lower bound \( (n/2)(n/2)^3 = n^4/16 \).

5. Give an algorithm for subtracting two \( n \)-digit decimal numbers. The numbers will be inputted as \( A[0\ldots N] \) and \( B[0\ldots N] \) and the output should be \( C[0\ldots N] \). How long does your algorithm take, expressing your answer in one of the standard \( \Theta(g(n)) \) forms.
**Solution:** Here is one way, the term BORROW being the truth value of whether you have “borrowed.”

BORROW=false;
FOR I=0 TO N;
  IF BORROW=false THEN X=A[I]-B[I];
  IF BORROW=true THEN X=A[I]-1-B[I];
  IF X ≥ 0 THEN
    C[I]=X;
    BORROW=false;
  ELSE
    C[I]=X+10;
    BORROW=true;
ENDFOR
IF BORROW=true THEN ERROR;
END

This takes only a single pass and so is a linear time, that is $\Theta(N)$ algorithm.