FA, Homework 4

Quentin McGaw (qm301)
02/16/17

The computer is useless. It can only answer questions.
– Pablo Picasso

When asked for the asymptotics answer in a form $\Theta(n^a)$ or $\Theta(\lg^b n)$ or $\Theta(n^a \lg^b n)$ for some reals a, b.

1. **Consider the recursion $T(n) = 9T(n/3) + n^2$ with initial value $T(1) = 1$. Calculate the *precise* values of $T(3), T(9), T(27), T(81), T(243)$.**

   $T(3) = 9T(1) + 3^2 = 9(1) + 3^2 = 18 = 2(3^2)$
   $T(9) = 9T(3) + 9^2 = 9(9(1) + 3^2) + 9^2 = 243 = 3(9^2)$
   $T(27) = 9T(9) + 27^2 = 2916 = 4(27^2)$
   $T(81) = 9T(27) + 81^2 = 32805 = 5(81^2)$
   $T(243) = 9T(81) + 243^2 = 354294 = 6(243^2)$
   We used the following Python code to calculate these values:

```python
def T(n):
    """ Recursive function
        T(n)=9T(n/3)+n^2 with T(1) = 1
        Only valid for n a power of 3
    """
    if n == 1:
        return 1
    elif n % 3 != 0:
        raise Exception("n should be divisible by 3")
    else:
        return 9*T(n/3) + n*n

if __name__ == "__main__":
    print T(3)
    print T(9)
    print T(27)
    print T(81)
    print 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.)

   From our calculations, it seems that $T(3^i) = (i + 1)3^2 i$
   We need to express i and $3^2 i$ as a function of n
   First, $n = 3^i \Rightarrow i = lg_3\ n$
   Also, $n = 3^i \Rightarrow n^2 = 3^{2i}$ Hence $T(n) = (1 + lg_3\ n)n^2$

   Now use the Master Theorem to give, in Thetaland, the asymptotics of $T(n)$. Check that the two answers are consistent.

In Thetaland, $T(n) = \Theta(n^2 lg\ n)$. With the Master Theorem, we are in the special case because $lg_3\ 9 = 2$ so $T(n) = \Theta(n^2 lg\ n)$

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

   (a) $T(n) = 6T(n/2) + n\sqrt{n}$

   We can rewrite $T(n)$ as $T(n) = 6T(\frac{n}{2}) + n^{1.5}$, hence we have the following parameters: $c = 1.5$, $b = 2$ and $a = 6$.
   Because $log_b\ a = log_2\ 6 = \frac{log\ 6}{log\ 2} = 2.58 > 1.5$, we are in the low overhead case and thus $T(n) = \Theta(n^{log_b\ a}) = \Theta(n^{log_2\ 6})$

   (b) $T(n) = 4T(n/2) + n^5$

   We have the following parameters: $c = 5$, $b = 2$ and $a = 4$.
   Because $log_2\ 4 < 5$, we are in the high overhead case and thus $T(n) = \Theta(f(n)) = \Theta(n^5)$

   (c) $T(n) = 4T(n/2) + 7n^2 + 2n + 1$

   We have the following parameters: $c = 2$, $b = 2$ and $a = 4$.
   Because $log_2\ 4 = 2$ and the overhead is $\Theta(n^2)$, we have $T(n) = \Theta(n^2 log^{(0+1)}\ n)$

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 $T(n)$.

   $T(n) = 5T(\frac{n}{3}) + O(30n) = 5T(\frac{n}{3}) + O(n)$
   From the Master theorem, we have the following parameters: $a = 5$, $b = 3$ and $c = 1$.
   Because $log_b\ a = log_3\ 5 = \frac{log\ 5}{log\ 3} = 1.465 > 1$, we are in the low overhead case and thus $T(n) = \Theta(n^{log_b\ a}) = \Theta(n^{log_3\ 5})$

   Compare with the time $\Theta(n^{\log_2 3})$ of Karatsuba. Which is faster when n is large?

   Because $log_3\ 5 = 1.465 < log_2\ 3 = 1.585$, the $\Theta(n^{log_3\ 5})$ of Toom-3 is faster than the $\Theta(n^{log_2\ 3})$ of Karatsuba, so Toom-3 is faster when n is large.

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 $c_1 g(n)$ and $c_2 g(n)$ for n large.**

   (a) $n^2 + (n+1)^2 + \ldots + (2n)^2$

   $\Theta(g(n)) = \Theta(n^3)$ and we have $c1 = 0.5$ and $c2 = 1.5$

(b) $\lg^2(1) + \lg^2(2) + \ldots + \lg^2(n)$

$\Theta(g(n)) = \Theta(nlg^2\ n)$ and we have $c1 = 0.5$ and $c2 = 1.5$

(c) $1^3 + \ldots + n^3$.

$\Theta(g(n)) = \Theta(n^4)$ and we have $c1 = 0.5$ and $c2 = 1.5$

5. **Give an algorithm for subtracting two n-digit decimal numbers. The numbers will be inputted as $A[0 \cdots N]$ and $B[0 \cdots N]$ and the output should be $C[0 \cdots N]$. (Assume that the result will be nonnegative.)**

Assuming the digit at position 0 is the most significant one, the following algorithm works:

```
for i from 0 to N do
    X = A[i] - B[i] if X ¿= 0 then
    |  C[i] = X
    else
    |  C[i-1]– C[i] = 10 + X
    end
end
```

Algorithm 1: n-digit decimal subtraction algorithm

Note that we used the following Python code to test it out:

```
def subtract_n_digit_numbers(A, B):
    N = len(A)
    if N != len(B):
        raise Exception("A and B must have the same length")
    C = [None for _ in range(N)]
    for i in range(0, N):
        x = A[i] - B[i]
        if x >= 0:
            C[i] = x
        else:
            C[i-1] -= 1 #non-negative result so that works
            C[i] = 10 + x
    return C

if __name__ == "__main__":
    A = [2,9,4,3,2]
    B = [1,5,3,7,1]
    C = subtract_n_digit_numbers(A, B)
    print C
```

How long does your algorithm take, expressing your answer in one of the standard $\Theta(g(n))$ forms.

Assuming the addition and subtraction operations of two digits take $O(1)$, this algorithm takes $\Theta(g(n)) = \Theta(n)$ as it is a simple for loop of size N.

The mind is not a vessel to be filled but a fire to be kindled.
– Plutarch