CIS 121
Solution Sketches to Practice Problems for Exam 1
February 16, 2016

---

1. Prove using induction that n is $O(2^n)$.

Solution. We will prove that there exists $c > 0$ and n_0 such that $n < c2^n$ for all $n >= n_0$. We pick $c = 1$ and $n_0 = 1$.

Induction Hypothesis: Assume that the claim is true when $n = k$, for some $k \geq 1$. In other words, we assume that $k \leq 2^k$, for some $k \geq 1$.
Base Case: $1 \leq 1 \cdot 2^1$
Induction Step: We want to prove the claim when $n = k + 1$.

$$k + 1 \leq 2^k + 1 \leq 2^k + 2^k = 2^{k+1}.$$

2. Prove that $2^{(n^2)}$ is not $O(5^n)$. Do *not* use any theorems about Big-Oh that you might happen to know other than the definitions.

Solution. We need to prove that for all real constants $c > 0$ and positive integers N, there exists an $n > N$ such that $2^{n^2} > c \cdot 5^n$. By taking logarithms of both parts, we have that

$$n^2 > \log c + n \log 5 \Rightarrow n^2 - n \log 5 - \log c > 0 \tag{1}$$

First suppose that $c \leq 1$, which means that $\log c \leq 0 \Rightarrow -\log c \geq 0$. Rewrite (1) as

$$n\,(n - \log 5) + (-\log c) > 0$$

and notice that this trivially holds for all $n > \log 5$.

Now suppose that $c > 1$, so that $\log c > 0$. Rewrite (1) as

$$n\,(n - \log 5) > \log c$$

and notice that this trivially holds for all $n > \log 5 + \log c$.

As a result, pick $n = \max\{\lceil \log 5 \rceil + 1, \lceil \log 5 + \log c \rceil + 1, N + 1\}$ and we are done.

3. Solve the following recurrence. Give a tight bound, i.e., express your answer using the Θ notation. Assume that $T(n) = 1$, when $n = 1$.

$$T(n) = T(n-1) + 1/n$$

Solution.

$$\begin{aligned}
T(n) &= T(n-1) + 1/n \\
&= T(n-2) + 1/(n-1) + 1/n \\
&= T(n-3) + 1/(n-2) + 1/(n-1) + 1/n \\
&\ldots\ldots \\
&\ldots\ldots \\
&= T(n-k) + 1/(n-(k-1)) + 1/(n-(k-2)) + \cdots + 1/(n-1) + 1/n
\end{aligned}$$

When $k = n-1$, we get

$$T(n) = 1 + \sum_{i=2}^{n} 1/i = \sum_{i=1}^{n} 1/i = \Theta(\log n)$$

4. Consider the following code fragment

```
for(int i=1;i<=n;i=2*i)
  for (int k = i; k >0; k = k/2)
    print('*');
```

a. Compute the number of stars printed as a function of n. You can assume that $n = 2^m$.
b. Give a Θ-bound.

Solution. Outer loop gets executed m times where $n = 2^m$. Inner loop gets executed $\log i + 1$ times for each iteration i of the outer loop, so in total we have

$$\begin{aligned}
&(\log(1)+1) + (\log(2)+1) + (\log(4)+1) + (\log(8)+1) + \cdots + (\log(2^m)+1) \\
&= 1 + \frac{m(m+1)}{2} + m + 1 \\
&= \frac{(m+1)(m+2)}{2}
\end{aligned}$$

hence it is $\Theta((\log n)^2)$.

5. Let X and Y be two n-digit numbers. Assume that the word size on your computer can store one base-10 digit. We want to compute the product XY. Assume that n is a power of 2.

a. What is the running time of the multiplication algorithm that you learned in school?

b. Let X_h, Y_h be the higher order digits of X and Y, respectively. Let X_l, Y_l be the lower order digits of X and Y, respectibely. Then the product XY can be written as

$$XY = (X_h \cdot 10^{n/2} + X_l)(Y_h \cdot 10^{n/2} + Y_l) = X_h Y_h (10^n) + (X_h Y_l + X_l Y_h) 10^{n/2} + X_l Y_l$$

Use this to devise a divide and conquer algorithm for the problem.

c. Write a recurrence for your algorithm in part (b). What is the running time of your algorithm?

d. Consider the product $(X_h + X_l)(Y_h + Y_l)$. Expand the product and after noticing the terms of this expansion, devise a divide and conquer algorithm that uses only 3 multiplications.

e. Write the recurrence for the algorithm in part (d) and derive its running time.

Solution. (a) $\Theta(n^2)$.
(b) Recursively, solve $X_hY_h, X_lY_l, X_hY_l, X_lY_h$. Then compute the expression given in question.
(c) $T(n) = 4T(n/2) + \Theta(n)$, when $n \geq 2$. $T(1) = 1$. This recurrence yields a running time of $\Theta(n^2)$.
(d) Recursively compute the following multiplications: $p = (X_h + X_l)(Y_h + Y_l), X_hY_h$, and X_lY_l. Our answer is given by $X_hY_h(10^n) + (p - X_hY_h - X_lY_l)(10^{n/2}) + X_lY_l$
(e) $T(n) = 3T(n/2) + \Theta(n)$, when $n \geq 2$, $T(1) = 1$. This recurrence yields $T(n) = n^{\log_2 3}$, using case 1 of the Simplified Master Theorem.

6. Modify the quicksort algorithm so that its running time is $O(n \log n)$ in the worst case. You may assume that all elements are distinct.

Solution. We modify the algorithm that was done in class. Instead of picking an arbitrary pivot, we now use the selection algorithm done in class to obtain the median of the input array (first two lines in `Partition`). These changes take an additional $O(n)$ time (note that we could have found the pivotIndex more efficiently, but that would not affect the asymtotic running time).

```
QSort(A[lo..hi])
  if hi <= lo then
    return
  else
    mid = Partition(A, lo, hi)
    QSort(A[lo..mid-1])
    QSort(A[mid+1..hi])

Partition(A, lo, hi, pIndex)
  pivot = Select(A, floor(lo+hi/2))
  pivotIndex = location of pivot in A
  swap(A, pivotIndex, hi)
  left = lo
  right = hi-1
  while left <= right do
     if (A[left] <= pivot) then
        left = left + 1
     else
```

```
            swap(A, left, right)
            right = right - 1
    swap(A, left, hi)
    return left
```

Since we partition around the median of the input array, we have the following recurrence (remember to include the base case, otherwise, you will lose points).

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & n \geq 2 \end{cases}$$

This recurrence results in $T(n) = \Theta(n \log n)$, as done in class.

7. The Hotel Partner Problem is as follows. There are $2n$ people and n hotel rooms. Each person maintains a preference list of the remaining $2n-1$ people. The objective is to assign two people to each room such that the assignment is stable. An assignment is stable if there are no two people assigned to different rooms who prefer each other over their current partners. Give an instance of the Hotel Partner Problem for which there is no stable solution.

Solution. Consider the following instance with four people, p_1, p_2, p_3, p_4. Their preference lists are given below.

$p_1 : (p_4, p_2, p_3)$
$p_2 : (p_1, p_4, p_3)$
$p_3 : (p_1, p_2, p_4)$
$p_4 : (p_2, p_1, p_3)$

It is easy to verify that there is no stable solution for the above instance.

8. Suppose that you have a "black box" worst-case linear-time median subroutine. Give a simple, linear-time algorithm that solves the selection problem for any arbitrary order statistic, i.e., given an unsorted array A containing n elements and an integer i, in $O(n)$ time, your algorithm should return the element in A, which is the i^{th} smallest element in A. Your algorithm must use the black-box median-finding subroutine. You may assume that i lies within the bounds of the input array A and that n is a power of 2.

Solution. Let the black-box median finding function be called `Median`. The following algorithm finds the element of rank i.

```
ModifiedSelect(A[lo,hi], i)
if lo==hi then
  reuturn A[lo]
m = Median(A[lo,hi]) // m is the median element
mIndex = floor(lo+hi/2)
Partition(A[lo,hi], mIndex)  // use Partition from QuickSort
```

```
if (i == midIndex) then
  return m
else if (i < mIndex) then
  return ModifiedSelect(A[lo,mIndex-1], i)
else
  return ModifiedSelect(A[mIndex+1,hi], i-mIndex)
```

The running time of the algorithm is given by the recurrence $T(n) = T(n/2) + n$, with the base case being $T(1) = 1$. This recurrence yields a running time of $T(n) = O(n)$, using case 3 of the simplified master theorem.