22C:44 Midterm Solution

---

1. (a) Prove that $n = o(n \lg n)$.
   Solution: We need to show that for every $c > 0$, there exists an $n_0 > 0$ such that $0 \leq n < cn \lg n$ for all $n \geq n_0$. This is equivalent to showing that for every $c > 0$, there exists $n_0 > 0$ such that $0 \leq 1/c < \lg n$. For each $c > 0$, pick $n_0 > 2^{1/c}$. Then, for all $n \geq n_0$, we have that $n > 2^{1/c}$, which is equivalent to saying that $\lg n > 1/c$.
   A simple alternate solution is obtained by using limits.

   (b) Let a and b be strictly positive constants. Express the sum

$$S_n = \sum_{i=0}^{\log_b n} \left(\frac{a}{b}\right)^i$$

   in Θ-notation as simply as possible. In other words, find an $f(n)$ that is as simple as possible such that $S_n = \Theta(f(n))$. Using your answer, point out the error in my solution to Problem 4(a) in Homework 2.

   Solution:
   Case 1: $a = b$ $S_n = \sum_{i=0}^{\log_b n} 1 = \Theta(\log n)$.
   Case 2: $a < b$

$$\left(\frac{a}{b}\right)^0 < S_n < \sum_{i=0}^{\infty} \left(\frac{a}{b}\right)^i.$$

$$1 < S_n < \frac{1}{1 - \frac{a}{b}} = \frac{b}{b-a}.$$

   Therefore, $S_n = \Theta(1)$.
   Case 3: $a > b$

$$\begin{aligned} S_n &= \sum_{i=0}^{\log_b n} \left(\frac{a}{b}\right)^i \\ &= \frac{\left(\frac{a}{b}\right)^{\log_b n + 1} - 1}{\left(\frac{a}{b}\right) - 1} \\ &= \Theta\left(\left(\frac{a}{b}\right)^{\log_b n}\right) \\ &= \Theta\left(\frac{n^{\log_b a}}{n}\right) \\ &= \Theta\left(n^{\log_b a - 1}\right). \end{aligned}$$

   In the homework solution the cases in which $a < b$ and $a > b$ were not distinguished. However, they ought to be distinguished since the sums in these two cases are asymptotically different and as a result the solution to the recurrence in Problem 4(a) is different in these two cases. When $a < b$, the solution is $T(n) = \Theta(n)$ and when $a > b$, the solution is $T(n) = \Theta(n^{\log_b a})$.

2. (a) Solve the following recurrence for $T(n)$ using your favorite method.

$$T(n) = 2T\left(\frac{n}{4}\right) + n\sqrt{n}.$$

Assume that the recurrence holds for $n > 1$ and $T(n) = \Theta(1)$ for $n \leq 1$.
Solution: Use the Master Theorem. So we have $a = 2$, $b = 4$, and therefore $n^{\log_b a} = n^{1/2}$. Also $f(n) = n\sqrt{n} = n^{3/2}$. We now check if Case (iii) applies. For any ϵ, $0 < \epsilon \leq 1$, $n^{3/2} = \Omega(n^{1/2+\epsilon})$. We now test the regularity condition.

$$af\left(\frac{n}{b}\right) = 2f\left(\frac{n}{4}\right) = 2 \cdot \frac{n}{4} \cdot \sqrt{\frac{n}{4}} = \frac{n\sqrt{n}}{4} = \frac{f(n)}{4}.$$

This verifies the regularity condition; Case (iii) applies and hence $T(n) = \Theta(n\sqrt{n})$.

(b) Solve the recurrence

$$T(n) = \frac{1}{n}\sum_{k=1}^{n-1} T(k) + n$$

using the substitution method. Assume that the recurrence holds for all $n > 1$ and $T(1) = 1$. Use $T(n) \leq cn^2$ for all $n \geq n_0$ as your guess. Identify specific values for c and n_0 for which the guess holds.

Solution: Choose $n_0 = 1$.
Base Case: We need to show that $T(1) \leq c \cdot 1^2$. Choosing $c \geq T(1) = 1$ ensures this.
Inductive Hypothesis: For all k, $1 \leq k < n$, $T(k) \leq ck^2$.
Inductive Step: Substituting the inductive hypothesis in the given recurrence relation we get

$$\begin{aligned}
T(n) &\leq \frac{3}{n}\sum_{k=1}^{n-1} ck^2 + n \\
&= \frac{3c}{n}\left(\frac{n(n-1)(2n-3)}{6}\right) + n \\
&= \frac{c}{2}(n-1)(2n-3) + n \\
&= cn^2 - \frac{c5n}{2} + \frac{3c}{2} + n
\end{aligned}$$

For this quantity to be at most cn^2, we need that

$$\frac{-5cn}{2} + \frac{3c}{2} + n \leq 0.$$

This is true for all $c \geq 1$ and $n \geq 1$. So choosing $c = 1$ makes both the inductive case and the base case go through.

3. (a) Here is an algorithm that computes y^z, given y and z.

```
Power(y, z) {
    if (z == 0) then
        return 1;
    else if (IsOdd(z)) then
        return Power(y^2, ⌊z/2⌋) * y;
    else if (IsEven(z)) then
        return Power(y^2, ⌊z/2⌋);
}
```

Here `IsOdd(z)` returns `True` if `z` is odd; `False` otherwise. Similarly, `IsEven(z)` returns `True` if `z` is even; `False` otherwise.

Analyze the above algorithm to determine the amount of time it takes to compute 5^n. Do this by setting up a recurrence and solving the recurrence.

Solution: Let $T(n)$ be the time the algorithm takes to compute 5^n. The following recurrence is immediate on examining the pseudocode of the algorithm

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1).$$

The recurrence holds for $n \geq 1$ and $T(0) = \Theta(1)$. This is identical to the "binary search recurrence" and solves to $T(n) = \Theta(\log n)$.

(b) Analyze the following function and determine its running time. Express your answer in Θ-notation.

```
   Strange(n) {
1.     for i ← 1 to n do
2.         if (IsPerfectSquare(i)) then
3.             for j ← 1 to i do
4.                 print (i, j);
   }
```

The function call `IsPerfectSquare(i)` returns `True` if `i` is a perfect square (that is, $i = x^2$ for some integer x); `False` otherwise.

Solution: Lines 1-2 contribute $\Theta(n)$ to the running time. To calculate the contribution of Lines 3-4 note that for every perfect square i, $1 \leq i \leq n$, Lines 3-4 contribute $\Theta(i)$ to the running time. So the total contribution of Lines 3-4 is $\sum \Theta(i)$, where the summation is over all perfect squares i, $1 \leq i \leq n$. This sum can be alternately written as

$$\sum_{x=1}^{\lfloor \sqrt{n} \rfloor} \Theta(x^2) = \Theta((\sqrt{n})^3) = \Theta(n\sqrt{n}).$$

Hence the total running time of all the lines is $\Theta(n\sqrt{n})$.

4. (a) Let $X = x_1, x_2, \ldots, x_n$ be a sequence such that $x_1 > x_2 > \cdots > x_n$. What is the running time of following piece of code?

```
for i ← 1 to n do
    INSERT(H, x_i);
```

Assume that `H` is an empty heap before the above code is executed. Briefly justify your answer.

Solution: The running time is $\Theta(n)$.

Explanation: When x_i is inserted, H contains $i - 1$ elements. Placing x_i in slot i of H creates a heap of size i without any rearrangement of the elements. This is because x_i is the smallest element in the heap and is therefore smaller than its parent. So each insertion tales $\Theta(1)$ time for a total of $\Theta(n)$ for all the insertions.

(b) Write pseudocode for an algorithm that finds the kth smallest element in an array $A[1..n]$ in $\Theta(n \log k)$ worst case time. Use the heap data structure to do this.

Solution:

```
KSmallest(A[1..n]){
    H ← empty heap;
```

```
2.        for i ← 1 to k do
3.             INSERT(H, A[i]);

4.        for i ← k+1 to n do
5.             if (A[i] < MAX(H)) then{
6.                  EXTRACT-MAX(H);
7.                  INSERT(H);
               }
8.        return MAX(H);
     }
```

It was not necessary to provide an explanation for why this function takes time $\Theta(n \log k)$. But, here it is anyway. Line 1 takes $\Theta(1)$ time, Lines 2-3 take $\Theta(k \log k)$ time in the worst case, Lines 6-7 take $\Theta(\log k)$ in the worst case for each execution. Therefore, Lines 4-7 take $\Theta((n-k)\log k)$ in the worst case. Line 8 takes $\Theta(1)$ time, for a total of $\Theta(n \log k)$ time, in the worst case.

5. (a) Show how the `PARTITION` function, as described in your textbook, would work on the following sequence of numbers:

$$7, 8, 11, 2, 18, 9, 1, 3, 10.$$

Show how the array would look at the beginning of each execution of the while-loop. Clearly show where the indices i and j are pointing to.

Solution:

i	7	8	11	2	18	9	1	3	10	j
	3i	8	11	2	18	9	1	7j	10	
	3	1i	11	2	18	9	8j	7	10	
	3	1	2i	11j	18	9	8	7	10	
	3	1	2j	11i	18	9	8	7	10	

(b) Suppose that the input `QUICKSORT` is an array of n elements that contains c distinct elements, for some positive constant c. Describe how `PARTITION` would have to be modified so that `QUICKSORT` has a worst case running time $\Theta(n \log n)$.

Notes: You do not have to write pseudocode; write down each modification you would make to `PARTITION` and then write a brief explanation as to why `QUICKSORT` will have a worst case running time of $\Theta(n \log n)$.

Solution: Start with the `PARTITION` described in Problem 5, Homework 4 and modify it as follows.

- Start by scanning the array and record in an array `B[1..c]` each distinct element in `A`. This takes $\Theta(n)$ time.
- Attempt to partition the array at most c times, using each of the elements `B[i]`, $1 \le i \le c$, as a pivot. Since c is a constant, this amounts to making $\Theta(1)$ attempts. Each attempt takes $\Theta(n)$ time, for a total of $\Theta(n)$ time.
- Recall that the `PARTITION` code described in Homework 4 computes two indices q and r such that `A[q+1..r]` is the middle block containing elements, all equal to the pivot. Compute the index that points to the middle of the middle block: $\lfloor (q+r+1)/2 \rfloor$ and check if this index points to the middle of the whole array (i.e., $\lfloor (p+s)/2 \rfloor$). If so, the partitioning attempt stops; otherwise the next distinct element is used as a pivot.

The modified partition returns an index that points to the middle of the array. This means that the running time of the function is given by the recurrence $T(n) = 2T(n/2) + \Theta(n)$. This in turn implies that the running time of the function is $\Theta(n \log n)$.