*EE360C: Algorithms*
University of Texas at Austin
Dr. Soloveichik

Homework 6
Due: -

# Homework 6

The purpose of this homework is to give you practice with Divide and Conquer, to prepare you for Exam 2. **There is no quiz associated with this homework.**

**Note that Exam 2 is cumulative, while this homework covers only Divide and Conquer.**

**Master Theorem Reminder:**
Let $T(n)$ be defined by the recurrence $T(n) = aT(n/b)+f(n)$ (where $a \geq 1$ and $b > 1$ are constants). Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

## Problem 1: Master Method

Use the Master Theorem above to give a tight asymptotic bound for each of the following recurrences, or argue why it doesn't apply.

1. $T(n) = 3T(n/2) + \Theta(n)$

   **Solution**
   $a = 3$, $b = 2$. We compare $f(n) = \Theta(n)$ with $n^{\log_2 3} = n^{1.58}$. Since the second is polynomially larger (with $\epsilon = 1.58 - 1$), we have Case 1: $T(n) = \Theta(n^{\log_2 3})$.

2. $T(n) = 3T(n/2) + \Theta(n^2)$

   **Solution**
   $a = 3$, $b = 2$. We compare $f(n) = \Theta(n^2)$ with $n^{\log_2 3} = n^{1.58}$. The second is polynomially smaller (with $\epsilon = 2 - 1.58$), so we are candidates for Case 3. (We also check the technical condition: $3(n/2)^2 \leq cn^2$ for $c = 3/4 < 1$.) Thus: $T(n) = \Theta(n^2)$.

3. $T(n) = 16T(n/2) + \Theta(n^3 \lg n)$

   **Solution**
   $a = 16$, $b = 2$. We compare $f(n) = \Theta(n^3 \lg n)$ to $n^{\log_2 16} = n^4$. The second is polynomially larger so Case 1 applies: $T(n) = \Theta(n^4)$

4. $T(n) = 8T(n/2) + \Theta(n^3 \lg n)$

**Solution**

$a = 8$, $b = 2$. We compare $f(n) = \Theta(n^3 \lg n)$ with $n^{\log_2 8} = n^3$. Since neither is polynomially larger, our version of the Master Theorem is not applicable. (Another version of the Master Theorem can give the result $T(n) = \Theta(n^3(\log n)^2)$.)

## Problem 2: Value matches position

Suppose you are given a sorted sequence of *distinct* integers $\{a_1, a_2, \ldots a_n\}$. Give an $O(\log n)$ algorithm to determine whether there exists an index $i$ such that $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$; there is no such $i$ in $\{2, 3, 4, 5, 6, 7\}$. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method).

**Solution**

Let MATCH$(i, j)$ return true if $a_i = i$ or if $a_{i+1} = i + 1 \ldots$ or if $a_j = j$. We call this function at the beginning with MATCH$(1, n)$.

MATCH$(i, j)$

```
1  if i > j
2        return false
3  if i = j
4        if a_i = i return true
5  else return false
6  if i < j
7        m = ⌊(i+j)/2⌋
8      if a_m = m then return true
9      if a_m > m then return MATCH(i, m − 1)
10     if a_m < m then return MATCH(m + 1, j)
```

The recurrence for the above is derived as follows. It generates one subproblem of size $n/2$. The other work (outside of the recursive call) takes $O(1)$ time. So the recurrence is $T(n) = T(n/2)+O(1)$. Using the master method, $a = 1$, $b = 2$, and we compare $f(n) = O(1)$ with $n^{\log_b a} = n^0 = 1$. This is case 2, so the solution is $O(n^{\log_b a} \log n) = O(\log n)$.

### Problem 3: Circular shift

Suppose you are given an array $A$ of $n$ sorted numbers that has been *circularly shifted* to the right by $k$ positions. For example $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\{27, 29, 35, 42, 4, 15\}$ has been shifted $k = 4$ positions. Give an $O(\log n)$ algorithm to find the largest number in $A$. You may assume the elements of $A$ are distinct. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method).

**Solution**

Let MAX-CIRC$(i.j)$ return $x$ if $a_x$ is the max element in the array and 0 if no $x$ in the range $i, j$ is the max element. We call this function at the beginning with MAX-CIRC$(1, n)$.

MAX-CIRC$(i, j)$

1 $m = \lfloor \frac{i+j}{2} \rfloor$
2 **if** $A[m] > A[m-1]$ **and** $A[m] > A[m+1]$ **then return** $m$
3 **if** $A[m] > A[1]$ **then return** MAX-CIRC$(m+1, j)$
4 **if** $A[m] < A[1]$ **then return** MAX-CIRC$(i, m-1)$

I'm assuming the elements are distinct; there would be a couple of extra checks otherwise. The recurrence for the above is derived as follows. It generates one subproblem of size $n/2$. The other work (outside of the recursive call) takes $O(1)$ time. So the recurrence is $T(n) = T(n/2)+O(1)$. Using the master method, $a = 1$, $b = 2$, and we compare $f(n) = O(1)$ with $n^{\log_b a} = n^0 = 1$. This is case 2, so the solution is $O(n^{\log_b a} \log n) = O(\log n)$.

### Problem 4: Maximum sum

Given a list of integers $a_1, a_2, \ldots, a_n$ we are interested in finding a subsequence having maximum sum; i.e., if for $i \leq j$ we define $A_{i,j} = \sum_{i \leq k \leq j} a_k$, we want $i, j$ such that $A_{i,j}$ is maximum. For example, if the given list is $\{1, -5, 1, 9, -7, 9, -4\}$, the maximum subsequence is $A_{3,6}$. Give a divide-and-conquer algorithm for this problem. Write the recurrence for your algorithm and solve for its asymptotic upper bound (e.g., using the Master Method).

**Solution**

The maximum subsequence is either contained in the left half entirely, or in the right half entirely, or it straddles the middle. To handle each of the first two possibilities we do a recursive call on a sequence half as long. We handle the last possibility (straddles the middle) as follows: In linear time we can compute $A_{i,n/2}$ for $i = 1, \ldots, n/2$ and $A_{n/2,j}$ for $j = n/2, \ldots n$. We work out from the middle so once we have computed $A_{i,n/2}$ we only need to add to it $a_{i-1}$ to obtain $A_{i-1,n/2}$, etc. As we produce these sums, we keep track of the maximum of each sequence; i.e., the $i^*$ for which $A_{i^*,n/2}$ is maximum (and similarly for $A_{n/2,j^*}$ to get $j^*$.) The maximum subsequence that straddles the middle is $A_{i^*,j^*}$.

If $T(n)$ is the running time to solve the problem on input size $n$ integers, we obtain the recurrence: $T(1) = O(n), T(n) = 2T(n/2) + O(n)$, which has as solution $T(n) = O(n \log n)$ by the Master Theorem.