CSE 431/531: Algorithm Analysis and Design	Fall 2022

Homework 3

Instructor: Shi Li	**Deadline: 10/23/2022**

Your Name: Zheyuan Ma   Your Student ID: 50321597

Problems	1	2	3	4	Total
Max. Score	16	16	24	24	80
Your Score					

Problem 1 For each of the following recurrences, use the master theorem to give the tight asymptotic upper bound. You just need to give the final bound for each recurrence.

(a) $T(n) = 5T(n/3) + O(n)$. $T(n) = O(n^{\lg_3 5})$.

(b) $T(n) = 3T(n/3) + O(n)$. $T(n) = O(n \lg n)$.

(c) $T(n) = 4T(n/2) + O(n^2\sqrt{n})$. $T(n) = O(n^{2.5})$.

(d) $T(n) = 8T(n/2) + O(n^2)$. $T(n) = O(n^3)$.

Problem 2 Given two n-digit integers, you need output their product. Design an $O(n^{\log_2 3})$-time algorithm for the problem, using the polynomial-multiplication algorithm as a black box to solve the problem.

Assume the two n-digit integers are given by two 0-indexed arrays A and B of length n, each entry being an integer between 0 and 9. The i-th integer in an array corresponds to the digit with weight 10^i. For example, if we need to multiple 3617140103 and 3106136492, then the two arrays are $A = (3, 0, 1, 0, 4, 1, 7, 1, 6, 3)$ and $B = (2, 9, 4, 6, 3, 1, 6, 0, 1, 3)$. That is, $A[0] = 3, A[1] = 0, A[2] = 1$, etc. You need to output the product "11235330870604938676". You can assume the two integers both have n digits, and there are no leading 0's.

You can use the polynomial-multiplication algorithm as a black-box; you do not need to give its code/pseudo-code.

Problem 3 Suppose you are given n pictures of human faces, numbered from 1 to n. There is a face comparison program A that, given two different indices i and j from $1, 2, \cdots, n$, returns whether face i and face j are the same, i.e., are of the same person. A majority face is a face that appears more than $n/2$ times in the n pictures.

The problem, then, is to decide whether there is a majority face or not, using the algorithm A as a black box. You need to design and analyze an algorithm that only calls A $O(n \log n)$ times.

Remark. A can only return whether two faces i and j are the same or not. If they are not the same, A can *not* tell you whether "face $i <$ face j" or "face $i >$ face j".

Problem 2

Let multiply(A, B, n) to be the polynomial-multiplication algorithm.

```
int_multiply(A, B, n)
C ← multiply(A, B, n)
result ← 0
for i ← 0 to 2n−1 do
    if C[i] ≥ 10 then
        C[i+1] ← C[i+1] + (C[i] − C[i] % 10)/10
        C[i] ← C[i] % 10
    result ← result + C[i] × 10^i
return result
```

multiply(A, B, n) runs in $O(n^{\log_2 3})$ time

the rest of the algorithm runs in $O(n)$ time.

Problem 3

```
majority_face(l, r)
if l = r then return l
mid ← ⌊(l+r)/2⌋
f_L ← majority_face(l, mid)
f_R ← majority_face(mid+1, r)
if is_majority(l, r, f_L) then return f_L
if is_majority(l, r, f_R) then return f_R
return ⊥
```

If face f is the majority face among the face group G, then if we divide G into two equal sub-groups, the face f must be the majority face of one of the two sub-groups.

Call majority_face(1, n)

```
is_majority(l, r, f)
if f = ⊥ return false
count = 0
for i ← l to r do
    if A(i, f) then count ← count + 1
if count > (r − l + 1)/2 then return true
return false
```

$T(n) = 2T(n/2) + O(n)$

1 2 3 4 5 6 | 7 8 9 10 11 12

Example. Suppose $n = 5$ and the function calls to $\mathcal{A}$ and their returned values are as follows: $A(1, 2) =$ different, $A(1, 3) =$ different, $A(2, 3) =$ same, $A(3, 4) =$ different, $A(3, 5) =$ same. Then your algorithm can correctly return "yes" since it knows that faces 2, 3 and 5 are the same. *This example is only for the purpose of helping you understand the problem. You should not use it as a guide to design your algorithm.*

Problem 4 Given an array A of n **distinct** numbers, we say that some index $i \in \{1, 2, 3 \cdots, n\}$ is a local minimum of A, if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ (we assume that $A[0] = A[n+1] = \infty$). Suppose the array A is already stored in memory. Give an $O(\log n)$-time algorithm to find a local minimum of A. (There could be multiple local minimums in A; you only need to output one of them.)

Problem 4

```
local_minimum(ℓ, r)
if ℓ = r then return ℓ
mid = ⌊(ℓ+r)/2⌋
if A[mid] > A[mid+1] then
    return local_minmium(mid+1, r)
else
    return local_minmium(ℓ, mid)
```

Call local_minimum(1, n)

$$T(n) = T(n/2) + 1 = O(n^0 \log n) = O(\log n)$$