CS5800 Algorithms

Mid Term, 1st October, 2019

Ravi Sundaram

- **Duration, Problems, and Points:** This is a 90-minute exam. There are five problems, for a total of 100 points. First, read through all the problems and then attempt them in the order that allows you to make the most progress.
- **Presentation of solutions:** While describing an algorithm, you may use any of the algorithms covered in class as a subroutine, without elaboration.
- **Proofs and analyses:** You need to provide a proof of correctness and/or an analysis of the running time *only for* those problems where they are explicitly required.
- **A note on grading:** Your grade on any problem will be determined on the basis of the correctness of your solution and the clarity of the description. In case you are not able to present an algorithm that has the desired properties, give the best algorithm you have designed. Show your work, as partial credit may be given. If you need extra space, use the blank facing page. Write your name on the top of every page.
- **Closed-book and closed-notes exam:** This exam is closed-book. No sources of any kind, electronic or physical, can be consulted during the course of this exam. No collaboration of any kind is permitted.
- **Policy on cheating:** Students who violate the above rules on scholastic honesty are subject to disciplinary penalties. Any student caught cheating will *immediately* receive an **F** (failing grade), and the case will be forwarded to the Office of Student Conduct and Conflict Resolution.

  Sign below and confirm that you are strictly abiding by the rules of this test.

Signature: ______________________

- Best of luck!

Question	Score	Maximum
1		15
2		20
3		20
4		20
5		25
Total		100

Name: ______________________

Problem 1 (Recurrences and Asymptotic notation) 4+6+5

a. Indicate whether the following statements are True or False by circling T or F respectively:

   i. Searching a sorted array in the comparison-based model must take $\Omega(n)$ time because all the n elements must be inspected at least once. [**False**]

   ii. It is known that numbers cannot be factored in polynomial time.[**False**]

   iii. Given enough (finite) time, it is possible to decide whether a given program halts or not.[**False**]

   iv. Diffie-Hellman requires modular exponentiation to be fast but discrete log to be slow.[**True**]

b. Solve the following recurrences:

   i. $T(n) = 2T(n/2) + O(\sqrt{n}); \quad T(1) = 1$

   Solution:

   By the Master method, $n^{\log_2(2)} = n$ is greater than $n^{\frac{1}{2}}$. Hence, complexity must be $O(n)$. ■

   ii. $T(n) = 2T(n-1) + 3^n; \quad T(1) = 1$

   Solution:

$$\begin{aligned}
T(n) &= 2T(n-1) + 3^n \\
T(n) &= 2(2T(n-2) + 3^{n-1}) + 3^n \\
&\vdots \\
T(n) &= 2^n T(1) + 3^n + 2 \cdot 3^{n-1} + \cdots + 2^{n-1} \cdot 3 \\
T(n) &= 2^n + 2^{n-1} \cdot 3 + \cdots + 3^n \\
T(n) &= O(3^n)
\end{aligned}$$

■

c. Let $f(n), g(n)$ and $h(n)$ be positive, monotonically non-decreasing functions from $\mathbb{N} \to \mathbb{N}+$. Prove or disprove the following statement.

Solution:

If $f(n), g(n) = \Theta(h(n))$ then $f(n) \cdot g(n) = \Theta((h(n))^2)$.

Given $f(n), g(n) = \Theta(h(n))$, there must exist c_1, c_2, c_3 and c_4 such that for all $n \geq n_1, n_2$

$$\begin{aligned} f(n) &\geq c_1 \cdot h(n) \\ f(n) &\leq c_2 \cdot h(n) \\ g(n) &\geq c_3 \cdot h(n) \\ g(n) &\leq c_4 \cdot h(n) \end{aligned}$$

Hence, for $c_5 = c_1 \cdot c_3, c_6 = c_2 \cdot c_4$ both of the following inequalities hold:

$$\begin{aligned} f(n) \cdot g(n) &\geq c_5 \cdot h(n)^2 \\ f(n) \cdot g(n) &\leq c_6 \cdot h(n)^2 \end{aligned}$$

Thus, $f(n) \cdot g(n) = \Theta((h(n))^2)$. ■

Problem 2 (n-way merge) 5+5+5+5

You are given n arrays, each containing k elements in sorted order. You need to merge all the $n * k$ elements into a single sorted array. Suppose that comparison is a unit time operation.

a. (Direct n-way merge) Suppose at each step you compare the first element of each of the n arrays and insert the minimum of these into the output array. What is the time complexity of this procedure?

Solution:

Finding minimum of n elements in each step takes $n-1$ comparisons, and since we are putting in nk elements one at a time there are nk steps, hence complexity is $O(n^2k)$. ■

b. (Sorting) What would be the time complexity if you concatenated all the elements into the output array and then sorted the whole output array using an optimal algorithm like mergesort?

Solution:

Concatenating all arrays, element by element, takes nk time. Sorting it takes $O(nk\log(nk))$ time. Hence, total complexity is $O(nk\log(nk))$. ■

c. (Divide-and-Conquer) What if you merge the arrays in groups of 2, doing the standard merge operation on each pair of arrays. So after each stage, you would halve the number of arrays and double the number of elements in any given array. Continue this grouping and merging until you are left with a single array of size $n * k$. What would be the time complexity of this procedure?

Solution:

In the first stage, merging 2 arrays takes time $2k$ and after we are done, there are $\frac{n}{2}$ arrays. Total time taken $= 2k \cdot \frac{n}{2} = nk$. In stage two, we'll spend $4k$ time merging the $\frac{n}{4}$ pairs for a total of nk time. This continues for $\log_2(n)$ stages after which 1 array will be left. In the last stage, merging 2 arrays of size $\frac{nk}{2}$ each will take time nk. Hence, each stage takes time nk. So, total time taken is $O(nk \log_2(n))$.

■

d. (Using a MinHeap) Suppose you have access to a MinHeap of size n. A MinHeap is a data structure that can be queried to return its minimum element in time $O(\log n)$, and you can add a new element to it in $O(\log n)$ time. Suppose you do the direct n-way merge but use a MinHeap to find the minimum, as described in Algorithm 1. What is the complexity of this procedure?

Algorithm 1 K-Way Merge using Min Heap

```
1: procedure K-WAY–MERGE
2:     Let $A_1, \ldots, A_n$ be the $n$ sorted input arrays of size $k$ each.
3:     Create an output array $B$ of size $n * k$
4:     Create a min-heap $H$ of size $k$ and insert $A_i[0]$ into $H$ for $1 \le i \le n$.
5:     $i \leftarrow 0$
6:     for $i < n * k$ do
7:         Extract-Min from $H$, let the element be $m$; append $m$ to $B$.
8:         If $m$ came from $A_i$ then insert next element $A_i$ into the heap. If $A_i$ is empty insert $\infty$
```

Solution:

According to the algorithm, every element in the output array was extracted from the min-heap. Each of the nk elements took $\log(n)$ time to be inserted into the min-heap. Extracting it also takes $\log(n)$ time and putting it into the output array took constant time. So, the total time taken must be $O(nk \log(n))$. ■

Problem 3 (Missing Weights) 10+10

You are running a gym equipment shop, where you stock $n-4$ dumb-bells weighing from $5, 2 \ldots n$ pounds (all the weights are distinct). A recent audit revealed that one of the dumb-bells is missing. Sadly the markings on each of the dumb-bells has worn off and there is no way to tell how much they weigh by looking at them. However, you do have a digital weighing scale in your shop which gives you the total weight of all dumb-bells you put on it. The problem can be abstracted as: Given an array of size $n-5$, containing all but one of the numbers from 5 to n, find the number not present in the array.

a. Give an algorithm to find the missing number in time linear in n.

Solution:

Keep an boolean array of size $n-5$ and whenever you see number i in the input, mark position i in your boolean array as `True`. At the end, whichever position is still `False`, is the missing weight.

Algorithm 2 Missing weight in linear space

```
1: procedure FindMissingWeight_LIN_SPACE
2:     Initialize a boolean array B[1..n], setting each of its elements to False.
3:     for each input weight w do
4:         $B[w] \leftarrow True$
5:     $i \leftarrow 5$
6:     for $i <= n$ do
7:         if $B[i] == False$ then
8:             $res \leftarrow w$
       return $res$
```

■

b. Give an algorithm to find the missing number in time linear in n and constant space (you can use a constant number of temporary variables but you cannot use an auxiliary array).

Solution:

If you had all $n-5$ numbers then you know that the sum of these would be $n(n+1)/2 - 5 * 6/2$. So we can compare the expected sum with the actual sum of the weights and we'll know which weight is missing.

■

Algorithm 3 Missing weight in constant space

```
1: procedure FindMissingWeight_CONST_SPACE
2:     $sum \leftarrow 0$
3:     for each input weight w do
4:         $sum \leftarrow sum + w$
       return $\frac{n(n+1)}{2} - 15 - sum$
```

Problem 4 (Min-and-Max) 20

Given n distinct numbers, where n is an even number, give a scheme for finding the maximum and minimum values by performing $\frac{3n}{2} - 2$ comparisons.

Solution:

As n is even, we can pair up the numbers and have $\frac{n}{2}$ comparisons. Now we have $\frac{n}{2}$ highs and $\frac{n}{2}$ lows. Greatest value among highs can be found in $\frac{n}{2} - 1$ comparisons and the lowest among lows can be found in $\frac{n}{2} - 1$ comparisons. Total number of comparisons, is thus $\frac{3n}{2} - 2$. ■

Problem 5 (Minimum Contiguous Subarray) 25

Given an array of n integers, write an efficient algorithm using the divide and conquer technique to find the sum of a contiguous subarray which has the smallest sum. Argue the correctness of your algorithm and derive its time complexity. A contiguous subarray of $A[1 \dots n]$ is a subarray $A[i \dots j]$ for some $1 \leq i \leq j \leq n$.

Solution:

The procedure *MinSubArraySum* finds the minimum subarray sum (MSS) by splitting the array in half, recursively finding the MMS on the left and right halves using two recursive calls to *MinSubArraySum*. These two are then compared with the MSS found by considering subarrays which cross from the left to the right half of the array, computed by calling *MinCrossingSubArrSum*.

MinCrossingSubArrSum computes the minimum sum of a subarray (boundaries L to H) including the middle element ($\frac{L+H}{2}$) of the array. This takes time linear in the the length of the array.

The recurrence re;ation can be written as: $T(n) = 2T(n/2) + O(n)$ which gives a $O(n \log(n))$ bound on the complexity.

Algorithm 4 Find minimum crossing sum

```
procedure MinCrossingSubArrSum(ARR,L,M,H)
    $i \leftarrow M$
    $sumLeft \leftarrow sumRight \leftarrow \infty$
    while $i >= L$ do
        $sm \leftarrow sumLeft + ARR[i]$
        if $sm < sumLeft$ then
            $i \leftarrow i - 1$
            $sumLeft \leftarrow sm$
        else break
    $i \leftarrow M + 1$
    while $i <= R$ do
        $sm \leftarrow sumRight + ARR[i]$
        if $sm < sumRight$ then
            $i \leftarrow i + 1$
            $sumRight \leftarrow sm$
        else break
    return $sumLeft + sumRight$
```

Algorithm 5 Find minimum sum of subarray

```
procedure MinSubArraySum(ARR,L,H)
    if $L == H$ then return $ARR[L]$
    $M \leftarrow \frac{L+H}{2}$
     return  $MIN(MinSubArraySum(ARR, L, M),$
                $MinCrossingSubArrSum(ARR, L, M, H),$
                $MinSubArraySum(ARR, M + 1, H))$
```

■