CMPSC 465 Spring 2022 | Data Structures & Algorithms Chunhao Wang and Mingfu Shao | Solution 02

1. (0 pts.) Acknowledgements. The assignment will receive a 0 if this question is not answered.

1. If you worked in a group, list the members of the group. Otherwise, write "I did not work in a group."
2. If you received significant ideas about your solutions from anyone not in your group, list their names here. Otherwise, write "I did not consult anyone except my group members".
3. List any resources besides the course material that you consulted in order to solve the material. If you did not consult anything, write "I did not consult any non-class materials."

2. (10 pts.) Solve each of the following recursions using master's theorem. Give the closed form of $T(n)$ in Big-Theta notation; you don't need to show your middle steps.

1. $T(n) = 8 \cdot T(n/2) + 100 \cdot n^3$.
2. $T(n) = 8 \cdot T(n/2) + 1000 \cdot n^{3.5}$.
3. $T(n) = 16 \cdot T(n/2) + n \cdot \log(n)$
4. $T(n) = 2 \cdot T(n/2) + n \cdot \log(n)$.
5. $T(n) = 8 \cdot T(n/2) + n^{3.5} \cdot \log^2(n)$.

Solution:

The first 2 subproblems follow recursion $T(n) = a \cdot T(n/b) + \Theta(n^d)$, and the last 3 subproblems follow recursion $T(n) = a \cdot T(n/b) + \Theta(n^d \cdot \log^s n)$.

1. $a = 8, b = 2, d = 3$, $\log_b a = \log_2 8 = 3$, $d = \log_b a$, $T(n) = \Theta(n^d \log n) = \Theta(n^3 \log n)$.
2. $a = 8, b = 2, d = 3.5$, $\log_b a = \log_2 8 = 3$, $d > \log_b a$, $T(n) = \Theta(n^d) = \Theta(n^{3.5})$.
3. $a = 16, b = 2, d = 1, s = 1$, $\log_b a = \log_2 16 = 4$, $d < \log_b a$, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^4)$.
4. $a = 2, b = 2, d = 1, s = 1$, $\log_b a = \log_2 2 = 1$, $d = \log_b a$, $T(n) = \Theta(n^d \log^{s+1} n) = \Theta(n \log^2 n)$.
5. $a = 8, b = 2, d = 3.5, s = 2$, $\log_b a = \log_2 8 = 3$, $d > \log_b a$, $T(n) = \Theta(n^d \log^s n) = \Theta(n^{3.5} \log^2 n)$.

3. (10 pts.) Suppose you have m sorted arrays, each with n elements, and you want to combine them into a single sorted array with mn elements.

1. If you do this by merging the first two arrays, next with the third, then with the fourth, until in the end with the last one. What is the time complexity of this algorithm, in terms of m and n? Please also provide the analysis.
2. Give a more efficient solution to this problem, using divide-and-conquer. What is the running time? Please also provide the analysis.

Solution:

1. Each merge step requires $O(x + y)$ time for x elements in first array and y elements in the second array.

   Extending this, we can see that initially we start with $2n$ being the time required to merge the first two arrays. Then to merge with the third array it would take $3n$, and then $4n$ for the fourth and so on. Thus, it can be represented as:

$$T(m, n) = 2n+3n+\cdots+(m-1)n+mn = n(2+3+\cdots+(m-1)+m) = n\left(\frac{m(m+1)}{2} - 1\right) = \Theta(nm^2)$$

2. Without loss of generality, recursively divide all the arrays into two sets, each with $m/2$ arrays. Then, recursively combine the arrays within the two sets and finally merge the resulting two sorted arrays into the output array. The base case of the recursion is $m = 1$, where it's just a single array and no merging needs to take place. This procedure is illustrated in the pseudocode below (though not required from the answer).

Algorithm 1: mergeArrays(A, num)

Input : An array A containing num sorted arrays, each with n elements
Output: A single sorted array with all the elements in all the arrays in A
if num = 1 **then**
    return $A[0]$
else
    $merged1 \leftarrow$ mergeArrays ($[A[0], \cdots, A[\lceil \frac{num}{2} \rceil - 1]]$, $\lceil \frac{num}{2} \rceil$)
    $merged2 \leftarrow$ mergeArrays ($[A[\lceil \frac{num}{2} \rceil], \cdots, A[num - 1]]$, $\lfloor \frac{num}{2} \rfloor$)
    return merge-two-sorted-arrays($merged1$, $merged2$)
end

   Note: From this point, we use $T(m)$ to represent the running time. Although n is an input of our problem, the array size is constant across all of them. Also, with only one variable in the running time expression, it's more similar to the cases we see in the class where we apply the Master's theorem.

   For merging the two arrays, i.e., $merged1$ and $merged2$, we can see that it will be $\theta(nm)$. Thus, the running time is given by $T(m) = 2T(m/2)+\theta(nm)$. By the Master theorem, $T(m) = \Theta(nm \log m)$.

4. (10 pts.) If we add one more partition step in find-pivot function in selecting problem, and use 3 as the size of each subarray, the algorithm would become:

function find-pivot (A, k)
    Partition A into $n/3$ subarrays;
    Let M be the list of medians of these $n/3$ subarrays;
    Partition M into $n/9$ subarrays;
    Let M' be the list of medians of these $n/9$ subarrays;
    return selection(M', $|M'|/2$)
end function;

Analyze the running time of the new selection algorithm with the find-pivot function described above.

Solution. Let m be the median of array M'. Consider the numbers in M. In half of the $n/9$ subarrays, i.e., those with median that is less than m, there are two numbers (i.e., the median of this subarray and another number) that are guaranteed less than m. This amounts to $2 \cdot (n/9)/2 = n/9$ elements in M that are less than m. Now consider the numbers in A. Recall that each element in M is a median of a subarray of size 3 in A. As we already show that there are $n/9$ elements in M that are less than m, in the corresponding

subarrays, there are 2 numbers (the median and another number) that are guaranteed less than m. This amounts to $2 \cdot n/9 = 2n/9$ elements in A that are less than or equal to m. Thus, there are at most $7n/9$ elements are larger than m in A. Symmetrically, there are at most $7n/9$ elements are smaller than m in A.

The recursive call of selection function in above algorithm takes $T(n/9)$ time as $|M'| = n/9$. The recurrence relation for the entire algorithm is: $T(n) \leq \Theta(n) + T(n/9) + T(7n/9)$, which gives $T(n) = \Theta(n)$.

5. (10 pts.) Suppose you are given an array $A[1 \cdots n]$ of integers which can be positive, negative or 0. A sub-array is a contiguous sequence of elements from A. In particular, for any two indexes i and j with $1 \leq i \leq j \leq n$, $A[i \cdots j]$ is the sub-array that starts at index i and ends at j. The sum of the sub-array $A[i \cdots j]$ is the sum of all the numbers it contains: $A[i] + A[i+1] + ... + A[j]$. For example, if $A = [5, 1, -3, -2, 4, 0]$, the sum of $A[0 \cdots 3]$ is 1 and the sum of $A[1 \cdots 4]$ is 0. Given A, design a divide-and-conquer algorithm to find the sub-array of minimum sum. For example, in the array $A = [5, 1, -3, -2, 4, 0]$, the sub-array of minimum sum is $A[2, 3]$ with sum -5. Your algorithm should run in $O(n \log n)$ time.

Solution. We can solve this problem by following merge sort algorithm with few changes. Let's divide the array into 2 halves, splitting at middle element(say 'mid'). Now the minimum contiguous sub-array has 3 possible scenarios.

- Present in the Left half
- Present in the Left half
- sub-array starts with an element in the left half and ending at some element in the right half.

The first 2 cases can be recursively solved to find the subproblem result. For the third case, we can find the minimum crossing sum by finding the minimum contiguous sum subarray starting at some point in left half and ending at mid, another starting at mid + 1 and ending at some point in the right half, and combine both to return the crossing sum. Now our solution is the minimum value from all the three cases.

Algorithm 2: minSumSubArray(A,left,right)

Input : An array A[] of n integers
Output: minimum sum of the contiguous subarray in $A[left \cdots right]$
if *left = right* **then**
    return $A[left]$
end
$mid \leftarrow (left + right)/2$;
$l \leftarrow minSumSubArray(A, left, mid)$;
$r \leftarrow minSumSubArray(A, mid + 1, right)$;
$c \leftarrow crossingSum(A, left, mid, right)$;
return $min(l, r, c)$;

At each iteration the problem size gets reduced to half and finding the crossing sum requires linear time as we would be visiting each element at most once. So the recurence relation becomes, $T(n) = 2T(n/2) + \Theta(n)$. Following the master theorem we can see that the time complexity is $O(n \log n)$.

Algorithm 3: crossingSum (A,left,mid,right)

Input : An array A and three positions left, mid, right
Output: minimum sum of the contiguous subarray in $A[left \cdots right]$ that span the mid position

$i \leftarrow mid - 1$;
$leftSum \leftarrow A[mid]$;
$leftSumTemp \leftarrow A[mid]$;
while $i \geq left$ **do**
    if $leftSumTemp + A[i] < leftSum$ **then**
        $leftSum \leftarrow leftSumTemp + A[i]$;
    end
    $leftSumTemp \leftarrow leftSumTemp + A[i]$;
    $i \leftarrow i - 1$;
end
$j \leftarrow mid + 2$;
$rightSum \leftarrow A[mid + 1]$;
$rightSumTemp \leftarrow A[mid + 1]$;
while $j \leq right$ **do**
    if $rightSumTemp + A[j] < rightSum$ **then**
        $rightSum \leftarrow rightSumTemp + A[j]$;
    end
    $rightSumTemp \leftarrow rightSumTemp + A[j]$;
    $j \leftarrow j + 1$;
end
return $leftSum + rightSum$;

6. (0 pts.) *(NOTE: you don't need to submit your solution for this problem.)* Consider recurrence relation $T(n) = \Theta(n) + T(a \cdot n) + T(b \cdot n)$, $T(1) = 1$, $0 < a < 1$, $0 < b < 1$. Prove the following:

1. $T(n) = \Theta(n)$, if $a + b < 1$.
2. $T(n) = \Theta(n \cdot \log n)$, if $a + b = 1$.

Solution. Assume that the term $\Theta(n)$ in the recursion admits a lower bound of $d_1 n$ and upper bound of $d_2 n$, for large enough n. That is $T(n) \geq d_1 n + T(an) + T(bn)$ and $T(n) \leq d_2 n + T(an) + T(bn)$.

1. We first prove that $T(n) = O(n)$, by induction. Assume that $T(n) \leq c_2 n$ for large enough n, where $c_2 = d_2/(1 - a - b)$. Now we prove that $T(n + 1) \leq c_2(n + 1)$. We can write

$$T(n + 1) \leq d_2(n + 1) + T(a(n + 1)) + T(b(n + 1)).$$

   We have $a(n + 1) \leq n$ and $b(n + 1) \leq n$ for large enough n as $a < 1$ and $b < 1$. By the inductive assumption we have

$$\begin{aligned} T(n+1) &\leq d_2(n+1) + c_2 a(n+1) + c_2 b(n+1) \\ &\leq d_2(n+1) + c_2(a+b)(n+1) \\ &= (d_2 + c_2(a+b))(n+1) \\ &= c_2(n+1). \end{aligned}$$

   It's easy to verify the last equation, i.e., $d_2 + c_2(a + b) = c_2$ with $c_2 = d_2/(1 - a - b)$. In fact this is where we determine the value of c_2. You can also see why $a + b < 1$ is required here. We then prove that $T(n) = \Omega(n)$, by induction. Assume that $T(n) \geq c_1 n$ for large enough n, where $c_1 = d_1/(1 - a - b)$. Now we prove that $T(n + 1) \geq c_1(n + 1)$. We can write

$$\begin{aligned} T(n+1) &\geq d_1(n+1) + T(a(n+1)) + T(b(n+1)) \\ &\geq d_1(n+1) + c_1 a(n+1) + c_1 b(n+1) \\ &\geq d_1(n+1) + c_1(a+b)(n+1) \\ &= (d_1 + c_1(a+b))(n+1) \\ &= c_1(n+1). \end{aligned}$$

   The last equation holds as $d_1 + c_1(a + b) = c_1$ with $c_1 = d_1/(1 - a - b)$.

2. We first prove that $T(n) = O(n \log n)$, by induction. Assume that $T(n) \leq c_2 n \log n$ for large enough n, where $c_2 = -d_2/(a \log a + b \log b)$. Now we prove that $T(n + 1) \leq c_2(n + 1) \log(n + 1)$. We can write

$$\begin{aligned} T(n+1) &\leq d_2(n+1) + T(a(n+1)) + T(b(n+1)) \\ &\leq d_2(n+1) + c_2 a(n+1)\log(a(n+1)) + c_2 b(n+1)\log(b(n+1)) \\ &\leq d_2(n+1) + c_2(a\log a + b\log b)(n+1) + c_2(a+b)(n+1)\log(n+1) \\ &= (d_2 + c_2(a\log a + b\log b))(n+1) + c_2(n+1)\log(n+1) \\ &= c_2(n+1)\log(n+1). \end{aligned}$$

   The last equation holds as $d_2 + c_2(a \log a + b \log b) = 0$ with $c_2 = -d_2/(a \log a + b \log b)$. We then prove that $T(n) = \Omega(n \log n)$, by induction. Assume that $T(n) \geq c_1 n \log n$ for large enough n, where

$c_1 = -d_1/(a\log a + b\log b)$. Now we prove that $T(n+1) \geq c_1(n+1)\log(n+1)$. We can write

$$\begin{aligned} T(n+1) &\geq d_1(n+1) + T(a(n+1)) + T(b(n+1)) \\ &\geq d_1(n+1) + c_1 a(n+1)\log(a(n+1)) + c_1 b(n+1)\log(b(n+1)) \\ &\geq d_1(n+1) + c_1(a\log a + b\log b)(n+1) + c_1(a+b)(n+1)\log(n+1) \\ &= (d_1 + c_1(a\log a + b\log b))(n+1) + c_1(n+1)\log(n+1) \\ &= c_1(n+1)\log(n+1). \end{aligned}$$

The last equation holds as $d_1 + c_1(a\log a + b\log b) = 0$ with $c_1 = -d_1/(a\log a + b\log b)$.

Rubrics:

Problem 2, 10 pts

Each part has 2 points.
2 - Provide a correct answer

Problem 3, 10 pts

1. 2 points: Finds the correct running time to merge two arrays.
   2 points: Provides the correct final answer.
2. 2 points: Identifies the size and the number of sub-problems that can result in a more efficient algorithm.
   2 points: Provides the correct merging time for each recursion.
   2 points: Provides the correct running time for the improved algorithm.

Problem 4, 10 pts

1. 5 points : Analyzed the function find-pivot appropriately.
2. 3 points : Provided a proper recurrence.
3. 2 points : Provided the answer.

Problem 5, 10 pts

1. 3 points : identified 3 possible cases.
2. 3 points : Analysed/explained crossing sum(case 3)
3. 2 points : Provided proper recurrence.
4. 2 points : solved the recurrence for time complexity.
5. 4 points : correct runtime analysis without master theorem. theorem/recurrence

Students can provide algorithm/pseudocode, but they should explain the correctness or how the algorithm works. The 5th case in rubric for problem 5 is when students follow any other methods(tree analysis, iterative analysis) to analyse the running time and get the answer correctly.