Problem Set 1 Solutions

This problem set is due **at 11:59pm** on **Thursday, September 15, 2016.**

EXERCISES (NOT TO BE TURNED IN)

Asymptotic Analysis, Recursion, and Master Theorem

- Do Exercise 4.3-7 in CLRS on page 87.
- Do Exercise 4.3-9 in CLRS on page 88.

Divide and Conquer Algorithms

- Do Exercise 4.2-3 in CLRS on page 82.
- Do Exercise 9.3-1 in CLRS on page 223.

Problem 1-1. Solving Recurrences [50 points]

Let $T(n)$ be the time complexity of an algorithm to solve a problem of size n. Assume $T(n)$ is $O(1)$ for any n less than 3. Solve the following recurrence relations for $T(n)$.

1. [10 points] $T(n) = 3T\left(\frac{n}{9}\right) + \sqrt{n}$

   Solution: By case 2 of the Master Theorem, since $f(n) = \sqrt{n}$ is $\Theta(n^{\log_9 3})$, then $T(n) = \Theta(\sqrt{n}\log n)$.

2. [15 points] $T(n) = 4T(\sqrt[4]{n}) + \log^2 n$.

   Solution: We solve this one by changing the variables. Assume that n is a power of 2, and let $n = 2^m$, then
   $$T(2^m) = 4T(2^{m/4}) + m^2.$$
   If we define $S(m) = T(2^m)$, the recurrence becomes
   $$S(m) = 4S(m/4) + m^2.$$
   We can use the Master Theorem (case 3), since $\log_4 4 < 2$ and the regularity condition holds $(4(m/4)^2 < m^2)$. Therefore, $S(m) = \Theta(m^2)$, and $T(2^m) = \Theta(m^2)$. Changing the variable back ($m = \log n$), we have $T(n) = \Theta(\log^2 n)$.

3. [15 points] $T(n) = 4\,T(n-1) + 5\,T(n-2)$.

   Solution: The characteristic equation for the above recurrence relation is $x^2 - 4x - 5$. The roots of this equation are -1 and 5, leading to a general solution of the form $O(5^n + (-1)^n) = O(5^n)$.

4. [10 points] $T(n) = 2\,T(n/3) + 2\,T(n/6) + n$.

   Solution: By the recursion tree method, in each level we spend n, and we have $\log n$ levels, resulting in $O(n \log n)$ total work.

Problem 1-2. Solving Recurrences [50 points]

Let $T(n)$ be the time complexity of an algorithm to solve a problem of size n. Assume $T(n)$ is $O(1)$ for any n less than 3. Solve the following recurrence relations for $T(n)$.

1. [10 points] $T(n) = 9T\left(\frac{n}{3}\right) + n^2 + n\log n$

   Solution: By case 2 of the Master Theorem, since $f(n) = n^2 + n\log n \in \Theta(n^{\log_3 9})$, then $T(n) = \Theta(n^2 \log n)$.

2. [15 points] $T(n) = 2\,T(\sqrt[3]{n}) + \log n$.

   Solution: We solve this one by changing the variables. Assume that n is a power of 2, and let $n = 2^m$, then
$$T(2^m) = 2\,T(2^{m/3}) + m.$$
   If we define $S(m) = T(2^m)$, the recurrence becomes
$$S(m) = 2\,S(m/3) + m.$$
   We can use the Master Theorem (case 3), since $\log_3 2 < 1$ and the regularity condition holds ($2(m/3) < m$). Therefore, $S(m) = \Theta(m)$, and $T(2^m) = \Theta(m)$. Changing the variable back ($m = \log n$), we have $T(n) = \Theta(\log n)$.

3. [15 points] $T(n) = 3\,T(n-1) + 4\,T(n-2)$.

   Solution: The characteristic equation for the above recurrence relation is $x^2 - 3x - 4$. The roots of this equation are -1 and 4, leading to a general solution of the form $O(4^n + (-1)^n) = O(4^n)$.

4. [10 points] $T(n) = 2\,T(n/5) + 6\,T(n/10) + n$.

   Solution: By the recursion tree method, in each level we spend n time, and we have $\log n$ levels, resulting in $O(n \log n)$ total time.

Problem 1-3. The k-th coldest day in history [50 points] Andy just got an internship position at a company. The company has a very large dataset of the city temperatures from the past several decades. Since the dataset is too large, the previous intern tried to summarize it. She constructed an array A of size n that contains all the unique values of the temperatures in the dataset. Then, she constructed an array W with the same size as A such that $W[i]$ indicates how many days in the dataset has the temperature $A[i]$. Andy's boss asked him to come up with an algorithm to find the temperature of the k-th coldest day in the big data set in $O(n)$ time. Andy cannot use the dataset directly and has to use A and W only. Note that A is not necessarily sorted. Help Andy to solve his problem.

For example, if the dataset contained $\{70, 60, 75, 60, 70, 70\}$, A and W would be $[70, 60, 75]$ and $[3, 2, 1]$ respectively. Moreover, the temperature of the fourth coldest day would be 70.

Hint: Think about how the median might be helpful.

Solution: For an element e in A, we define rank of the element e as the following
$$\text{rank}(e) := \sum_{i:A[i]<e} W[i].$$
Now, the question is equivalent to find the largest element e in A such that $\text{rank}(e) < k$. First, we find the median of A, namely m, in $O(n)$ using the selection algorithm. Now, we use m as a pivot to divide the array into two groups:

1. The elements that are less than m
2. The elements that are larger than or equal to m.

Now, we compute the score of m in $O(n)$. If rank(m) is greater than or equal to k, we know that e cannot be in the second group. Thus, we solve the problem for the first group. If rank(m) is less than k, we know that e is not in the first group. We solve the problem for the second half and we are looking for the ($k-$rank(m))-th element. For the case where $n \leq 2$, we can solve the problem directly by checking $W[1] < k$ in constant time.

Runtime analysis: If we write the recurrence relation, we have

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

By substituting, we have $T(n) = O(n) + O(n/2) + O(n/4) + \ldots + O(1)$. Since the geometric series, $1 + 1/2 + 1/4 + \ldots$, is bounded by 2, The total running time is $O(n)$.

Problem 1-4. The chocolate lover [50 points]

Andy is organizing a chocolate-lovers party. He needs to buy at least w ounces of chocolate for the party. In the store, there are n chocolate bags each with the weight w_i (and they are not necessarily sorted based on the weights). Andy wants to buy the smallest number of bags. Provide an algorithm with running time $O(n)$ to help Andy.

Hint: Think about how the median might be helpful.

Solution: First, we show that the greedy algorithm that picks the next heaviest bag until collecting enough chocolates is optimal. Then, we show how to run it in $O(n)$ time.

An optimal solution: Let $i_1, i_2, \ldots, i_n$ be the indices of all the bags such that $w_{i_1} \geq w_{i_2} \geq \ldots \geq w_{i_n}$. We call a subset $S \subseteq [n]$ a *valid solution* if $\sum_{i \in S} w_i$ is at least w. Let k be the smallest number such that

$$\sum_{j=1}^{k} w_{i_j} \geq w.$$

Obviously, $S = \{i_1, i_2, \ldots, i_k\}$ is a valid solution. We show that any other valid solution has to have at least k bags. The proof is by contradiction. Let S' be a valid solution with less than k bags that maximize $|S \cap S'|$. S' cannot be a subset of S, since we assumed k is the smallest index such that the sum of the weights are at least w. Now, assume S' is not a subset of S. Thus, there exist an element $x \in S'$ such that x in not in S. Also, since $|S'|$ is less than $|S|$, there exists an element $y \in S$ such that y is not in S'. Since S contains the heaviest elements it is clear that $w_x \leq w_y$. Now, let $S'' = (S' \setminus \{x\}) \cup \{y\}$. It is clear that S'' is a valid solution and it has the same number of elements as S', but $|S \cap S''|$ is greater than $|S \cap S'|$. This contradicts the fact that S' maximizes $|S \cap S'|$. Therefore, S is an optimal solution.

Algorithm: Now, the problem is equivalent to find the largest element e such that

$$\sum_{i:w_i \geq e} w_i \geq w.$$

First, we find the median of w_i's, namely m, in $O(n)$ using the selection algorithm. Now, we use m as a pivot to divide the array into two groups:

1.The elements that are less than m

2.The elements that are larger than or equal to m.

Now, we compute the total weight, namely w', of all the elements in the second group. If w' is smaller than w, all the elements in the second group are in the solution and the element e cannot be in the second group. Thus, we solve the problem for the first group and we are looking for $w - w'$ ounces of chocolate. If w' is greater than or equal to w, we know that none of the elements in the first group is in the solution. Thus, we solve the problem for the second group. For the case where $n \leq 2$, we can solve the problem directly by checking $\max(w_1, w_2) > w$ in constant time.

Runtime analysis: If we write the recurrence relation, we have

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

By substituting, we have $T(n) = O(n) + O(n/2) + O(n/4) + \ldots + O(1)$. Since the geometric series, $1 + 1/2 + 1/4 + \ldots$, is bounded by 2, The total running time is $O(n)$.