CSI - 3105 Design & Analysis of Algorithms

Fall 2021

Assignment #1

Deadline: Tuesday, September 28, 2021 before 10:00 AM (to be delivered in BrightSpace)

Note: Make sure your handwriting is readable, otherwise your assignment will not be marked.

1. (10 marks) Assume that $n = 2^k$, where k is a positive integer, and solve the following recurrence by unfolding.

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2\,T\left(\frac{1}{2}n\right) + \log_2(n) & \text{if } n \geq 2. \end{cases}$$

Your final answer must be an exact formula in terms of n (do not use O-notation). Your final answer must be simplified as much as possible. You must provide all details of your solution.

Solution: Assuming that $n = 2^k$, we have

$$\begin{aligned}
T(n) &= 2\,T\left(\frac{1}{2}n\right) + \log_2(n) \\
&= 2\left(2\,T\left(\frac{1}{2^2}n\right) + \log_2\left(\frac{1}{2}n\right)\right) + \log_2(n) \\
&= 2^2\,T\left(\frac{1}{2^2}n\right) + 2\log_2\left(\frac{1}{2}n\right) + \log_2(n) \\
&= 2^2\left(2\,T\left(\frac{1}{2^3}n\right) + \log_2\left(\frac{1}{2^2}n\right)\right) + 2\log_2\left(\frac{1}{2}n\right) + \log_2(n) \\
&= 2^3\,T\left(\frac{1}{2^3}n\right) + 2^2\log_2\left(\frac{1}{2^2}n\right) + 2\log_2\left(\frac{1}{2}n\right) + \log_2(n) \\
&= 2^3\left(2\,T\left(\frac{1}{2^4}n\right) + \log_2\left(\frac{1}{2^3}n\right)\right) + 2^2\log_2\left(\frac{1}{2^2}n\right) + 2\log_2\left(\frac{1}{2}n\right) + \log_2(n) \\
&= 2^4\,T\left(\frac{1}{2^4}n\right) + 2^3\log_2\left(\frac{1}{2^3}n\right) + 2^2\log_2\left(\frac{1}{2^2}n\right) + 2\log_2\left(\frac{1}{2}n\right) + \log_2(n) \\
&\;\;\vdots
\end{aligned}$$

$$= 2^k\, T\left(\frac{1}{2^k} n\right) + \sum_{i=0}^{k-1} 2^i \log_2\left(\frac{1}{2^i} n\right)$$

$$= 2^k\, T\left(\frac{1}{2^k} n\right) + \sum_{i=0}^{k-1} 2^i \left(\log_2(n) - i\right)$$

$$\text{since } \log_2\left(\frac{1}{2^i} n\right) = \log_2(n) - \log_2\left(2^i\right) = \log_2(n) - i,$$

$$= 2^k\, T\left(\frac{1}{2^k} n\right) + \log_2(n) \sum_{i=0}^{k-1} 2^i - \sum_{i=0}^{k-1} i\, 2^i$$

$$= 2^k\, T\left(\frac{1}{2^k} n\right) + \log_2(n)\left(2^k - 1\right) - \left((k-2)2^k + 2\right)$$

$$\text{since } \sum_{i=0}^{k-1} 2^i = 2^k - 1 \text{ and } \sum_{i=0}^{k-1} i\, 2^i = (k-2)2^k + 2,$$

$$= 2^k + \log_2(n)\left(2^k - 1\right) - \left((k-2)2^k + 2\right)$$

$$\text{since } T\left(\frac{1}{2^k} n\right) = T(1) = 1,$$

$$= n + \log_2(n)\,(n-1) - \left((\log_2(n) - 2)\, n + 2\right)$$

$$\text{since } n = 2^k,$$

$$= 3n - \log_2(n) - 2.$$

2. (10 marks) Design a deterministic algorithm to solve the following problem.

input: An array $A[1..n]$ of n integers.

output: Three different indices i, j and k such that $A[i] + A[j] + A[k] = 2021$, if such indices exist. Otherwise, return **NONE**.

Your algorithm must be deterministic. Your algorithm must take $O(n^2)$ time. You must describe your algorithm in plain English (no pseudocode) and you must explain why the running time of your algorithm is $O(n^2)$.

Solution: Be careful: we need to return the indices i, j and k, not the numbers $A[i]$, $A[j]$ and $A[k]$. Moreover, if we sort A, we destroy the original indices.

So first, create a secondary table B in the following way. Scan A and for each number $x \in A$, store the pair (x, k) in B, where k is the index of x in A. To do that, we have to visit each position in A once. This takes $O(n)$ time. For example, from $A = [1, -5, -140, 2028, 42, -8]$, we get $B = [(1,1), (-5,2), (-140,3), (2028,4), (42,5), (-8,6)]$. If $B[i] = (x, k)$, we denote the number x by $B[i,1]$ and the index k by $B[i,2]$. For instance, we have $B[3,1] = -140$ and $B[3,2] = 3$ in our example.

Second, sort B (with respect to the first number in each pair) using Merge sort. This takes $O(n \log(n))$ time. For instance, $[(1,1),(-5,2),(-140,3),(2028,4),(42,5),(-8,6)]$ becomes $[(-140,3),(-8,6),(-5,2),(1,1),(42,5),(2028,4)]$.

Finally, for each number $B[i,1]$ in B (for all values of i from 1 to $n-2$), we do the following. Consider two indices j and k starting at $j = i+1$ and $k = n$. As long as $j < k$, scan B in the following way. If $B[k,1] = 2021 - B[i,1] - B[j,1]$, then return $B[i,2]$, $B[j,2]$ and $B[k,2]$, and then STOP. If $B[k,1] > 2021 - B[i,1] - B[j,1]$, then let $k = k-1$. Otherwise, if $B[k,1] < 2021 - B[i,1] - B[j,1]$, then let $j = j+1$.

If all scans (for all the numbers $B[i,1]$) were unsuccessful, then return **NONE** and then STOP. For each number $B[i,1]$, we scan B at most once. So we spend at most $O(n)$ time for each number $B[i,1]$. Therefore, in total, this step takes $O(n^2)$ time. In our example, the scan for $B[1,1] = -140$ is unsuccessful. Then the scan for $B[2,1] = -8$ starts with $j = 3$ and $k = 6$. Since $B[k,1] = 2028 < 2034 = 2021 - B[i,1] - B[j,1]$, we do $j = j+1 = 4$. Then, since $B[k,1] = 2028 = 2021 - B[i,1] - B[j,1]$, we return $B[i,2] = 6$, $B[j,2] = 1$ and $B[k,2] = 4$.

In total, this algorithm takes $O(n) + O(n \log(n)) + O(n^2) = O(n^2)$ time.