ICCS200: Assignment 2

Hasdin Ghogar(Harvey)
hghogar@gmail.com
08/05/2019
Collaborators:Chakeera, Ink, Taem.

1: Poisoned Wine

We will label the testers as 1 and 0. 1 if they are poisoned and 0 if they are not. We test n-1 bottle and leave 1 bottle out. If no one gets poisoned then the bottle left out is the poisoned bottle. For 2^n bottles we use n bits to label each bottle and every i_{th} bit is assigned to one tester only. If the tester is labeled 1 on that bottle they have to drink the wine in that bottle. Therefore by doing this we check every possible result by labeling the 7 remaining bottle(for 8 bottles) as 000,001,010,011,100,101,110,111(8 possibilities). This way we can the scheme meets the O(log(n)) tester requirements in 31 days.

2: More Running Time Analysis

Determine the best case running time and the worst case running time of method 1 in terms of Θ.

(1) Assigning n as an integer of array.length costs $\Theta(1)$. The cost of for loop in helpermethod1 is $\Theta(n)$ while the other line in that method cost $\Theta(1)$. While the method swap costs $\Theta(1)$. The for loop in method1 cost $\Theta(n)$ as it runs n-1 times. We run both helpermethod and swap in it therefore the cost of the program is $\Theta(n^2 - n).c(constant) + n.k(constant)$=$\Theta(n^2)$. Since the best-case and worst-case are same both of them are $\Theta(n^2)$

(2) Assigning n as an integer of array.length costs $\Theta(1)$. The for loop runs n times therefore it's cost is $\Theta(n)$. In best-case it runs only once therefore it costs $\Theta(1)$. In worst-case it runs n times before returning therefore it's cost is $\Theta(n)$.

(3) Assigning n as an integer of array.length costs $\Theta(1)$. The outer loop costs constant c. The middle loop runs 2*n times costing $\Theta(n)$. The inner loop costs $\Theta(logn)$. Therefore the total cost of the program $\Theta(nlogn)$. Since it is a for loop therefore the best-case and worst-case is the same. Therefore in both case the cost is $\Theta(nlogn)$.

3: Recursive Code

(1) The loop run n times each times cost k(constant). So, the cost for this loop is n.k, which is O(n). In the program it returns ys which is the length of xs/2 which is n/2 therefore the recurrence of it is T(n/2). The rest of the lines cost O(1) therefore the total cost for this recursion is T(n/2) + O(n) = O(n)

(2) The loop runs n time and each time cost k(constant). So the total cost is O(n). Others

work cost O(1). In the program it returns ys which is the length of (xs-1) which is n-1 using ys=copyOfRange(xs, 1, xs.length), therefore the recursion is T(n-1). The total work is T(n-1) + O(n) = O(n^2)

(3) First and the second for loop runs n/2 time each.
int[] left=Arrays.copyOfRange(xs, 0, n/2) takes n/2.
int[] right=Arrays.copyOfRange(xs, 0, n/2) takes n/2.
int[] ps = new int[xs.length] cost n.
Others function only cost O(1).
There are two recursive part of the program which are
left = prefixSum(left);
right = prefixSum(right);
Since both left=Arrays.copyOfRange(xs, 0, n/2) and right=Arrays.copyOfRange(xs, 0, n/2) costs T(n/2) each, therefore it's cost is 2T(n/2) In the program it returns ys which is the length of xs/2 which is n/2 therefore the recursion is 2T(n/2). This means the run time of the program takes $2T$(n/2) + O(n) = O(n$\log_2 n$).