CISC 621: Algorithm Design and Analysis

H/W H: Partition

Model Answer

1. A call to 'partition' first calculates the sum of the elements in the set taking Θ(n) time. It then calls 'SubsetSum' on the n elements if the sum is even. Thus the time taken by 'partition' in the worst case is:

   $T(n) = T_{SubsetSun}(n) + \Theta(n)$

   In the worst case every call to 'SubsetSum' will make two recursive calls to itself on an input of size (n – 1). Thus the time taken by 'SubsetSum' on n elements is given by the following recurrence relation:

   $T_{SubsetSun}(n) = 2\, T_{SubsetSun}(n - 1) + \Theta(1)$ .1

   The above recurrence may be solved in two ways:

   1. Using the substitution method:
      Since the question tells us that we can prove $O(2^n)$, we guess that the solution is $T_{SubsetSun}(n) = O(2^n)$. To prove by induction we note that the recurrence holds for the boundary conditions n = 0 and for n = 1 and we assume that it holds for $T_{SubsetSun}(n - 1)$, that is:

      $T_{SubsetSun}(n - 1) \le c2^{n-1} - b$ .2

      where b is some constant. Substituting 2 in 1 (replacing Θ(1) by a constant d), we get:

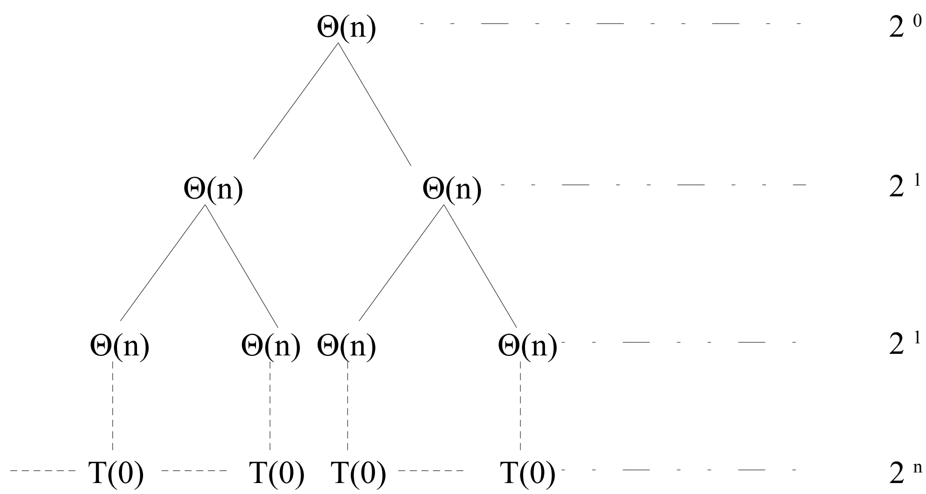
      $T_{SubsetSun}(n) = 2\,(c2^{n-1} - b) + d$
      $T_{SubsetSun}(n) = c2^{n} - 2b + d$
      $T_{subsetSun}(n) \le c2^{n} - b$

      Where the last inequality holds if $(2b - d) \le b$ or $b \le d$ for all values of c. Thus we get:

      $$\mathbf{T(n) = O(2^n)}$$

   2. Using the recurrence tree method:
      Drawing the recurrence tree (shown overleaf), we get:

      $T_{subsetSun}(n) = \Theta(1)\,(2^0 + 2^1 + 2^2 + 2^3 + 2^4 \ldots \ldots 2^n)$
      $T_{subsetSun}(n) = \Theta(1)\Sigma_{(i=0)}^{n}(2^i) = \Theta(1)\frac{(2^{(n+1)}-1)}{(2-1)} = \Theta(1)(2^{(n+1)}-1) = O(2^n)$



2. The above worst case of $O(2^n)$ was obtained by assuming the binary tree that we get is complete and has a depth of n. To prove that the worst case has a lower bound of 2^n (that is, the worst case is $\Omega(2^n)$) we have to find a sequence of n elements that result in a recurrence tree that is a complete binary tree with a depth of n. To get such a recurrence tree, assume a data set 2, 4, 8,... 2^n. In this data set, the last term is greater than the sum of all the previous terms. For this data set, the test `if (T < *b)` will be false for all but the last element of the data set. Hence, in each case other than the last element, we will get two calls to SubsetSum and hence we will obtain a recurrence tree which is a complete binary tree of depth n. Thus, by the recurrence tree shown above, we get:

$$\mathbf{T(n) = \Omega(2^n)}$$

3. We may do **slightly** better in the average case by performing a number of checks:
   1. If a single element is greater than the sum of half the elements, return false.
   2. Perform a check for $T \leq 0$, if true, return false.

   However, we **cannot** do much better in the worst case as this problem has a run time of $\Theta(2^n)$ and hence belongs to a class of NP complete problems and till date no one has been able to prove that NP = P.