Analysis of Algorithms - Quiz II (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. **Recurrences:** Solve the following recurrences using the Master method:

   (i)

$$\begin{aligned} T(1) &= 0 \\ T(n) &= 2 \cdot T(\frac{n}{2}) + \log n, \ n > 1 \end{aligned}$$

   (ii)

$$\begin{aligned} T(1) &= 0 \\ T(n) &= 9 \cdot T(\frac{n}{3}) + n^3 \log n, \ n > 1 \end{aligned}$$

   Solution:

   (i) Referring to the format of the Master Theorem, $a = 2$, $b = 2$ and $f(n) = \log n$. Hence, $\log_b a = 1$ and $n^{\log_b a} = n$. Clearly, $f(n) \in O(n^{1-\epsilon})$, for any $0 < \epsilon < 1$ and therefore, $T(n) \in \Theta(n)$, by the Master Theorem.

   (ii) Observe that, $a = 9$, $b = 3$ and $f(n) = n^3 \log n$ in the format of the Master Theorem. Hence, $\log_b a = \log_3 9 = 2$ and $n^{\log_b a} = n^2$. Clearly, $f(n) \in \Omega(n^{2+\epsilon})$, for $\epsilon = 1$ and therefore, $T(n) \in \Theta(n^3 \log n)$.

   □

2. **Divide-And-Conquer (Application)** Use Strassen's matrix mutiplication algorithm to multiply

$$\mathbf{X} = \begin{bmatrix} 3 & 2 \\ 4 & 8 \end{bmatrix} \text{ and } \mathbf{Y} = \begin{bmatrix} 1 & 5 \\ 9 & 6 \end{bmatrix}.$$

   Solution: We set $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y}$ and partition each matrix into four submatrices as discussed in class. Accordingly, $\mathbf{A} = [3]$, $\mathbf{B} = [2]$, $\mathbf{C} = [4]$, $\mathbf{D} = [8]$, $\mathbf{E} = [1]$, $\mathbf{F} = [5]$, $\mathbf{G} = [9]$ and $\mathbf{H} = [6]$, where,

$$\mathbf{Z} = \begin{bmatrix} \mathbf{I} & \mathbf{J} \\ \mathbf{K} & \mathbf{L} \end{bmatrix}, \mathbf{X} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \text{ and } \mathbf{Y} = \begin{bmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix}$$

   Applying Strassen's algorithm, we compute the following products:

   (i) $S_1 = \mathbf{A} \cdot (\mathbf{F} - \mathbf{H}) = [3] \cdot ([5] - [6]) = [-3]$.

   (ii) $S_2 = (\mathbf{A} + \mathbf{B}) \cdot \mathbf{H} = ([3] + [2]) \cdot [6] = [30]$.

   (iii) $S_3 = (\mathbf{C} + \mathbf{D}) \cdot \mathbf{E} = ([4] + [8]) \cdot [1] = [12]$.

(iv) $S_4 = \mathbf{D} \cdot (\mathbf{G} - \mathbf{E}) = [8] \cdot ([9] - [1]) = [64]$.

(v) $S_5 = (\mathbf{A} + \mathbf{D}) \cdot (\mathbf{E} + \mathbf{H}) = ([3] + [8]) \cdot ([1] + [6]) = [77]$.

(vi) $S_6 = (\mathbf{B} - \mathbf{D}) \cdot (\mathbf{G} + \mathbf{H}) = ([2] - [8]) \cdot ([9] + [6]) = [-90]$.

(vii) $S_7 = (\mathbf{A} - \mathbf{C}) \cdot (\mathbf{E} + \mathbf{F}) = ([3] - [4]) \cdot ([1] + [5]) = [-6]$.

From the above products, we can compute **Z** as follows:

(i) $\mathbf{I} = S_5 + S_6 + S_4 - S_2 = 21$

(ii) $\mathbf{J} = S_1 + S_2 = 27$

(iii) $\mathbf{K} = S_3 + S_4 = 76$

(iv) $\mathbf{L} = S_1 - S_7 - S_3 + S_5 = 68$

The correctness can easily be verified using the naive algorithm. □

3. **Divide-And-Conquer (Theory)** Design a Divide-And-Conquer strategy to find both the maximum and the minimum elements of an integer array using at most $\frac{3n}{2}$ comparisons. Analyze your algorithm through a recurrence relation. Note that the strategy discussed in the Midterm solutions is *not* Divide-And-Conquer.

Solution:

Without loss of generality, we assume that the number of elements in the input array is 2^k, for some $k \geq 1$.

Function FIND-MAXMIN(**A**, low, $high$)

if $((high - low + 1) = 2)$ **then**
  if $(\mathbf{A}[low] \leq \mathbf{A}[high])$ **then**
    $max = \mathbf{A}[high]$; $min = \mathbf{A}[low]$
    return(max, min)
  else
    $max = \mathbf{A}[low]$; $min = \mathbf{A}[high]$
    return(max, min)
  end if
else
  $mid = \frac{high+low}{2}$
  $(max_l, min_l) =$FIND-MAXMIN$(\mathbf{A},\ low,\ mid)$
  $(max_r, min_r) =$FIND-MAXMIN$(\mathbf{A},\ mid+1,\ high)$
  if $(max_l \geq max_r)$ **then**
    $max = max_l$
  else
    $max = max_r$
  end if
  if $(min_l \leq min_r)$ **then**
    $min = min_l$
  else
    $min = min_r$
  end if
end if
return(max, min)

Algorithm 1.1: Divide and Conquer for Minimum and Maximum

Algorithm (1.1) is called as FIND-MAXMIN(**A**, 1, n) from the main program.

Let $T(n)$ denote the comparison complexity of Algorithm (1.1). We have,

$$\begin{aligned} T(2) &= 1 \\ T(n) &= 2 \cdot T(\frac{n}{2}) + 2 \end{aligned}$$

We have assumed that $n = 2^k,\ k \geq 1$. We thus have,

$$\begin{aligned} T(2^1) &= 1 \\ T(2^k) &= 2 \cdot T(2^{k-1}) + 2 \end{aligned}$$

Therefore,

$$\begin{aligned} T(n = 2^k) &= 2 \cdot [2 \cdot T(2^{k-2}) + 2] + 2 \\ &= 2^2 \cdot T(2^{k-2}) + 2^2 + 2 \\ &= 2^{k-1} \cdot T(2^{k-(k-1)}) + 2^{k-1} + \ldots 2^2 + 2 \\ &= 2^{k-1} \cdot T(2) + 2 \cdot [1 + 2 + \ldots 2^{k-2}] \\ &= 2^{k-1} + 2 \cdot [2^{k-1} - 1] \\ &= 2^{k-1} + 2 \cdot 2^{k-1} - 2 \\ &= 3 \cdot 2^{k-1} - 2 \\ &= \frac{3n}{2} - 2 \end{aligned}$$

□

4. **Greedy:** Let $G =< \mathbf{V}, \mathbf{E} >$ denote an undirected graph with vertex set $\mathbf{V}$ and edge set $\mathbf{E}$. Assume that the weights on the edges of G are distinct, i.e., no two edges have the same weight. Argue that G has a unique Minimum Spanning Tree. *Hint: Recall the proof of correctness of Kruskal's algorithm and modify it ever so slightly!*

   Solution: Let T be the Minimum Spanning Tree (MST) produced by Kruskal's algorithm and let T' be another MST, which is different from T. Since both T and T have exactly $(n-1)$ edges, there must be at least one edge e, such that $e \in T$, but $e \notin T'$ and at least one edge e', such that $e' \in T'$, but $e' \notin T$. Without loss of generality, let e denote the lightest edge which belongs to T, but not to T'; likewise, let e' denote the lightest edge which belongs to T', but not to T. It follows that all edges e_r which are lighter than both e and e' are in both T and T' or in neither. Let S denote the set of edges, which are lighter than both e and e' and are present in both T and T'. We claim that e is lighter than e'. To see this, observe that if e' were lighter than e and could exist with the edges in S, then Kruskal's algorithm would have considered it first. Since e and e' are different edges, they cannot have the same weight (this is where the distinctness of the edge weights kicks in) and thus, e must be lighter than e'. Insert e into T'; a cycle C will be created. We now claim that at least one edge in this cycle, say e_f, does not belong to S. To see this, observe that if all the edges in C, were also in S, then e forms a cycle with the edges in S, which contradicts the fact that T is a spanning tree. Since $e_f \notin S$, it must be the case that e_f is heavier than e (they cannot have the same weight, since they are distinct edges). On removing e_f from C, we get a spanning tree which is lighter than T', contradicting the hypothesis that T' was a Minimum Spanning Tree.

   It follows that G must have a unique Minimum Spanning Tree.

   □

5. **Dynamic Programming:** Assume that you are given a chain of matrices $< A_1\ A_2\ A_3\ A_4 >$, with dimensions 2×5, 5×4, 4×2 and 2×4 respectively. Compute the optimal number of multiplications required to calculate the chain product.

Solution: Let $m[i,j]$ denote the optimal number of multiplications to multiply the chain $< A_i, A_{i+1}, \ldots, A_j >$, where matrix A_i has dimensions $d_{i-1} \times d_i$. As per the discussion in class, we know that

$$\begin{aligned} m[i,j] &= 0, \text{ if j} \leq \text{i} \\ &= \min_{k: i \leq k < j} m[i,k] + m[k+1,j] + d_{i-1} \cdot d_k \cdot d_j \end{aligned}$$

Computing $\mathbf{M} = [m[i,j]], i = 1,2,3,4;\ j = i, i+1, \ldots, 4$, in bottom-up fashion, we get

$$\mathbf{M} = \begin{bmatrix} 0 & 40 & 56 & 72 \\ 0 & 0 & 40 & 80 \\ 0 & 0 & 0 & 32 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

As per the above table, the optimal number of multiplications to multiply the given chain is 72. $\square$