Advanced Analysis of Algorithms - Homework I (Solutions)

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

1 Problems

1. Given an array **A** of n integer elements, how would you find the second smallest element in $n + \log_2 n$ comparisons.

   Solution: Consider the following algorithm:

```
Function FIND-2MIN(A, low, high)
n = high − low + 1
S_{2min} = φ
if (n = 1) then
   min_w = A[1]
   S_{2min} = φ.
   return(min_w, S_{2min})
end if
if (n = 2) then
   if (A[1] ≤ A[2]) then
      min_w = A[1]
      Add A[2] to S_{2min}
      return(min_w, S_{2min})
   else
      min_w = A[2].
      Add A[1] to S_{2min}
      return(min_w, S_{2min})
   end if
end if
{We know that n ≥ 3}
mid = ⌊(high+low)/2⌋
(lmin_w, lS_{2min}) = FIND-2MIN(A, low, mid)
(rmin_w, rS_{2min}) = FIND-2MIN(A, mid + 1, high)
if (lmin_w ≤ rmin_w) then
   min_w = lmin_w
   S_{2min} = lS_{2min} ∪ rmin_2
else
   min_w = rmin_w
   S_{2min} = rS_{2min} ∪ lmin_2
end if
return(min_w, S_{2min})
```

Algorithm 1.1: Finding the two smallest elements in an array

The above algorithm returns the smallest element in the whole array min_w and a set S_{2min} of candidate elements for the second minimum element.

The number of comparisons is characterized by the following recurrence relation:

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

This recurrence is easily solved to get $T(n) = n - 1$.

The size of the candidate set S_{2min} can be characterized by the following recurrence:

$$\begin{aligned} G(1) &= 0 \\ G(2) &= 1 \\ G(n) &= 1 + G(\frac{n}{2}) \end{aligned}$$

$G(n)$ is easily seen to be $\log_2 n$. We can find the smallest element in S_{2min} using at most $\log_2 n$ comparisons; it thus follows that the second smallest element can be found in $n + \log_2 n$ comparisons. $\Box$

2. Indicate whether each of the following identities is true or false, giving a proof if true and a counterexample otherwise.

(a) $f(n) + o(f(n)) \in \Theta(f(n))$.

(b) $(f(n) \in O(g(n))) \wedge (g(n) \in O(h(n))) \Rightarrow (f(n) \in O(h(n)))$.

(c) $\log^{1/\epsilon} n \in O(n^\epsilon)$, $(\forall \epsilon)\ 0 < \epsilon < 1$.

(d) $2^n \in \Omega(5^{\log_e n})$.

Solution:

(a) The key observation is that $o(f(n)) \in O(f(n))$. Also, $f(n) \leq f(n) + o(f(n))$; it follows that $f(n) + o(f(n) \in \Theta(f(n))$.

(b) The premises state that $f(n) \leq c_1 g(n)$ and $g(n) \leq c_2 h(n)$. It follows that $f(n) \leq c_1 \cdot c_2 h(n)$ and hence $f(n) \in O(h(n))$.

(c) Observe that

$$\begin{aligned} & \lim_{n\to\infty} \frac{log^{1/\epsilon} n}{n^\epsilon} \\ = & \lim_{n\to\infty} \frac{\frac{1}{\epsilon}\log\log n}{\epsilon \log n} \\ = & \ 0 \text{ by applying L'Hospital's rule} \end{aligned}$$

The identity is therefore true.

(d) Observe that

$$\begin{aligned} & \lim_{n\to\infty} \frac{2^n}{5^{\log n}} \\ = & \lim_{n\to\infty} \frac{n \log 2}{\log n \cdot 5} \\ \to & \ \infty \text{ by applying L'Hospital's rule} \end{aligned}$$

It therefore follows that the identity is true.

$\Box$

Function FIND-KLARGEST(**A**, k, n)

We assume that the array elements are stored in $\mathbf{A}[1]$ through $\mathbf{A}[n]$ and that k is an integer $\in [1, n]$. We also assume without loss of generality, we assume that the numbers are distinct.
if $(n = 1)$ **then**
  $\{k$ has to be 1 as well$\}$
  return($\mathbf{A}[n]$)
end if
We consider a variation of the PARTITION() procedure in which elements *larger* than the pivot are thrown in the left subarray and elements *smaller* than the pivot are thrown in the right subarray.
Partition **A** using $\mathbf{A}[1]$ as the pivot, using the above PARTITION() procedure. Let j denote the index returned by PARTITION(). $\{$As per the mechanics of PARTITION(), elements $\mathbf{A}[1]$ through $\mathbf{A}[j-1]$ are greater than $\mathbf{A}[j]$ and elements $\mathbf{A}[j+1]$ through $\mathbf{A}[n]$ are smaller than $\mathbf{A}[j]$.$\}$
Copy the elements larger than $\mathbf{A}[j]$ into a new array **C** and the elements smaller than $\mathbf{A}[j]$ into a new array **D**.
if $((k = j)$ **then**
  return($\mathbf{A}[j]$)
else
  if $(k < j)$ **then**
    return(FIND-KLARGEST(**C**, k, $(j-1)$))
  else
    return(FIND-KLARGEST(**D**, $k - j$, $(n-j)$))
  end if
end if

Algorithm 1.2: Selection through Partition

3. Devise a Divide-and-Conquer procedure for computing the k^{th} largest element in an array of integers. Analyze the asymptotic time complexity of your algorithm. (*Hint: Use the Partition procedure discussed in class.*)

   Solution:

   Algorithm (1.2) represents a Divide-and-Conquer strategy for our problem.

   The worst case running time of the algorithm is captured by the recurrence:

$$\begin{aligned} T(1) &= O(1) \\ T(n) &= T(n-1) + O(n) \end{aligned}$$

   This implies that algorithm runs in time $O(n^2)$ in the worst case. $\square$

4. Argue the correctness of the MERGE() procedure discussed in class. (*Hint: Write a recursive version of* MERGE() *and then use induction.*)

Solution:

Function MERGE($\mathbf{A}, l_1, h_1, \mathbf{B}, l_2, h_2, \mathbf{C}, l_3, h_3$)
1: {We assume that the arrays $\mathbf{A}[l_1 \cdots h_1]$ and $\mathbf{B}[l_2 \cdots h_2]$ are being merged into the array $\mathbf{C}[l_3 \cdots h_3]$. }
2: **if** (**A** is empty) **then**
3:    Copy **B** into **C**
4:    **return**
5: **end if**
6: **if** (**B** is empty) **then**
7:    Copy **A** into **C**
8:    **return**
9: **end if**
10: **if** ($\mathbf{A}[l_1] \leq \mathbf{B}[l_2]$) **then**
11:    $\mathbf{C}[l_3] = \mathbf{A}[l_1]$
12:    MERGE($\mathbf{A}, l_1 + 1, h_1, \mathbf{B}, l_2, h_2, \mathbf{C}, l_3 + 1, h_3$)
13: **else**
14:    $\mathbf{C}[l_3] = \mathbf{B}[l_2]$
15:    MERGE($\mathbf{A}, l_1, h_1, \mathbf{B}, l_2 + 1, h_2, \mathbf{C}, l_3 + 1, h_3$)
16: **end if**

Algorithm 1.3: Recursive Merge

To prove the correctness of the algorithm, we use induction on the sum s of the elements in **A** and **B**.

Clearly if $s = 1$, the algorithm functions correctly, since it copies the non-empty array into **C**.

Assume that the algorithm works correctly when $1 \leq s \leq k$, for some $k > 1$. Now consider the case in which $s = k + 1$. In this case, Step $(10 :)$ of the algorithm moves the smallest element in both **A** and **B** into the first position in **C**. This is followed by a recursive call on arrays whose total cardinality is at most k. By the inductive hypothesis, the recursive calls work correctly and since $\mathbf{C}[l_3]$ is already in its correct place, we can conclude that the arrays **A** and **B** have been correctly merged into array **C**.

□

5. What is the value returned by Algorithm (1.4) when called with $n = 10$?

Function LOOP-COUNTER(n)
$count = 0$
for ($i = 1$ **to** n) **do**
   for ($j = 1$ **to** i) **do**
      for ($k = 1$ **to** j) **do**
         $count++$
      end for
   end for
end for
return($count$)

Algorithm 1.4: Loop Counter

Solution:

For arbitrary n, it is clear that the value of $count$ is given by:

$$count(n) = \Sigma_{i=1}^{n} \Sigma_{j=1}^{i} \Sigma_{k=1}^{j} 1$$

$$
\begin{aligned}
&= \Sigma_{i=1}^{n}\Sigma_{j=1}^{i} j \\
&= \Sigma_{i=1}^{n}\frac{i\cdot(i+1)}{2} \\
&= \frac{1}{2}\cdot[\Sigma_{i=1}^{n} i^2 + \Sigma_{i=1}^{n} i] \\
&= \frac{1}{2}\cdot[\frac{n\cdot(n+1)\cdot(2n+1)}{6} + \frac{n\cdot(n+1)}{2}] \\
&= \frac{n\cdot(n+1)}{4}\cdot[\frac{2n+1}{3} + 1]
\end{aligned}
$$

It follows that $count(10) = \frac{55}{2} \times 8 = 220$. □