Problem Solutions to CLRS

Zeaiter Zeaiter

Contents

1 Chapter 2

2.1–2

DECREASING-INSERTION-SORT(A)

```
for i = 1 to A.length − 1
    key = A[i]
    j = i − 1
    while j > 0 and A[j] < key
        A[j + 1] = A[j]
        j = j − 1
    A[i + 1] = key
```

2.1–3

LINEAR-SEARCH(A, v)

```
1 for i = 0 to A.length − 1
2     if A[i] == v
3         return i
4 return NIL
```

Loop Invariant: At the start of each iteration of the **for** loop (lines 1–4) $i - 1$ is not an index of A such that $A[i-1] = v$.

Proof. Let us now prove the correctness of our algorithm. Suppose $i = 0$, then $i - 1$ is clearly not an index of A and hence $A[i-1]$ is undefined. Now suppose the loop invariant is true for some i, that is, $i - 1$ is not an index of A such that $A[i-1] = v$, or equivalently, $A[i-1] \neq v$. Then at line 3 the **if** loop will **return** i if $A[i] = v$, in which case the **for** loop terminates and there is no further iteration. Otherwise, if $A[i] \neq v$ then at the start of the next for loop iteration $(i+1) - 1$ is not an index of A such that $A[(i+1)-1] = v$. Finally, for termination to occur we have either $i = n + 1$ where $n = A$.length in which case the algorithm returns NIL indicating v is not an element of A. Otherwise, termination occurs because of the nested **if** on line 3 which causes the algorithm to return i which indicates the index of A such that $A[i] = v$. $\square$

2.1–4

Input: Two sequences of n integers, $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_n)$, such that $0 \leq a_i, b_i \leq 1$ for $i = 1, \ldots, n$. Least significant digits are first.

Output: An array $C = (c_1, \ldots, c_n, c_{n+1})$ such that $0 \le c_i \le 1$ for $i = 1, \ldots, n+1$ and $C' = A' + B'$ where $\cdot'$ is the integer represented by $\cdot$.

```
BINARY-ADDITION(A, B)
define integer C[A.length + 1]
overflow = 0
for i = 0 to A.length − 1
      C[i] = (A[i] + B[i] + overflow) % 2
      overflow = (A[i] + B[i] + overflow)/2
C[i] = overflow
return C
```

2.2–1

The function is $O(n^3)$

2.2–2

SELECTION-SORT(A)		cost	times
1	**for** $i = 0$ **to** A.length − 2	c_1	n
2	min = i	c_2	$n-1$
3	**for** $j = i+1$ **to** A.length−1	c_3	$\sum_{i=0}^{n}(n-i+1)$
4	**if** $A[j] < A[\text{min}]$	c_4	$\sum_{i=0}^{n}(n-i)$
5	min = j	c_5	$\sum_{i=0}^{n} t_i$
6	$M = A[\text{min}]$	c_6	$n-1$
7	$A[\text{min}] = A[i]$	c_7	$n-1$
8	$A[i] = M$	c_8	$n-1$

Loop Invariant: At the start of each iteration of the **for** loop (lines 1–8) the sub-array $A[0 \ldots i]$ is sorted in non-decreasing order.

The algorithm only needs to run for the first $n-1$ elements since this will arrange the $n-1$ smallest elements in non-decreasing order, ensuring the n^{th} element at the end is in the appropriate position. That is, $A[n] \ge A[i]$ for $i = 0, \ldots, n-2$.

The best-case running time occurs when the given array is already sorted from smallest to largest. In such a case $t_i = 0$ since we never need to re-assign the minimum index. The runtime equation is,

$$\begin{aligned}
T(n) &= c_1 n + (c_2 + c_6 + c_7 + c_8)(n-1) + c_3 \sum_{i=0}^{n}(n-i+1) + c_4 \sum_{i=0}^{n}(n-i) \\
&= c_1 n + (c_2 + c_6 + c_7 + c_8)(n-1) + c_3 \left((n+1) + \frac{n}{2}(n+1)\right) + c_4 \left(n + \frac{n}{2}(n-1)\right) \\
&= (c_3 + c_4)\frac{n^2}{2} + (c_1 + c_2 + c_6 + c_7 + c_8 + \frac{3}{2}c_3 + \frac{1}{2}c_4)n + (c_2 + c_6 + c_7 + c_8 + c_3)
\end{aligned}$$

and so the best-case running time is $O(n^2)$. In a worst-case scenario, the array given to the procedure is in descending order, however this would only include an additional term to $T(n)$ above,

$$c_5 \sum_{i=0}^{n}(n-1) = c_5 \left(n + \frac{n}{2}(n-1)\right) = \frac{1}{2}c_5(n^2+n)$$

since here line 5 will re-assign the minimum for all remaining entries in the array. So the runtime in a worst-case scenario is also $O(n^2)$.

2.2–3

LINEAR-SEARCH(A, v)	cost	times
for $i = 0$ **to** A.length $- 1$	c_1	$n+1$
if $A[i] == v$	c_2	n
return i	c_3	t_1
return NIL	c_4	t_2

If each of the n elements of A have equal probability p to be v then the expected value is,

$$E[v] = 0 \times \frac{1}{n} + 1 \times \frac{1}{n} + 2 \times \frac{1}{n} + \cdots + n \times \frac{1}{n} = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{n}{2}(n+1) = \frac{n+1}{2}$$

and hence on average we need to search through $\frac{n+1}{2}$ elements to find v. In the worst case we need to search n elements since v is not present in A. We have the following runtime equation,

$$T(n) = c_1(n+1) + c_2 n + c_3 t_1 + c_4 t_2$$

In the average-case $t_1 = \frac{1}{2} = t_2$ then,

$$T(n) = (c_1 + c_2)n + c_1 + \frac{1}{2}(c_3 + c_4)$$

and so the runtime is $O(n)$. In the worst-case $t_1 = 0$ and $t_2 = 1$ so the runtime equation is,

$$T(n) = (c_1 + c_2)n + c_1 + c_3$$

and so we still have $O(n)$ runtime.

2.2–4

Implement a checking loop/statement to return the procedure if in a best-case scenario. For example in Selection-Sort we can implement an initial loop that checks if the given array is already in sorted order and then return,

	cost	times
for $i = 0$ **to** A.length $- 2$	c_1	n
if $A[i] > A[i+1]$	c_2	$n - 1$
break	c_3	t_1
if $i == A$.length $- 2$	c_4	1
return	c_5	t_2

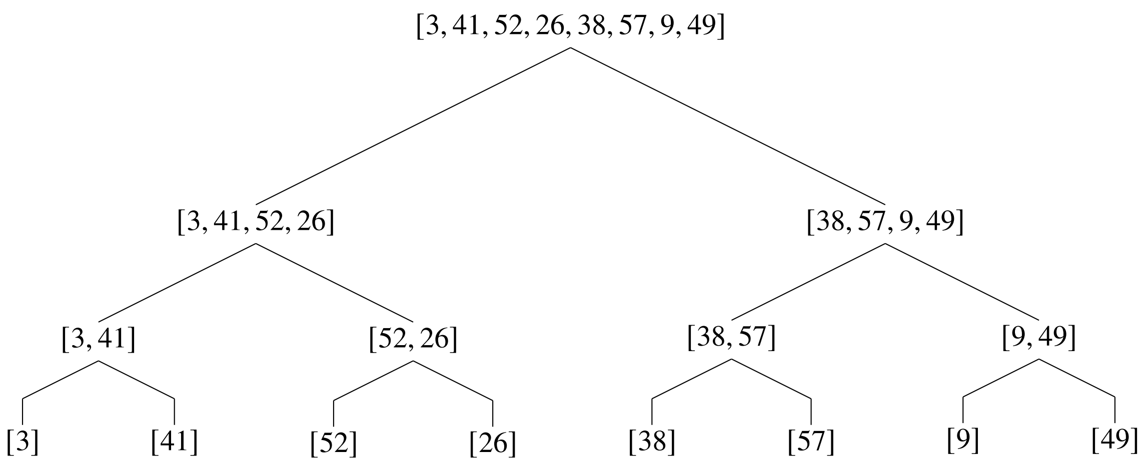
In such a case the runtime will be,

$$T(n) = (c_1 + c_2)n - c_2 + c_4 + c_5$$

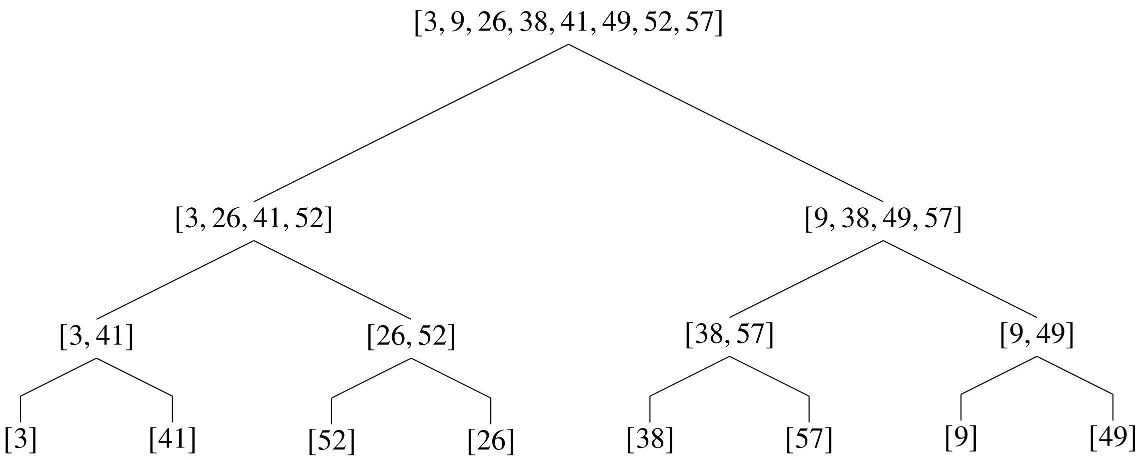
which is $O(n)$ a significant improvement over $O(n^2)$ in the above exercise.

2.3–1

We first divide the array into sub-arrays until we have arrays of length 1.



Then we merge to eventually recover the original array in sorted order.



2.3–2

```
Merge(A, p, q, r)
n1 = q − p
n2 = r − q − 1
define integers L[0 . . . n1] and R[0 . . . n2]
for i = 0 to n1
    L[i] = A[p + i]
for j = 0 to n2
    R[j] = A[q + j + 1]
i = 0
j = 0
for k = p to r
    if i > n1
        A[k] = R[j]
        j = j + 1
    elseif j > n2
        A[k] = L[i]
        i = i + 1
    elseif L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
```

2.3–3

Proposition 1.1. *If $n = 2^k$ for $k \in \mathbb{N} \setminus \{0\}$ then the solution of,*

$$T(n) = \begin{cases} 2 & \text{if } k = 1 \\ 2T(n/2) + n & \text{if } k > 1 \end{cases}$$

is $T(n) = n \lg n$.

Proof. If $k = 1$ we have $n = 2$ so $T(2) = 2 = 2 \lg 2$. Now assume this is true for some $k = m > 1$ then $T(2^m) = 2^m \lg 2^m$ so for 2^{m+1} we have the recurrence,

$$\begin{aligned} T(2^{m+1}) = 2T(2^{m+1}/2) + 2^{m+1} &= 2T(2^m) + 2 \cdot 2^m \\ &= 2 \cdot 2^m \lg 2^m + 2 \cdot 2^m \\ &= 2^{m+1} (\lg 2^m + 1) \\ &= 2^{m+1} (\lg 2^m + \lg 2) = 2^{m+1} \lg 2^{m+1} \end{aligned}$$

Hence the solution for any $n = 2^k$, $k \in \mathbb{N} \setminus \{0\}$, is $T(n) = n \lg n$. □