

22000 Algorithms, Summer 2004, CCNY CUNY

Instructor: Natalia Novak

Practice problems for Midterm Exam

1. Prove that the recurrence $T(n) = T(n-1) + 2T(n-2)$ is $\Omega(2^n)$;
 $T(0) = 3, T(1) = 15$ - boundary conditions

solution:

Let's make an inductive assumption that $T(n) \geq c \cdot 2^n$,
then $T(n) \geq c \cdot 2^{n-1} + 2 \cdot c \cdot 2^{n-2} = c \cdot 2^{n-1} + c \cdot 2^{n-1} = c \cdot 2^n$,
so $T(n) \geq c \cdot 2^n$

Now let's check the boundary conditions:

for $n = 2$ $T(2) = 2 \cdot T(1) + T(0) = 2 \cdot 15 + 3 = 33 \geq c \cdot 2^2$, holds as long
as $0 < c \leq \frac{21}{4}$

2. Draw a recursion tree to determine a good asymptotic upper bound on
the recurrence $T(n) = 3T(\frac{n}{3}) + 4T(\frac{n}{4}) + n^2$

solution: see 2.jpg (link can be found on the course's Notices -page).

3. Is an array that is in reverse sorted order a heap (max-heap in terms of
new book)?

solution: Let's recall the heap (or max-heap) property:

$A[\text{Parent}(i)] \geq A[i]$ holds for any node i (i is the index of a node) except
the root node.

And we remember that

$\text{Parent}(i)$
return $\lfloor \frac{i}{2} \rfloor$

So if we assume that the array that is in the reverse sorted order is not
a heap, then there exist a node x such that $A[x] \geq A[y]$, where y if
the parent of x . Since y is the parent node of the node x , $x \geq y$ (because
 $y = \lfloor \frac{x}{2} \rfloor$ thus $x > y$) therefore $A[x]$ should be less than or equal to $A[y]$,
because the array A is in reverse sorted order and index x is greater than
or equal to y - contradiction.

Hence our assumption is wrong and an array that is in reverse sorted order
is a heap.

4. Suppose that the **for** loop header in line 9 of the Counting-sort procedure
is rewritten as

9 **for** $j \leftarrow 1$ to $\text{length}[A]$

Show that the algorithm still works properly.

solution: lets prove that for two indices i, j ($i \neq j$) the order in which
operations ($B[C[A[j]]] \leftarrow A[j]$ and $C[A[j]] \leftarrow C[A[j]] - 1$) are performed (for
 i first then for j , or vice versa) is not important.

Consider two cases: $A[i] = A[j]$ and $A[i] \neq A[j]$

case 1. if $A[i] = A[j]$ then it means that $C[A[j]] = C[A[i]] \geq 2$, let $C[A[i]]$

= C[A[j]] = m. If we perform those two operations for index i first: B[m]=B[C[A[i]]] ← A[i] and C[A[i]] is decremented (C[A[i]] = C[A[j]] = m-1 ≥ 1 now)

and then for index j: B[m-1]=B[C[A[j]]] ← A[j] and C[A[j]] is decremented (C[A[j]] = C[A[i]] = m-2).

The elements A[i] and A[j] are near each other (A[i] is followed by A[j] in array B). And if we do it vice versa we'll receive that A[i] and A[j] are near each other also, but A[j] is followed by A[i] now. So the output array is still sorted.

case 2. if $A[i] \neq A[j]$ then $C[A[i]] \neq C[A[j]]$ and the order in which the elements A[i] and A[j] are placed in their correct sorted positions is not important ($B[C[A[i]]] \neq B[C[A[j]]]$, $C[A[j]] \neq C[A[i]]$ thus operations that change those values do not affect each other). The output array B is still sorted.

5. What is the running time of Quicksort when all elements of array A have the same value?

solution: If all elements of the array A have the same value the Partition operation returns $q = \lceil \frac{p-r+1}{2} \rceil$,

thus the running time $T(n) = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n)$ (running time of the Partition operation is $\Theta(n)$). And we usually omit ceilings and floors, hence the recurrence for running time will be $T(n) = 2T(\frac{n}{2}) + \Theta(n)$. We remember such form of recurrences and know that $T(n) = O(n \lg n)$ for such a recurrence. Let's prove this bound:

Assume $T(n) \leq c \cdot n \cdot \lg n$ (for n smaller than the given),

then $T(n) \leq 2 \cdot (c \cdot \frac{n}{2} \cdot \lg \frac{n}{2}) + \Theta(n) = cn \lg n - cn + \Theta(n) = cn \lg n - cn + cn = cn \lg n$

6. For the set of keys {1, 5, 10, 13, 16, 19, 20, 24, 50} draw binary search trees of heights 3,4,5,6,7

solution: see 6.jpg (the link can be found on the course's **Notice** page)

7. Is there exists a tree with more than one node which is a heap (max-heap in terms of new book) and at the same time is a binary search tree? (explain why not or give an example if it exists)

solution: Yes. example: a binary tree T (or an array A) where all nodes have the same **key** value (all elements of the array A have the same value). The heap property, $A[\text{parent}(i)] \geq A[i]$, holds for all nodes i except the root node. And the binary search tree property also holds.

8. consider the **searching problem**:

Input: A sequence of n numbers $A = \langle a_1, \dots, a_n \rangle$ and a value v

Output: An index i such that $v = A[i]$ or the special value NIL if v doesn't appear in A.

Write a pseudocode for **linear search**, which scans through the sequence,

looking for v . Give an asymptotically tight upper bound on its running time.

solution:

```
linear-search(A,v)
1. if length[A] > 0
2.   for j <- 1 to length[A]
3.     do if A[j] = v
4.         then return j
5.         else j <- j+1
6.   return NIL
```

The algorithm checks values of the array A one by one starting with the first element. If it finds element $A[k] = v$ then it returns k , otherwise it returns NIL. So the running time of the presented algorithm is at most n , $T(n) = O(n)$.

9. Find an asymptotic upper bound on the summation

$$\sum_{k=0}^n \frac{k}{2^k}$$

by bounding the terms method.

solution: let's find a ration r : $\frac{\frac{k+1}{2^{k+1}}}{\frac{k}{2^k}} = \frac{(k+1) \cdot 2^k}{k \cdot 2^{k+1}} = \frac{k+1}{2k} \leq r$,

(where $0 < r < 1$)

if $k = 1$ $\frac{k+1}{2^k} = \frac{2}{2} = 1$

if $k = 2$ $\frac{k+1}{2^k} = \frac{3}{4}$

So $\frac{k+1}{2^k} \leq \frac{3}{4}$ for all $k \geq 2$

Thus

$$\sum_{k=0}^n \frac{k}{2^k} \leq \sum_{k=0}^{\infty} \frac{k}{2^k} = 0 + \frac{1}{2} + \sum_{k=2}^{\infty} \frac{k}{2^k} \leq \frac{1}{2} + \frac{2}{4} \cdot \frac{1}{1 - \frac{3}{4}} = 2\frac{1}{2}$$

Hence

$$\sum_{k=0}^n \frac{k}{2^k} = O(1)$$