Exercises week 4 (partial solutions)

Complexity 2011-2012

A. Silva, H. Barendregt, B. Westerbaan & B. Westerbaan

Exercises marked with (†) are harder exercises. Exercises marked with a **(*)** can be handed in, we will correct them and give them back to you the week after. This week the answers should be handed in before **March 6 at 23h59 (CET time)**.

Handing in your answers: There are two options: e-mail to `alexandra@cs.ru.nl` or put your solutions in the post box of Alexandra (more detailed instructions are in the first week exercise sheet: `http://alexandrasilva.org/files/teaching/complexity2012/ex1.pdf`).

Exercise 2. (*) Consider the following algorithm which computes Fibonacci numbers

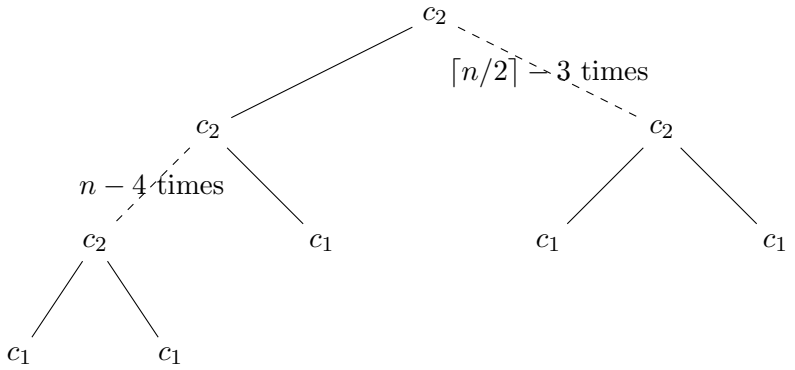
```
int fib (int n)  {
  if (n==0 || n==1) return 1;
  else return fib(n-1) + fib(n-2);
}
```

Write a recurrence relation for this algorithm. Draw the recursion tree and obtain a complexity bound from the tree. Prove your complexity bound using the substitution method.

Answer. Recurrence relation for the execution time of the algorithm

$$T(n) = \begin{cases} c_1 & n \leq 1 \\ c_2 + T(n-1) + T(n-2) & n \geq 2 \end{cases}$$

Drawing the recursion tree:



The tree is not complete and we will find a solution for the recurrence using O notation, that is, we will overestimate the execution time.

Observe that the longest path in the tree (the leftmost one) has n nodes, that is, the height of the tree is n (the shortest path, on the right, has length $\lceil n/2 \rceil$). The sum per level

is always $\leq 2^i \times c$, where c is the maximum between c_1 and c_2. Hence, we can guess that the total execution time is always less or equal than the sum of all levels: $S = \sum_{i=0}^{n-1} 2^i \times c$. How much is this? Just compute $S = 2S - S$:

$$S = 2S - S = 2 \times \sum_{i=0}^{n-1} 2^i \times c - \sum_{i=0}^{n-1} 2^i \times c = \sum_{i=1}^{n} 2^i \times c - \sum_{i=0}^{n-1} 2^i \times c = 2^n c - c$$

Hence, we can guess that $T(n) \in O(2^n)$. We prove it by induction. That is, we prove for all n that $T(n) \leq d2^n$, for some d.

First, the base cases.

$$T(0) = c_1 \leq 2^0 \times d, \text{ if } d \geq c_1$$
$$T(1) = c_1 \leq 2^1 \times d, \text{ if } d \geq c_1/2$$

Hence, for $d \geq c_1$, both inequalities hold.

Next, we assume that $T(k) \leq d2^k$, for all $k < n$ and we show that $T(n) \leq d2^n$.

$$\begin{array}{rcll} T(n) & = & c_2 + T(n-1) + T(n-2) & \text{definition } T(n) \\ & \leq & d2^{n-1} + d2^{n-2} & \text{induction hyp.} \\ & \leq & d2^{n-1} + d2^{n-1} & 2^{n-2} \leq 2^{n-1} \\ & = & 2d2^{n-1} = d2^n & \end{array}$$

Exercise 3. Consider the following function:

```
void example(int A[], int N) {
 /* N is the size of the array */
    int i;
    Node p;
    for (i = 1; i <= N; i++)
        p = insert(A,N,i,p);
    convert(p,A,1);
  }
```

Assuming $T_{\texttt{insert}} = O(\lg N)$ and $T_{\texttt{convert}} = O(N^2)$, analyze the execution time of the function `example` in the worst case.

Answer. First, we note that the `for` loop executes N times, no matter what. Then, we write $T(n)$, the execution time function for `example`:

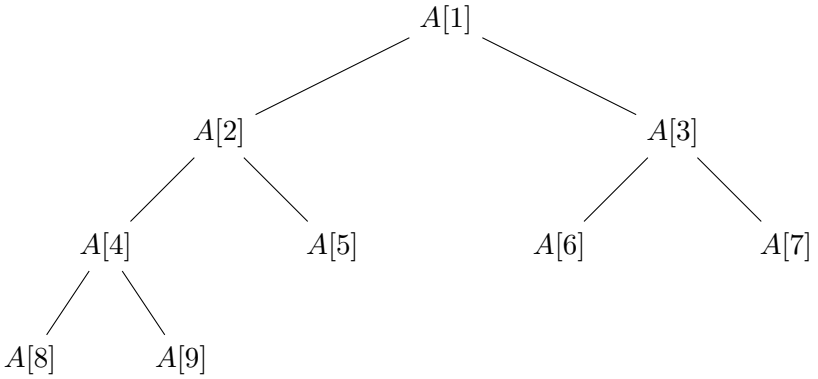
$$T(N) \leq c_1(N+1) + (c_2 \lg N)N + c_3 N^2$$

Here, the $\leq$ comes from the fact that we know $T_{\texttt{insert}} = O(\lg N)$ and $T_{\texttt{convert}} = O(N^2)$, which means $T_{\texttt{insert}} \leq c_2 \lg N$ and $T_{\texttt{convert}} \leq c_3 N^2$ for some constants c_2 and c_3.

Now we can see that $T(N) \in O(N^2)$, because N^2 is the greatest factor in $T(N)$ above, but let us show that indeed $T(N) \leq CN^2$ (for almost all N).

$$\begin{array}{rcll} T(N) & \leq & c_1(N+1) + (c_2 \lg N)N + c_3 N^2 & \\ & = & c_1 N + c_1 + (c_2 \lg N)N + c_3 N^2 & \\ & \leq & c_1 N^2 + c_1 N^2 + (c_2 N)N + c_3 N^2 & N \leq N^2, 1 \leq N^2, \lg N \leq N, \text{ for } N > 0. \\ & = & \underbrace{(2c_1 + c_2 + c_3)}_{C} N^2 & \end{array}$$

Exercise 4. (*) A *complete binary tree* is a binary tree of which all levels are completely filled, except possibly the last, on which all nodes are packed to the left. Complete binary trees can be represented as arrays in a natural way: enumerate the nodes line by line from left to right. A 9 element array A represents the following complete binary tree.



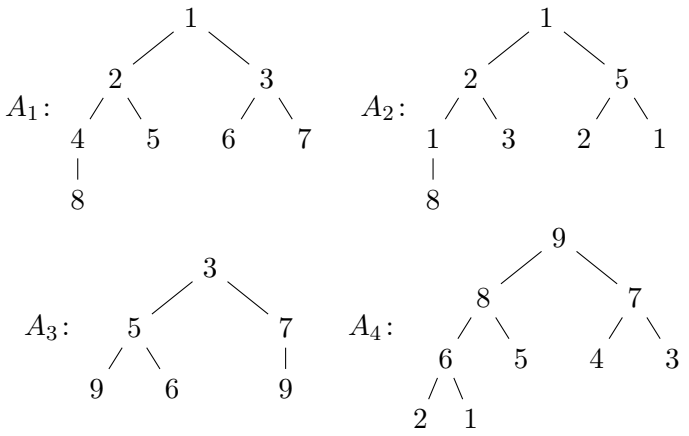
A *heap* is an array, which interpreted as complete binary tree has the following property: the value of every node is less than or equal to the values of its childnodes.

(i) Given the following arrays

$A_1 = [1,2,3,4,5,6,7,8]$ $A_2 = [1,2,5,1,3,2,1,8]$ $A_3 = [3,5,7,9,6,9]$ $A_4 = [9,8,7,6,5,4,3,2,1]$

(a) Draw the arrays A_1, A_2, A_3 and A_4 interpreted as complete binary trees.

Answer.



(b) Which ones of A_1, A_2, A_3 and A_4 are heaps?

Answer. Both A_1 and A_3 are heaps; A_2 and A_4 are not.

(ii) One application of heaps is a simple sorting algorithm.

```
void HeapSort(int A[], int N) {
 /* N is the size of the array */
 int B[N];
 for (i=0, i< N, i++)
```

```
    B[i] = A[i];  /*copy A to B */
 heapify(B);
 for (i=1, i< N, i++)
    A[i] = popmin(B);
}
```

`heapify` reorders an array such that it becomes a heap; `popmin` returns and removes the least element from the heap. Assuming that `heapify` and `popmin` run worst-case in time of order, respectively, $O(n)$ and $O(\lg n)$, prove that HeapSort runs worst-case in time of order $O(n \lg n)$.

Answer. We have, for almost all n:

$$\begin{aligned} T_{\texttt{HeapSort}}(n) &\le c_1 n + c_2 n + c_3 n \lg n \\ &\le c_1 n \lg n + c_2 n \lg n + c_3 n \lg n \\ &\le (c_1 + c_2 + c_3) n \lg n \in O(n \lg n). \end{aligned}$$

Exercise 6. (†) Let $T\colon \mathbb{N} \to \mathbb{N}$ be monotonic increasing. Show the following implications; the assumption is that the inequalities hold for all even numbers n.

$$\begin{array}{lrcll} (1) & T(n) & \le & T(n/2) + 12 & \Rightarrow \quad T \in O(\lg n). \\ (2) & T(n) & \le & 2T(n/2) + 7 & \Rightarrow \quad T \in O(n). \\ (3) & T(n) & \le & 2T(n/2) + 8n & \Rightarrow \quad T \in O(n \lg n). \end{array}$$

(i) For $m \ge 2$

$$\begin{aligned} T(2^m) &\le T(2^{m-1}) + 12 && \text{applying assumption} \\ &\le T(2) + 12(m-1) && \text{applying assumption } m-1 \text{ times} \\ &= \left(\frac{T(2)}{m-1} + 12 \right) (m-1) \\ &\le (T(2) + 12) m \\ &= (T(2) + 12) \lg 2^m. \end{aligned}$$

Given any $n > 2$, there exist $m, r \in \mathbb{N}$ such that

$$2^{m+1} \ge n, \quad m \ge 1 \quad \text{and} \quad n = 2^m + r.$$

And thus

$$\begin{aligned} T(n) &= T(2^m + r) && \text{by definition of } m \text{ and } r \\ &\le T(2^{m+1}) && \text{by monotonicity} \\ &\le (T(2) + 12) \lg 2^{m+1} && \text{by the previous} \\ &= (T(2) + 12)(1 + \lg 2^m) \\ &\le (T(2) + 12)(1 + \lg n) \\ &\le (2T(2) + 24) \lg n, \end{aligned}$$

which shows $T \in O(\lg n)$.

(ii) For $m \geq 2$

$$\begin{aligned} T(2^m) &\leq 2T(2^{m-1}) + 7 && \text{applying assumption} \\ &\leq T(2)2^{m-1} + 7(2^0 + 2^1 + \ldots + 2^{m-2}) && \ldots m-1 \text{ times} \\ &= T(2)2^{m-1} + 7(2^{m-1} - 1) \\ &\leq (T(2) + 7)2^{m-1} \\ &\leq (T(2) + 7)2^m. \end{aligned}$$

Given any $n > 2$, there exist $m, r \in \mathbb{N}$ such that

$$2^{m+1} \geq n, \quad m \geq 1 \quad \text{and} \quad n = 2^m + r.$$

And thus

$$\begin{aligned} T(n) &= T(2^m + r) \\ &\leq T(2^{m+1}) \\ &\leq (T(2) + 7)2^{m+1} \\ &= (2T(2) + 14)2^m \\ &\leq (2T(2) + 14)n, \end{aligned}$$

which shows $T \in O(n)$.

(iii) For $m \geq 2$

$$\begin{aligned} T(2^m) &\leq 2T(2^{m-1}) + 82^m && \text{applying assumption} \\ &\leq T(2)2^{m-1} + 8(m-1)2^m && \ldots m-1 \text{ times} \\ &\leq T(2)2^m + 8m2^m \\ &= T(2)2^m + 82^m \lg 2^m \\ &\leq (T(2) + 8)2^m \lg 2^m. \end{aligned}$$

Given any $n > 2$, there exist $m, r \in \mathbb{N}$ such that

$$2^{m+1} \geq n, \quad m \geq 1 \quad \text{and} \quad n = 2^m + r.$$

And thus

$$\begin{aligned} T(n) &= T(2^m + r) \\ &\leq T(2^{m+1}) \\ &\leq (T(2) + 8)2^{m+1} \lg 2^{m+1} \\ &= (2T(2) + 16)2^m \lg 2^{m+1} \\ &= (2T(2) + 16)2^m (\lg 2^m + 1) \\ &\leq (4T(2) + 32)2^m \lg 2^m \\ &\leq (4T(2) + 32)n \lg n, \end{aligned}$$

which shows $T \in O(n \lg n)$.