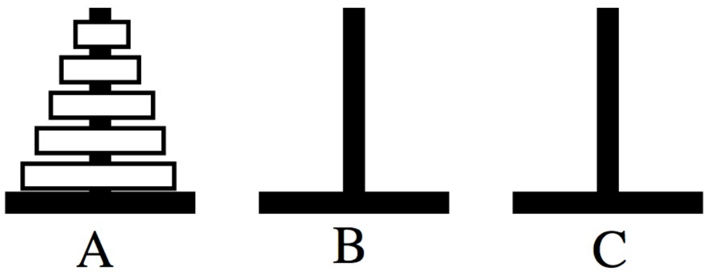
Test 2 (draft solutions)

Complexity 2011-2012

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

Note: The students who handed in homework assignments (and got the points) can skip Exercise 4.

Exercise 1. (2 points)



The Towers of Hanoi is an ancient puzzle, originating with Hindu priests in the great temple of Benares. Suppose we are given three towers, or more humbly, three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape (see image above). The objective of the puzzle is to move the entire stack to another rod (say from A to B), obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

If there are three discs in A the puzzle can be solved as follows: move A to B ; move A to C; move B to C; move A to B; move C to A; move C to B; move A to B. Hence, in 7 steps.

Below is a recursive procedure/algorithm that solves the above puzzle. In pseudocode:

```
function F($n, s, t$)
    if $n = 1$ then
        Move $s$ to $t$
    else
        $\{u\} \leftarrow \{A, B, C\} - \{s, t\}$
        $F(n-1, s, u)$
        Move $s$ to $t$
        $F(n-1, u, t)$
```

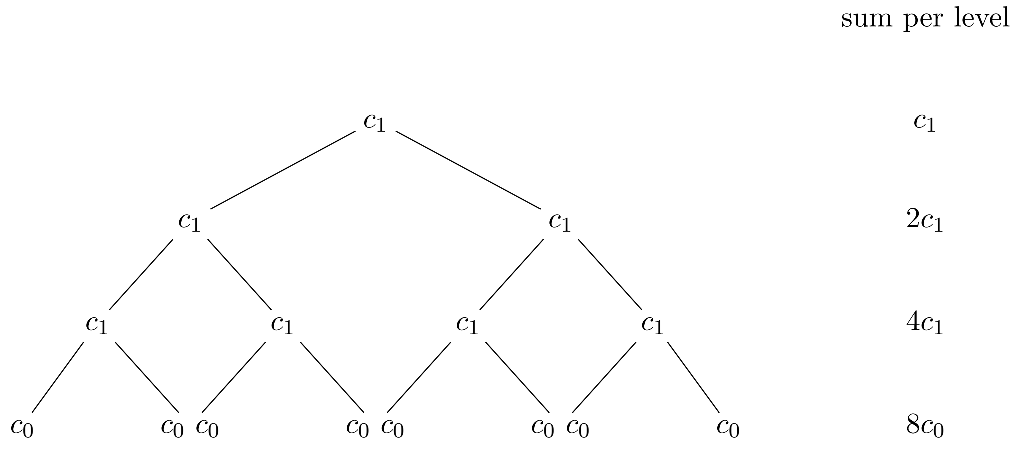
(i) **(1 point)** Write a recurrence relation T for the execution time of the algorithm in the worst case.

Answer.

$$T(n) = \begin{cases} c_0 & n = 1 \\ c_1 + 2T(n-1) & \text{otherwise} \end{cases}$$

(ii) **(1 point)** Write the recursion tree to move 4 discs from A to B. Guess to which O class the algorithm belongs.

Answer.



By observing the pattern of the sum per level, one can guess that for n the cost will roughly be

$$\sum_{i=0}^{n-1} 2^i \times c, \text{ where } c = \max\{c_1, c_0\}.$$

Then observe that $S = \sum_{i=0}^{n-1} 2^i$ satifies:

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

Hence, $T(n) \in O(2^n)$.

Exercise 2. (3 points) Consider the following recursive implementation of the algorithm *maxsort*:

```
void maxsort (int v[], int n){ % n is the size of the array
       int i;
       if (n > 1) {
         i = max(v, n);
         swap(v, i, n);
         maxsort(v, n-1);
     }
}
```

The function `swap` swaps the values in two positions of the array in constant time, whereas the function `max` finds the maximum element in an array in linear time.

(i) **(1 point)** Write a recurrence relation T for the execution time of the algorithm in the worst case.

Answer.

$$T(n) \leq \begin{cases} c_0 & n < 1 \\ c_1 n + c_2 + T(n-1) & \text{otherwise} \end{cases}$$

(ii) **(2 points)** Find the solution for the recurrence and prove it is a solution, using O-notation (by induction, using the substitution method).

Answer. To guess the solution observe that, for $n > 1$, we have

$$\begin{array}{rcl} T(n) & \leq & c_1 n + c_2 + c_1(n-1) + c_2 + \cdots + c_1 + c_2 + c_0 \\ & = & c_1(n + (n-1) + (n-2) + \cdots + 1) + nc_2 + c_0 \\ & = & c_1 \frac{n^2+n}{2} + nc_2 + c_0 \in O(n^2) \end{array}$$

We now prove, by induction, that $T(n) \in O(n^2)$, that is, there exist a constant c and $n_0 \in \mathbb{N}$ such that $T(n) \leq cn^2$, for all $n \geq n_0$.

Base case is trivial. For $n > 1$, we have that:

$$\begin{array}{rcll} T(n) & \leq & c_1 n + c_2 + T(n-1) & \text{definition} \\ & \leq & c_1 n + c_2 + c(n-1)^2 & \text{induction} \\ & = & c_1 n + c_2 + cn^2 - 2nc + 1 & \\ & \leq & cn^2 & \text{if } (*) \end{array}$$

$$(*)\ c_1 n + c_2 - 2nc + 1 \leq 0 \Leftrightarrow n \geq \frac{c_2 + 1}{2c - c_1}$$

Exercise 3. (2 points) A given set of *distinct* numbers may be sorted by first building a binary search tree containing those numbers (inserting the numbers one by one), and then printing the numbers during a traversal. In what follows, give your answer using O notation.

(i) **(1 point)** What is the worst case running time for this sorting algorithm? Explain your answer.

Answer. In the worst case, the array is reverse sorted and the binary search tree built has only one long branch of size n. Inserting to such a "tree" of k nodes has cost k, so building the tree will cost:

$$1 + 2 + ... + (n-1)$$

which is $O(n^2)$. Traversing/printing will have to go through every node, having cost $O(n)$. So the total cost is $O(n^2)$.

(ii) **(1 point)** What is the best case running time for this sorting algorithm? Explain your answer.

Answer. In the best case, the tree will be balanced. Insertion in a balanced tree of k nodes has cost $\lg(k)$, so building the tree will cost:

$$\lg(1) + \lg(2) + \cdots + \lg(n-1) = \log((n-1)!)$$

which is $O(n \lg n)$.

Traversing/printing will again have to go through every node, having cost $O(n)$. So the total cost is $O(n \lg n)$.

Exercise 4. (3 points) Give an algorithm with running time $O(nk \lg k)$ which merges k sorted lists with a total of n elements producing a sorted list (of length kn). [Hint: Use a heap to improve on the obvious algorithm of order $O(nk^2)$].

Answer. One solution is to store a heap of the minimum elements of each of the lists (each node of the heap containing the minimum and the indication from which list the number came from) . At each step, output the minimum of the heap and insert the next element from the list where the minimum came from into the heap.