Submission instructions as in previous homeworks.

1 (100 pts.) Bogi sort.

Consider the following exciting sorting algorithm. For simplicity we will assume that $n$ is always some positive power of 2 (i.e. $n = 2^i$, for some positive integer $i > 0$).

```plaintext
bogiSort(A[0..n−1]) :
    if $n \leq 16$
        InsertionSort(A[0..n−1])
    else /* $n > 16$ */
        for $i$ ← 0 to 2 do
            for $j$ ← 2 down to $i$
                bogiSort(A[jn/4..(j+2)n/4−1])
```

1.A. (25 pts.) Prove that `bogiSort` actually sorts its input. (You can assume that all the numbers in the array $A$ are distinct.)

1.B. (25 pts.) State a recurrence (including the base case(s)) for the number of comparisons executed by `bogiSort`.

1.C. (25 pts.) Solve the recurrence, and prove that your solution is correct. (Your proof should be self contained and not use off the shelf tools like the master theorem [puke]).

1.D. (25 pts.) Show that the number of `swaps` executed by `bogiSort` is at most $\left(\frac{n}{2}\right)$. 

2 (100 pts.) Pick it up.

You are given an array $A$ with $n$ distinct numbers in it, and another array $B$ of ranks $i_1 < i_2 < \ldots < i_k$. An element $x$ of $A$ has rank $u$ if there are exactly $u − 1$ numbers in $A$ smaller than it. Design an algorithm that outputs the $k$ elements in $A$ that have the ranks $i_1, i_2, \ldots, i_k$.

2.A. (20 pts.) As a warm-up exercise describe how to solve this problem in $O(nk)$ time.

2.B. (60 pts.) Describe a $O(n \log k)$ recursive algorithm for this problem. Prove the bound on the running time of the algorithm.

2.C. (20 pts.) Show, that if this problem can be solved in $T(n, k)$ time, then one can sort $n$ numbers in $O(n + T(n, n))$ time (i.e., give a reduction). Provide a strong intuitive reason why the above problem can not be solved in time faster than $O(n \log k)$.

3 (100 pts.) Is good???

You are given an array $A$ of $n$ numbers (not necessarily sorted). You are given a function `isGood(x)`, which can tell you for a number $x$ if is good or not. A number $x$ is `good` if it is at most some unknown value $\alpha$ (i.e., $x \leq \alpha$). It is bad if $x > \alpha$. Think about calling `isGood` as being an expensive operation, that your algorithm should perform as little as possible.
3.A. (20 pts.) (Easy.) Show how to compute all the numbers of \( A \) that are good using \( O(\log n) \) calls to \texttt{isGood}. What is the running time of your algorithm. (Here, the solution should be short and simpler than what follows. (Here, short means a few lines.))

3.B. (40 pts.) Show how to find all the elements of \( A \) that are good using \( O(\log n) \) calls to \texttt{isGood} and with total running time \( O(n) \). (The solution here should be simpler than the algorithm in (C)).

3.C. (40 pts.) \texttt{isGood} turns out to be better than good! Given a set \( Y \) of numbers, where \( |Y| \leq k \), the generalized \( \texttt{isGood}(Y) \), returns to you (in a single call) for each number of \( Y \) whether it is good or not. As a function of \( n \) and \( k \) describe an algorithm that (asymptotically) performs the minimal number of calls to the improved \texttt{isGood}. For full credit your algorithm should be as fast as possible. What is the running time of your algorithm? State the recurrences you used to derive your bounds. (Hint: Look on other problems in this homework.)