CS 4820, Spring 2017 **Homework 2, Problem 1**

Name: Yuxiang Peng NetID: yp344 Collaborators: jl3455, zl542

(1) *(10 points)*
Solve Chapter 4, Exercise 14 in Kleinberg & Tardos.

(Solution: a)
Algorithm 1:
This problem is similar to the interval scheduling problem discussed in class. We apply the same "Earlest Finish Time" approach to solve this problem. The pseudocode is as follows:

```
while the set of intervals not empty {
select the interval i with minimum f(i)
add it to the set of invoking time for status_check
let $S_i$={ the set of intervals that conflict with i}
delete all elements of $S_i$
} The resulting $S_i$ is the time that we need to invoke status_check.
```

The algorithm only terminates after all the processes have been checked.

The runtime of this algorithm is similar to that analyzed in the textbook. First we sort the n process in the order of finish time and this takes $O(nlogn)$ time. Then we contruct the array $S[1, 2...n]$ which indicates the start time of all processes, and this takes $O(n)$.We select the processes in increasing order, and after selecting each f, we select the first j that satisfies s(j)¿f. For each interval it is constant time, and this part takes $O(n)$

Algorithm 2: (not sure if optimal)
First we discretize the time interval. The reason of discretizing is that as each invocation of status_check is modeled to last only this single point in time, thereare infinite number of moments that we can invoke the program status_check. Therefore, we have to discretize the time interval to generate a finite number of candidat moments. The step, which is the length between two consecutive candidate moments, has to be smaller than the minimum overlapping time of all the sensive processes. In this way we can guarantee that the discretization process does not rule out any candidate moment on any intervals.

The proposed algorithm is of greedy type. Every time it picks the candidate moment that intersects with most sensitive prosesses. The pseudcode is as follows.

```
set of all processes that are not checked U
set of invocation moments of status_check T = ø

while U is not empty{
find the moment m that intersects with most sensitive processes
add m to T
delete all processes that intersects with m from U
```

}

(Solution: b)
The statement that we need exactly k* times of invokation is correct.
The discussion here is based on Algorithm 1 in problem a.
The proof of Algorithm 1 is similar to the proof for "Earlest Finish Time".
First, every interval belongs to one of the sets S_i. Second, every element of S_i contains f(i), as it ooverlaps with i and cannot finish earlier. Third, any set of non-conflicting intervals contain at most 1 element from S_i to avoid overlap. This algorithm picks 1 from each S_I, therefore, the number of invocation is optimal.

Note also that the number of invocation is at least k*. Therefore, the optimal number is k*. On the other hand, we can calculate k* using "Earlest Finish Time" algorithm discussed in class, and would yield the same result for the number of invocations.