

CS 535 Homework 7

Due: April 15 (Th), in class.

Undergrads, please answer questions 1, 2, and 4; question 3 is a bonus question. Grads, please answer questions 2, 3, and 4; question 1 is a bonus question.

1. *Finding a counterexample.* Recall the task scheduling problem presented in class (which can also be found in Sec. 5.1.2). Let T be the set of tasks where task i starts at time s_i and finishes at time f_i . The goal is to schedule all tasks in T on the fewest number of machines. Suppose we modify the algorithm as follows.

NewTaskSchedule(T)

```
 $m \leftarrow 0$ 
for  $i = 1$  to  $n$ 
  if there is a machine  $j$  that is free
    schedule task  $i$  on machine  $j$ 
  else
     $m \leftarrow m + 1$ 
    schedule task  $i$  on machine  $m$ .
```

Thus, in the original TaskSchedule algorithm, a task with an earlier start time is always scheduled ahead of a task with a later start time; in NewTaskSchedule this property may not hold anymore. Find an example where NewTaskSchedule fails to use the optimal number of machines. Why does the proof of correctness for TaskSchedule not apply to NewTaskSchedule anymore?

Note: Since the tasks in T are not sorted, you may encounter a task whose start time is earlier than the ones you've processed. You are allowed to schedule this task in a machine as long as it does not overlap with the tasks already scheduled in the machine.

2. C-5.2
3. C-5.5
4. An array $A[1 \dots n]$ has a *majority element* if more than half of its entries are the same. Given such an array A , our goal is to design an efficient algorithm that will determine if A has a majority element, and if so, to find that element.
 - a. Suppose we simply use the following brute force method: For $i = 1$ to n , compute the number of times $A[i]$ appears in A . If the number is greater than $n/2$, return the number. Otherwise, once the for loop has ended and no number has been returned, return that A has no majority element. What is the runtime of this algorithm?
 - b. As always, we want to do better than brute force. Suppose we take a divide-and-conquer approach. Split A into two smaller arrays A_1 and A_2 whose sizes

are half that of A . Determine, if any exists, the majority elements in A_1 and A_2 .

(i) How does the majority element of A (if it exists) relate to the majority elements of A_1 and A_2 (if they exist)?

(ii) Given your answer in (i), create a divide-and-conquer algorithm for our problem. Why should the runtime be $O(n \log n)$ in the worst-case?