

Due: By 6:00pm on Wednesday 26 January, in the CSC 373H drop box in room BA 2220. **Worth:** 7.5%

1. [15 marks]

Consider a variant on the problem of Interval Scheduling where instead of wanting to schedule as many intervals (“jobs”) as we can on one processor, we now want to schedule all of the jobs on as few processors as possible.

As before, the input is $(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)$ where $n \geq 1$ and all $s_i < f_i$ are nonnegative integers.

A schedule specifies for each job i a positive integer $P(i)$ (the “processor number” for job i). It must be the case that if $i \neq j$ and $P(i) = P(j)$, then jobs i and j do not overlap. We wish to find a schedule that uses as few processors as possible, *i.e.*, such that $\max\{P(1), P(2), \dots, P(n)\}$ is minimal.

Consider the following greedy algorithm: Sort the jobs by nondecreasing start time, then for each job, schedule it on some existing processor if that is possible (note that if more than one processor can be used, the algorithm doesn’t specify which one to choose); if no processor is free, create a new processor and schedule the job on the new processor. The algorithm uses variable m to store the number of processors currently being used. For each processor j , $E[j]$ is the last finish time of the jobs currently scheduled on processor j .

```

sort jobs so that  $s_1 \leq s_2 \leq \dots \leq s_n$ 
 $m := 1$ 
 $P[1] := 1$ 
 $E[1] := f_1$ 
for  $i := 2$  to  $n$ :
    if there is a processor number  $j$  such that  $1 \leq j \leq m$  and  $E[j] \leq s_i$ :
         $j :=$  some processor number such that  $E[j] \leq s_i$ 
         $P[i] := j$ 
         $E[j] := f_i$ 
    else:
         $m := m + 1$ 
         $P[i] := m$ 
         $E[m] := f_i$ 

```

(a) [10 marks]

Prove that this algorithm is guaranteed to compute a schedule that uses the minimum number of processors.

(b) [5 marks]

Briefly describe an efficient implementation of the algorithm, making it clear what data structures you are using. Express the running time of your implementation as a function of n (the number of jobs), using appropriate asymptotic notation.

2. [10 marks]

Here is another variant on the problem of Interval Scheduling.

Suppose we now have *two* processors, and we want to schedule as many jobs as we can. As before, the input is $(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)$ where $n \geq 1$ and all $s_i < f_i$ are nonnegative integers.

A schedule is now defined as a pair of sets (A_1, A_2) , the intuition being that A_i is the set of jobs scheduled on processor i . A schedule must satisfy the obvious constraints: $A_1 \subseteq \{1, 2, \dots, n\}$,

$A_2 \subseteq \{1, 2, \dots, n\}$, $A_1 \cap A_2 = \emptyset$, and for all $i \neq j$ such that $i, j \in A_1$ or $i, j \in A_2$, jobs i and j do not overlap.

A natural greedy algorithm for this variation is as follows. First, sort the jobs in order of nondecreasing finish times. Then, handle the jobs one at a time, scheduling each job if possible. If a job can be scheduled on either processor, pick the processor where it will cause the smallest “gap”. This algorithm is described in detail below, where variables E_1 and E_2 are used to keep track of the largest finish time of jobs in A_1 and A_2 , respectively.

```

sort jobs so that  $f_1 \leq f_2 \leq \dots \leq f_n$ 
 $A_1 = \emptyset$ 
 $A_2 = \emptyset$ 
 $E_1 := 0$ 
 $E_2 := 0$ 
for  $i := 1$  to  $n$ :
    if  $E_2 \leq E_1 \leq s_i$  or  $E_1 \leq s < E_2$ :
         $A_1 := A_1 \cup \{s_i\}$ 
         $E_1 := f_i$ 
    else if  $E_2 \leq s_i$ :
         $A_2 := A_2 \cup \{s_i\}$ 
         $E_2 := f_i$ 

```

Prove that this algorithm works, that is, it is guaranteed to produce an optimal schedule.

3. [10 marks]

Consider the following variant of the Minimum Spanning Tree problem.

We are given a connected, undirected graph $G = (V, E)$ with a positive cost $c(e)$ for every edge $e \in E$. Say we are also given one specific edge $e_0 = \{u_0, v_0\} \in E$ that must be included in the spanning tree.

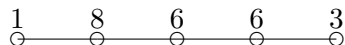
We wish to compute a spanning tree of G that contains edge e_0 and has smallest total cost from amongst all such trees.

Give a variant of Kruskal’s algorithm for this problem, and prove that it works.

4. [15 marks]

Let $G = (V, E)$ be an undirected graph with n nodes. A subset of the nodes is called an *independent set* if no two nodes in the subset are joined by an edge. Finding large independent sets is difficult in general, but there are simple cases where it can be done efficiently.

A graph $G = (V, E)$ is called a *path* if its nodes can be listed as $V = v_1, v_2, \dots, v_n$ such that $E = \{(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)\}$. A *weighted path* is a path in which each node v_i is given a positive integer weight w_i , like the example pictured below (where the weights are written above each node).



The goal in this question is to solve the following problem: given a weighted path G , find an independent set with maximum total weight.

(a) [3 marks]

Give an example to show that the following algorithm does not always find an independent set of maximum total weight.

```
 $S := \emptyset$   
while  $G$  is not empty:  
    pick  $v_i$  in  $G$  with maximum weight  $w_i$   
    remove  $v_i$  and its neighbours from  $G$   
     $S := S \cup \{v_i\}$   
return  $S$ 
```

(b) [2 marks]

Give an example to show that the following algorithm does not always find an independent set of maximum total weight.

```
let  $S_1$  be the set of all  $v_i$  where  $i$  is odd  
let  $S_2$  be the set of all  $v_i$  where  $i$  is even  
return  $S_1$  or  $S_2$ , whichever has largest total weight
```

(c) [10 marks]

Give an algorithm that takes an n -node weighted path G and returns an independent set of maximum total weight. Justify briefly that the running time of your algorithm is polynomial in n , independent of the values of the weights.