4.1 Interval Scheduling:

The Greedy Algorithm Stays Ahead Let's recall the Interval Scheduling Problem, which was the first of the five representative problems we considered in Chapter 1. We have a set of requests {1, 2, …, n}; the i th request corresponds to an interval of time starting at s(i) and finishing at f(i). (Note that we are slightly changing the notation from Section 1.2, where we used si rather than s(i) and f i rather than f(i). This change of notation will make things easier to talk about in the proofs.) We'll say that a subset of the requests is compatible if no two of them overlap in time, and our goal is to accept as large a compatible subset as possible. Compatible sets of maximum size will be called optimal. Designing a Greedy Algorithm

Using the Interval Scheduling Problem, we can make our discussion of

greedy algorithms much more concrete. The basic idea in a greedy

algorithm for interval scheduling is to use a simple rule to select a first

request i1

. Once a request i1

is accepted, we reject all requests that are not

compatible with i1

. We then select the next request i2

to be accepted, and

again reject all requests that are not compatible with i2

. We continue in this

fashion until we run out of requests. The challenge in designing a good

greedy algorithm is in deciding which simple rule to use for the selection—

and there are many natural rules for this problem that do not give good

solutions

Analyzing the Algorithm

While this greedy method is quite natural, it is certainly not obvious that it

returns an optimal set of intervals. Indeed, it would only be sensible to

reserve judgment on its optimality: the ideas that led to the previous

nonoptimal versions of the greedy method also seemed promising at first.

As a start, we can immediately declare that the intervals in the set A

returned by the algorithm are all compatible

Implementation and Running Time

We can make our algorithm run in time O(n log n) as follows. We begin by sorting the n requests in order of finishing time and labeling them in this order; that is, we will assume that f(i) ≤ f(j) when i < j. This takes time O(n log n). In an additional O(n) time, we construct an array S[1 … n] with the property that S[i] contains the value s(i). We now select requests by processing the intervals in order of increasing f(i). We always select the first interval; we then iterate through the intervals in order until reaching the first interval j for which s(j) ≥ f(1); we then select this one as well. More generally, if the most recent interval we've selected ends at time f, we continue iterating through subsequent intervals until we reach the first j for which s(j) ≥ f. In this way, we implement the greedy algorithm analyzed above in one pass through the intervals, spending constant time per interval. Thus this part of the algorithm takes time O(n).