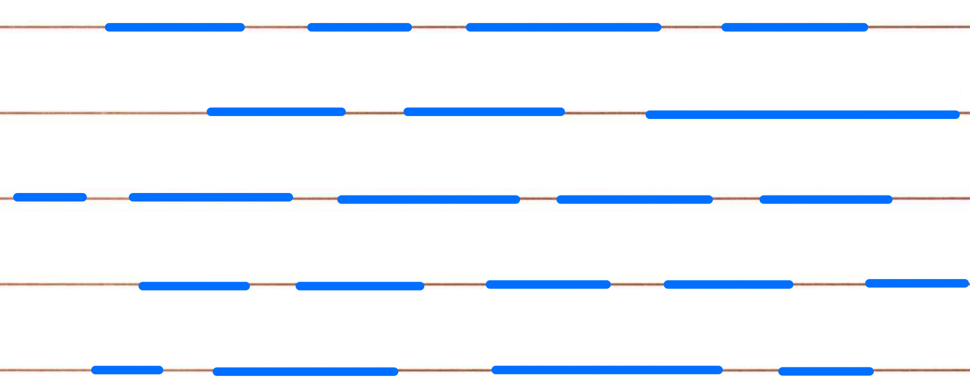
Greedy

Interval Scheduling Problem

Input: A set of requests $\{1..n\}$, where the i^{th} request starts at $s(i)$ and ends at $f(i)$.

Output: A largest compatible subset of these requests.



High Level Solution

Initially R is the complete set of requests and A is empty
While R is not empty
    Choose a request $i \in R$ that has the smallest finish time
    Add request i to A
    Delete all requests from R that are not compatible with i
Endwhile
Return A

Proof of Correctness

1_ Prove that A is a compatible set

2_ Prove that A is an optimal set

Easy to show #1: Since we always delete all overlapping requests before choosing the next request, we can never end up with overlapping requests in A.

#2: Say A is of size k, and
suppose there is an optimal solution O.

Label requests in A: $i_1, i_2, \ldots i_k$
" " " O: $j_1, j_2, \ldots, j_m$

We will first prove that for all indices $1 \leq r \leq k$, we have $f(i_r) \leq f(j_r)$

Implementation

Sort requests in order of finish time and label in this order:

$$f(i) \leq f(j) \text{ where } i < j$$

Select requests in order of increasing $f(i)$, always selecting the first request.
Then iterate through the intervals in this order until reaching the first interval j where $s(j) \geq f(i)$ and then pick j.

Scheduling to Minimize Lateness

Input: A set of requests $\{1..n\}$, where the i^{th} request has deadline d_i (and time duration t_i.

Output: A schedule of the requests that minimizes the maximum lateness. Where the lateness for request i is $l_i = f(i) - d_i$

Based on the above objective, which of the following solutions is preferred?

Sol. 1	job 1	late by 5 hrs
	job 2	late by 6 hrs
Sol. 2	job 1	late by 0 hrs
	job 2	late by 7 hrs

Proof of Correctness:

1- There is an optimal solution with no gaps.

2- Jobs with identical deadlines can be scheduled in any order without affecting maximum lateness.

3- Def. Schedule A' has an inversion if a job i with deadline d_i is scheduled before job j with an earlier deadline.

4- All schedules with no inversions and no idle Time have the same maximum lateness

5- There are an optimal schedule that has no inversions and no idle time.

6- Proved that there exists an optimal schedule with no inversions and no idle time.

Also proved that all schedules with no inversions and no idle time have the same maximum lateness.

Our greedy algorithm produces one such solution, therefore our solution is also optimal.

Discussion 3

1. Let's consider a long, quiet country road with houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. Further, let's suppose that, despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations.
Give an efficient algorithm that achieves this goal and uses as few base stations as possible. Prove that your algorithm correctly minimizes the number of base stations.

2. Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of campers. One of his plans is the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming, and so on.
Each contestant has a projected *swimming time*, a projected *biking time*, and a projected *running time*. Your friend wants to decide on a *schedule* for the triathlon: an order in which to sequence the starts of the contestants. Let's say that the *completion time* of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathlon, assuming the time projections are accurate.
What is the best order for sending people out, if one wants the whole competition to be over as soon as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible. Prove that your algorithm achieves this.

3. The values 1, 2, 3, . . . , 63 are all inserted (in any order) into an initially empty min-heap. What is the smallest number that could be a leaf node?

4. Given an unsorted array of size *n*. Devise a heap-based algorithm that finds the *k*-th largest element in the array. What is its runtime complexity?

5. Suppose you have two min-heaps, A and B, with a total of *n* elements between them. You want to discover if A and B have a key in common. Give a solution to this problem that takes time O(n log n) and explain why it is correct. Give a brief explanation for why your algorithm has the required running time. For this problem, do not use the fact that heaps are implemented as arrays; treat them as abstract data types.

1. Let's consider a long, quiet country road with houses scattered very sparsely along it. We can picture the road as a long line segment, with an eastern endpoint and a western endpoint. Further, let's suppose that, despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations.
Give an efficient algorithm that achieves this goal and uses as few base stations as possible. Prove that your algorithm correctly minimizes the number of base stations.

2. Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of campers. One of his plans is the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he or she is out and starts biking, a third contestant begins swimming, and so on.
Each contestant has a projected *swimming time*, a projected *biking time*, and a projected *running time*. Your friend wants to decide on a *schedule* for the triathlon: an order in which to sequence the starts of the contestants. Let's say that the *completion time* of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathlon, assuming the time projections are accurate.
What is the best order for sending people out, if one wants the whole competition to be over as soon as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible. Prove that your algorithm achieves this.