Let $A_i$ be assignments handed out (available) on $a_i$ and due on $d_i,$ with
$a = \min_i a_i, d = \max_i d_i.$ Let a schedule $D$ specify, for each day $t \in [a, d],$ either
$i,$ an assignment with $d_i \leq t$ or NIL, another intellectually stimulating activity.
Our goal is to find a schedule allowing us to complete as many of the $A_i$ as possible.

**Correctness**

**Strategy:** On each day, choose the available assignment with the earliest due date.

**Proof:** (by modify-the-solution)

*Modify-the-solution:* Let $d$ be the “decision point” chosen by the greedy algorithm and $o$ be the greedy option. Let $S'$ be a solution that does not choose $o.$ We must find a solution $S$ that does choose $o$ for $d,$ and that is at least as good as $S'.$ Using this fact, we can prove by induction that a non-greedy solution can be turned into an equally-good greedy one, one decision at a time.

Let $a$ be the first availability date, $i$ be the assignment available on $a$ with earliest due date $d_i.$ Let $S'$ be a solution not choosing $i.$ We must find an $S$ choosing $i$ completing at least as many solution as $S'$ (i.e. $|S| \geq |S'|$). We will find $S$ by dividing the possible situations into cases, and analyzing each case.

If $S'$ is not doing $i$ on day $t,$ what could it be doing (what cases are there)? Either nothing, or some other assignment $j$:

1. $S'$ is doing nothing. Then $S$ schedules $i$ on $t,$ and if $S'$ does $i$ on $t',$ then $S$ does nothing on $t'.$ Hence the greedy solution $S,$ which is clearly legal, is at least as large as $S'.$

2. $S'$ is doing $j \neq i.$ Here we consider two subcases, based on whether or not $S'$ performs $i.$

   (a) $S'$ does not perform $i.$ Then if $S$ replaces $j$ with $i$ on $t$ and does not perform $j,$ $|S| = |S'|.$
(b) $S'$ performs $i$ on some later date $t' > t$. Then let $S$ be the same as $S'$, except $S(t) = i, S(t') = j$. Clearly $|S| = |S'|$, since they complete the same assignments. However, we must also show that $S$ is legal, i.e. that $j$ is possible on $t'$. To do this, we show $a_j \leq t'$ and $t' \leq d_j$. Since $i$ is the greedy choice, $d_i \leq d_j$. Since $S'$ schedules $i$ on $t'$, $t' \leq d_i \leq d_j$. Since $S'$ schedules $j$ on $t$, and schedules $i$ on $t'$, $a_j \leq t \leq t'$. Therefore $a_j \leq t' \leq d_j$, and $S$ is legal.

Example

Let’s look at how this “mutation plus induction” process works on a real problem:

\[\begin{array}{cccc}
a & 1 & 4 \\
b & 2 & 3 \\
c & 1 & 5 \\
d & 1 & 4 \\
e & 1 & 3 \\
f & 3 & 5 \\
\end{array}\]

The greedy solution is: Another solution is:

\[\begin{array}{ccc}
1 & e & e \\
2 & b & b \\
3 & a & a \\
4 & d & d \\
5 & c & c \\
\end{array}\]

The modification procedure works as

\[\begin{array}{ccccccc}
1 & a & e & e & e & e & e \\
2 & b & b & b & b & b & b \\
3 & f & f & a & a & a & a \\
4 & c & c & c & d & d & d \\
5 & nil & nil & nil & nil & c \\
\end{array}\]

Implementation

Structure: A set $A$ of unfinished assignments that are not yet due.

Access: Find the first-due element in $A$.

Update: (1) Add all assignments that become available at $t$. (2) Remove a completed assignment. (3) Remove all overdue assignments.
This suggests an implementation using a min-heap:

```plaintext
1  H <- empty min-heap
2  A <- assignments, sorted by assigned(i)
3  Do <- schedule array
4  i <- 0
5  for t = min(assigned(i)) to max(due(i))
6      while min(H) <= t - 1
7          delete_min(H)
8      if empty(H)
9          Do[t] <- NIL
10     else
11        Do[t] <- min(H)
12        delete_min(H)
13        while assigned(A[i]) = t
14            insert(H, (due(A[i]), i))
15        i <- i + 1
```

Line 2 takes $O(n \log n)$. Line 14 takes $O(\log n)$ and, since each assignment is inserted at most once, executes $n$ times. Likewise, lines 6-12 perform at most one `delete_min` taking $O(\log n)$ for each element. The loop at line 5-15 executes $T = \max_i d_i - \min_i a_i$ times, so the total running time is $O(T + n \log n)$.

Note: if there are very few assignments to be scheduled over a long time period, then the $T$ term dominates this algorithm. How would you modify the algorithm, and the data structures it uses, to make it run in $O(n \log n)$ for any value of $T$?