ALGORITHM: Greedy Change-Making Algorithm.
\[ c_1 > c_2 > \ldots > c_r \]  // values of denominations of coins,
\[ \text{int } n: \]  // a positive integer

for \( i := 1 \) to \( r \)
\{ 
  \[ d_i = 0 \]  // \( d_i \) counts the coins of denomination \( c_i \) used
  \{ 
    \[ d_i := d_i + 1 \]  // add a coin of denomination \( c_i \)
    \[ n := n - c_i \]  // \( d_i \) is the number of coins of denomination \( c_i \) in the change
  \}
\}

Problem:
Suppose we have a group of proposed talks with preset start and end times. Devise a greedy algorithm to schedule as many of these talks as possible in a lecture hall, under the assumptions that once a talk starts, it continues until it ends, no two talks can proceed at the same time, and a talk can begin at the same time another one ends. Assume that talk \( j \) begins at time \( s_j \) (where \( s \) stands for start) and ends at time \( e_j \) (where \( e \) stands for end).

Solution: In each step select the talk with the earliest ending time among the talks compatible with those already selected.

ALGORITHM: Greedy Algorithm for Scheduling Talks.
Procedure \( \text{schedule}(s_1 \leq s_2 \leq \ldots \leq s_n: \text{start times of talks}, e_1 \leq e_2 \leq \ldots \leq e_n: \text{ending times of talks}) \)
\{ 
  // you can sort talks by finish time and reorder so that \( e_1 \leq e_2 \leq \ldots \leq e_n \)
  \[ S := \emptyset \]
  for \( j = 1..n \)
  \{ 
    if ( talk \( j \) is compatible with \( S \) ) 
    \[ S := S \cup \{ \text{talk } j \} \]
  \}

return \( S \) \{ \( S \) is the set of talks scheduled \}

let \( P(n) \) be the proposition that if the greedy algorithm schedules \( n \) talks in the main lecture hall, then it is not possible to schedule more than \( n \) talks in this hall.

BASIS STEP: Suppose that the greedy algorithm managed to schedule just one talk, \( t_1 \), in the main lecture hall. This means that no other talk can start at or after \( e_1 \), the end time of \( t_1 \). Otherwise, the first such talk we come to as we go through the talks in order of non decreasing end times could be added. Hence, at time \( e_1 \) each of the remaining talks needs to use the main lecture
hall because they all start before e1 and end after e1. It follows that no two talks can be scheduled because both need to use the main lecture hall at time e1. This shows that P(1) is true and completes the basis step.

**INDUCTIVE STEP:** The inductive hypothesis is that P(k) is true, where k is an arbitrary positive integer, that is, that the greedy algorithm always schedules the most possible talks when it selects k talks, where k is a positive integer, given any set of talks, no matter how many.

We must show that P(k+1) follows from the assumption that P(k) is true, that is, we must show that under the assumption of P(k), the greedy algorithm always schedules the most possible talks when it selects k+1 talks.

Now suppose that the greedy algorithm has selected k+1 talks. Our first step in completing the inductive step is to show there is a schedule including the most talks possible that contains talk t1, a talk with the earliest end time. This is easy to see because a schedule that begins with the talk ti in the list, where i > 1, can be changed so that talk t1 replaces talk ti.

To see this, note that because e1 ≤ ei, all talks that were scheduled to follow talk ti can still be scheduled. Once we included talk t1, scheduling the talks so that as many as possible are scheduled is reduced to scheduling as many talks as possible that begin at or after time e1. So, if we have scheduled as many talks as possible, the schedule of talks other than talk t1 is an optimal schedule of the original talks that begin once talk t1 has ended. Because the greedy algorithm schedules k+1 talks when it creates this schedule, we can apply the inductive hypothesis to conclude that it has scheduled the most possible talks. It follows that the greedy algorithm has scheduled the most possible talks, k+1, when it produced a schedule with k+1 talks, so P(k+1) is true. This completes the inductive step.

### A Scheduling Problem

Setup: - One shared resource (e.g., a processor).
- Many “jobs” to do (e.g., processes).

Assume: Each job has a:
- weight wj (“priority”)
- length lJ

#### Completion Times

**Definition:** The completion time Cj of job j = Sum of job lengths up to and including j.

**Example:** 3 jobs, l1 = 1, l2 = 2, l3 = 3.

**Schedule:**

- #1 #2 #3 0 →

- #1 : 1
#2 : (1+2)
#3 : (1 + 2 + 3)

## The Objective Function

Goal: Minimize the weighted sum of completion times:

\[
\min_{j=1}^{n} \sum w_j C_j
\]

Back to example: If \(w_1 = 3\), \(w_2 = 2\), \(w_3 = 1\), this sum is \(3 \cdot 1 + 2 \cdot 3 + 1 \cdot 6 = 15\).

## Greedy Algorithm

Schedule: [Just by renaming jobs] Greedy schedule \(\sigma\) is just 1,2,3,...,\(n\) where \(w_1/l_1 \geq w_2/l_2 \geq ... \geq w_n/l_n\) in this order.

## Correctness Proof

Case (1) All \(w_j/l_j\)'s distinct; that is \(w_1/l_1 > w_2/l_2 > ... > w_n/l_n\).

Greedy schedule \(\sigma\) is just 1,2,3,...,\(n\). Suppose this is not optimal and we have \(\sigma^*\) which is optimal. Hence \(\sigma^* \neq \sigma\), then there are consecutive jobs \(i,j\) with \(i > j\) in the \(\sigma^*\) schedule.

[Only \(\sigma\) schedule indices always go up is 1,2,3,...,\(n\)] hence \(w_i/l_i < w_j/l_j\Rightarrow w_i * l_j < w_j * l_i\)

\[\sigma^*\]

<table>
<thead>
<tr>
<th>C</th>
<th>stuff</th>
<th>more stuff</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>I</td>
<td>li</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>modified (\sigma^*) with (i, j) swaped</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>stuff</th>
<th>more stuff</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>I</td>
<td>li</td>
</tr>
</tbody>
</table>

\(\sigma^*\): Objective Function

\[C + (l+li)wi + (l+li +lj)wj\]---(1)

Modified \(\sigma^*\) objective function

\[C + (l+lj)wj + (l +lj + li) wi\]---(2)

\(1) - (2) = w_j * li - w_i * lj > 0\) contradicting the optimality of \(\sigma^*\)

Case (2) \(w_1/l_1 \geq w_2/l_2 \geq ... \geq w_n/l_n\).

If greedy \(\sigma\) is not optimal then there exits a schedule \(\sigma^*\) which is different.
This means we can swap the inversions. By case (1) we will either win of stay the same. In the worst case we have to make a maximum of $n \text{ chose } 2$ exchanges can transform $\sigma^*$ into $\sigma$. 