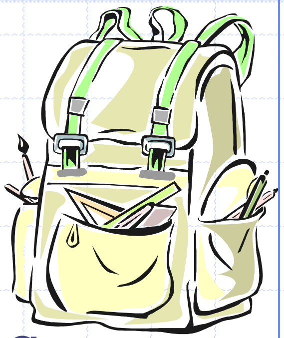
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

The Greedy Method

Civil War Knapsack. U.S. government image. Vicksburg National Military Park. Public domain.

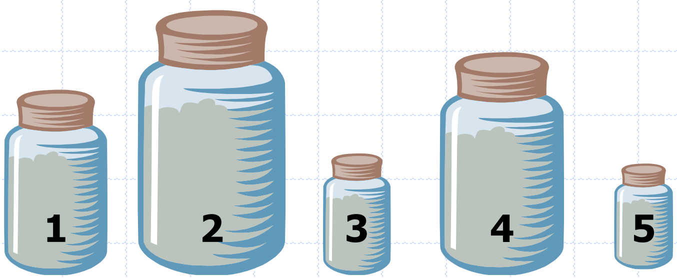
The Fractional Knapsack Problem



- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.
- If we are allowed to take fractional amounts, then this is the **fractional knapsack problem**.
  - In this case, we let x_i denote the amount we take of item i
  - Objective: maximize $\sum_{i \in S} b_i (x_i / w_i)$
  - Constraint: $\sum_{i \in S} x_i \leq W$

Example

- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.



Items:	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	$12	$32	$40	$30	$50
Value: ($ per ml)	3	4	20	5	50

“knapsack”

10 ml

Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

The Fractional Knapsack Algorithm

- Greedy choice: Keep taking item with highest value (benefit to weight ratio)
  - Since $\sum_{i \in S} b_i (x_i / w_i) = \sum_{i \in S} (b_i / w_i) x_i$
  - Run time: O(n log n). Why?
- Correctness: Suppose there is a better solution
  - there is an item i with higher value than a chosen item j, but $x_i < w_i$, $x_j > 0$ and $v_i < v_j$
  - If we substitute some i with j, we get a better solution
  - How much of i: $\min\{w_i - x_i, x_j\}$
  - Thus, there is no better solution than the greedy one

Algorithm *fractionalKnapsack*(*S*, *W*)
 Input: set *S* of items w/ benefit b_i and weight w_i; max. weight W
 Output: amount x_i of each item i to maximize benefit w/ weight at most W
 for *each item i in S*
  $x_i \leftarrow 0$
  $v_i \leftarrow b_i / w_i$ {value}
 $w \leftarrow 0$ {total weight}
 while $w < W$
  remove item i w/ highest v_i
  $x_i \leftarrow \min\{w_i, W - w\}$
  $w \leftarrow w + \min\{w_i, W - w\}$
 return x

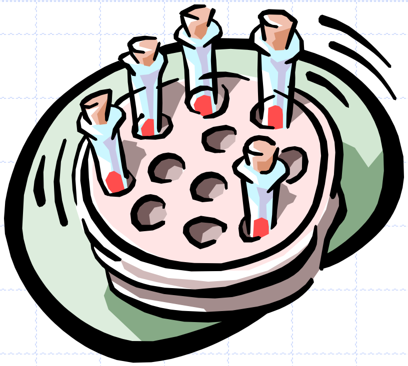
Analysis of Greedy Algorithm for Fractional Knapsack Problem

- We can sort the items by their benefit-to-weight values, and then process them in this order.
- This would require O(n log n) time to sort the items and then O(n) time to process them in the while-loop.
- To see that our algorithm is correct, suppose, for the sake of contradiction, that there is an optimal solution better than the one chosen by this greedy algorithm.
- Then there must be two items i and j such that

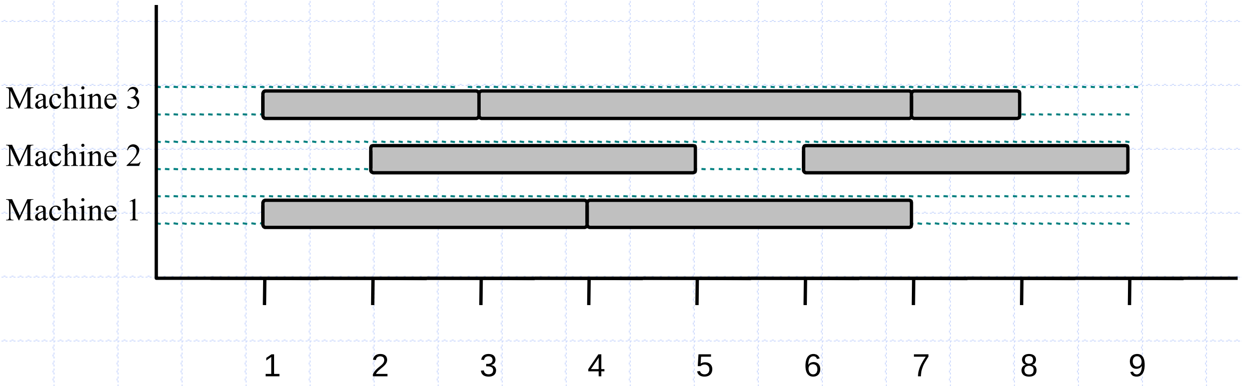
$$x_i < w_i,\ x_j > 0, \text{ and } v_i > v_j\ .$$

- Let $y = \min\{w_i - x_i, x_j\}$.
- But then we could replace an amount y of item j with an equal amount of item i, thus increasing the total benefit without changing the total weight, which contradicts the assumption that this non-greedy solution is optimal.

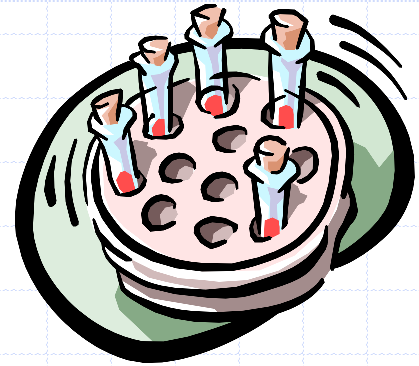
Task Scheduling



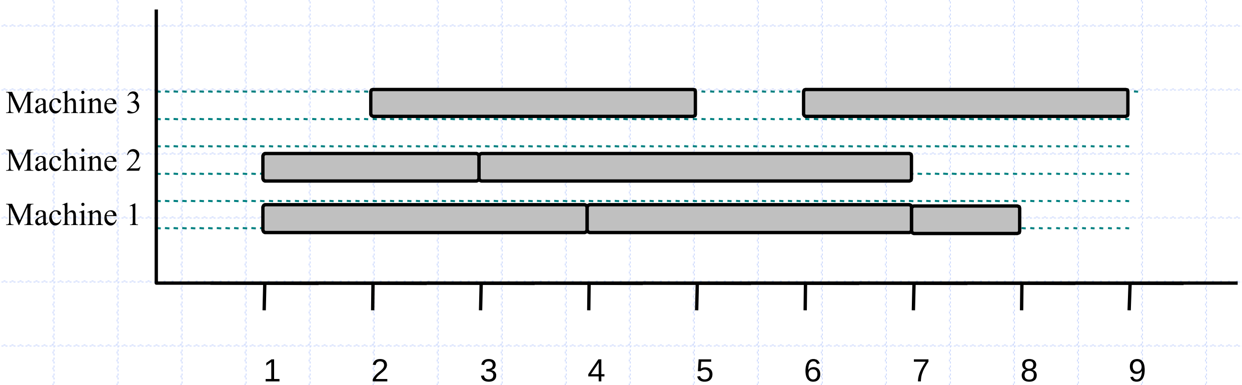
- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



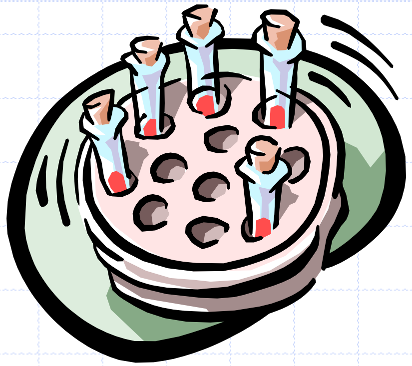
Example



- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
  - [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines



Task Scheduling Algorithm



- Greedy choice: consider tasks by their start time and use as few machines as possible with this order.
  - Run time: O(n log n). Why?
- Correctness: Suppose there is a better schedule.
  - We can use k-1 machines
  - The algorithm uses k
  - Let i be first task scheduled on machine k
  - Machine i must conflict with k-1 other tasks
  - But that means there is no non-conflicting schedule using k-1 machines

Algorithm *taskSchedule*(T)

  Input: set T of tasks w/ start time s_i and finish time f_i

  Output: non-conflicting schedule with minimum number of machines

  $m \leftarrow 0$ {no. of machines}

  while *T is not empty*

    remove task i w/ smallest s_i

    if *there's a machine j for i* **then**

      schedule i on machine j

    else

      $m \leftarrow m + 1$

      schedule i on machine m

  return *schedule*