

Problem Set 3: Greedy **Sketch**

Released 14 February 2024

Due 21 Feb 2024 10:30pm in Gradescope

Instructions. You may discuss the material with your classmates, but you must write solutions yourself. List collaborators on your submission. Do not use other sources of help other than the course staff and the textbook.

If you are asked to design an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submissions. Please submit a PDF file typed using L^AT_EX. Please assign pages to questions in Gradescope.

1. **15.1-2** Select Last activity to start instead of first to finish

Solution: This becomes exactly the same problem if we imagine time running in reverse.

2. **15.1-3** Give counter examples that fail for always selecting: Least duration, overlaps the fewest, earliest start time.

Solution:

- Least Duration: 3 elements, 2 long, 1 overlapping short
- Overlaps the fewest: many elements, lots of repeats, fewest overlap prevents maximum number
- Earliest start time: earliest start time overlaps everything

3. **15.2-3** 0-1 Knapsack that sorted by increasing weight is the same as sorted by decreasing value.

Solution: Sketch:

Assume the items are sorted as above. Then take items from left to right until the bag is full.

Validity of Greedy Choice: The greedy choice is the one that maximizes the value density as we saw in class. Assume we have an optimal solution A_o that differs from our solution. Then there is at least one pair of objects a_i, a_j such that a_i is in the greedy solution, but not the optimal solution and a_j is in the optimal solution but not the greedy solution.

However, by the problem statement, we know that a_i weighs less than or equal to a_j and has at least as much value. So we could swap a_i for a_j in our optimal solution A_o and be at least as optimal.

4. **15-1abc** Coin Changing

Solution:

(sketch)

- (a) Give the largest coin that is less than the current amount of change to give.

– Optimal Substructure

- * Assume we have a minimum set of coins to make change for n cents. Assume that coin c_i is used. Then we have a subproblem to make change for $n - c_i$ cents with the minimal number of coins. < Insert the cut and paste argument here >

– Greedy Choice

- * Let A_m be a set of coins that solves the coin-changing problem. Then a_m be a coin with the largest value less than the amount of change to give.
 - * Let A_k be a minimally sized set of coins that solves the coin changing problem.
 - * Assume a_m is not in A_k . Since 1 divides 5 and 5 divides 10, any choice of a_m other than a quarter would require multiple other coins to replace (a contradiction). For the quarter: observe that any number where a quarter could be used (25+) is more efficiently created using a quarter and other values since neither 5 nor 10 are relatively prime with 25. It is clear that any number from 25-49 is more efficiently obtained using a quarter than without a quarter. Any number above that is repeated applications of the same logic. (sketch)
- (b) Similar logic to A. Not including the largest coin that is less than the value would require more coins to make up for it. since every value divides the larger values exactly, so there is no value that can be obtained otherwise.
 - (c) Relatively prime coins 1,19,20. Goal: make change for 38 cents. Optimal: 19,19. Greedy: 20+1+...+1

5. 15-2a Average Completion Time

Solution: Order tasks from smallest to largest, run them in that order.

Must also show Optimal Substructure and Validity of Greedy Choice

Bonus (token opportunity). 15.2-7 Optimal Payoff

Solution:

Sketch: Sort both in increasing order.

Assume Current optimal solution above. Assume there is a better solution. Pick any two values i and j to get a_i, a_j, b_i, b_j such that $a_i < a_j$ and $b_i > b_j$ (two locations that differ), compare $a_i^{b_i} a_j^{b_j}$ to $a_i^{b_j} a_j^{b_i}$. Computing the difference gives us a fraction < 1 so the new ordering cannot be larger than the optimal solution assumed.

Runtime: $\Theta(1)$. Once they've been reordered, the algorithm does nothing so it takes no/constant time.