

Assignment 7 Solutions

1. Consider the maximal value interval selection problem from class: we are given a list of intervals $I_1 = (s_1, e_1, v_1), I_2 = (s_2, e_2, v_2), \dots, I_n = (s_n, e_n, v_n)$ where s_i, e_i , and v_i are the start and end times and value of interval I_i . Our goal is to select a collection of non-overlapping intervals of maximal value.

In class a proposed algorithm that I realized later was actually incorrect was to sort the intervals by value per total time, and then pick intervals greedily according to that strategy.

Convince me that this algorithm is incorrect.

Solution: Consider intervals $I_1 = (0, 5, 5), I_2 = (4, 6, 4), I_3 = (5, 10, 5)$. Then I_2 has the greatest value per unit time (I_2 has 2, I_1 and I_3 are both 1 unit value per unit time). Hence the algorithm would pick I_2 , immediately excluding I_1 and I_3 , giving a total value of 4. However, the optimal solution is to take both I_1 and I_3 , giving a total value of 10.

2. Consider the knapsack problem from class: we are given a list of items $e_1 = (w_1, v_1), e_2 = (w_2, v_2), \dots, e_n = (w_n, v_n)$ and a maximal weight W . Consider the algorithm that orders the items by value per unit weight and greedily picks according to this ordering. Argue that this algorithm is incorrect.

Solution: Let $e_1 = (4, 5), e_2 = (3, 3), e_3 = (3, 3)$ with $W = 6$. Then since the value per weight is greatest for e_1 , the algorithm will take that first, excluding e_2 and e_3 , giving a total value of 5. However, the optimal solution would be to take e_2 and e_3 to give a total value of 6.

3. (a) How many digits are required to write the number 1? **Solution:** 1
(b) How many digits are required to write the number 1000? **Solution:** 4
(c) Determine an upperbound to the number of digits to write a number n . **Solution:** $\log n + 1$
(d) In class we gave a $O(nW)$ algorithm to solve the knapsack problem. Think about part c and ask yourself if this algorithm is *technically* a polynomial time algorithm - we'll see later in this semester that it's very unlikely that there is *strongly* polynomial time solution. https://en.wikipedia.org/wiki/Time_complexity#Strongly_and_weakly_polynomial_time
Solution: Typically by polytime solutions, we mean that the runtime is a function that is polynomial with respect to the *size* of the input. In this case, the value of W is exponentially bigger than its size - as such, this is technically still an exponential algorithm.