CSE 40113: Algorithms
Homework 4

You may complete this homework in groups of 3 or less students. Note that the integrity policy applies: your group should write up your own work, although you're welcome to work on the problems in a larger group. If you have any questions, please re-read both the homework guidelines and the academic integrity policy carefully, and then come discuss any questions or concerns with me.

Required Problems

1. A local baker has come to you with the following problem. There are many orders that need to be filled, and for each item there is a baking time, b_i, and a subsequent cooling time, c_i. Each item must be baked and then cooled. Furthermore, only one item can be baked at any given time, since the oven is not large enough to fit two of the pans.

   The baker must decide on the order for baking the items. That is, the first item will be baked and then taken out of the oven to cool. The second item may be baked as soon as the first is taken out of the oven (that is, while that first item cools), and so on. The baker's goal is to minimize the overall time spent on the process (that is until all items have cooled). Your goal is to develop an efficient algorithm which produces a schedule whose completion time is as small as possible.

   To help you organize your thought, we suggest the following greedy rules:

   (a) Order the schedule based on b_i, from smallest baking time to largest baking time.

   (b) Order the schedule based on c_i, from largest cooling time to smallest cooling time.

   (c) Order the schedule based on the composite quantity $b_i + c_i$, ordered from smallest to largest.

   One of these greedy rules is guaranteed to produce the optimal schedule; the other rules are flawed. Your job is to figure out which is which! For each flawed rule, provide a relatively simple instance that demonstrates the non-optimality of the result. For the rule that is valid, give a proof that it guarantees an optimal completion time.

   Solution: Both a and c fail. To see this, consider the following schedule: the first item takes 1 minute to cook and 1 to cool, and the second takes 2 minutes to cook and 2 to cool. So both a and c would order with the first item and then the second, as this is smallest to largest bake time as well as smallest to largest sum of bake+cool, and the total time is $\max\{1+1, 1+2+2\} = 5$. But if we instead baked item 2 and then item 1, we would have $\max\{2+2, 2+1+1\} = 4$.

   So we proceed with proving that b is optimal. This intuitively makes sense, as we can cool in parallel, so getting the longest one cooling first seems reasonable. To make this formal, we suppose this greedy schedule is not optimal, for the purposes of contradiction. Then, we have optimal schedule ordering the items from o_1 to o_n, with each having baking time b_i and cooling time c_i. Since it is not sorted by cooling time, we know there must be some place where the cooling time of two neighbors is not ordered largest to smallest – if this never

happened, it would be sorted! Consider the first index i where this occurs: so we have o_i, o_{i+1} as neighbors in the order, but the cooling time c_i is smaller than c_{i+1}.

We modify the optimal solution by swapping o_i with o_{i+1}, so the order in this new schedule is now $o_1, \ldots, o_{i-1}, o_{i+1}, o_i, o_{i+2}, \ldots o_n$. We note that the completion time for everything other than o_i and o_{i+1} is unchanged: if it comes before i, then it never depending on these two, and if it comes after, then it only depended on how long these two took to bake, and $b_i + b_{i+1} = b_{i+1} + b_i$, so total time is unchanged.

The worst case time for the entire schedule is maximum over all items k of the time to bake everything up to this item, plus the time to bake and cool the k^{th} item: in other words,

$$\max_k \{(\sum_{j<k} b_j) + (b_k + c_k)\}$$

. Before and after swapping, none of the items other than item i and $i+1$ change, so if they are the maximum, they remain unchanged and we're no worse off.

But both i and $i+1$ change as follows: before the swap, item i took $(\sum_{j<i} b_j) + b_i + c_i$ and item $i+1$ took $(\sum_{j<i} b_j) + b_i + b_{i+1} + c_{i+1}$. Note that we know $c_i < c_{i+1}$, and that baking times are not negative, so we can immediately say that $(\sum_{j<i} b_j) + b_i + b_{i+1} + c_{i+1}$ is the larger of these two in terms of finish time. After the swap, we have item $i+1$ finishing at time $(\sum_{j<i} b_j) + b_{i+1} + c_{i+1}$, and item i at time $(\sum_{j<i} b_j) + b_{i+1} + b_i + c_i$. It is not clear which of these is larger, but we can argue that *both* are smaller than the max before we swapped: the sum is unchanged in all terms, and so we're really comparing $b_{i+1} + c_{i+1}$ and $b_{i+1} + b_i + c_i$ to $b_i + b_{i+1} + c_{i+1}$. The first is smaller because we know $b_i > 0$, and the second because we assumed $c_i < c_{i+1}$.

Therefore, if our schedule is not sorted by cooling time from largest to smallest, we can improve it by swapping out of order pairs. This means the optimal schedule can improve, which contradicts the fact that it was ever optimal.

■

2. Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., $n = 6$ and the values are 1,5,10,25,50, and 100 cents, unless we get ride of the penny soon[1].) Your beloved and benevolent dictator, El Generalissimo, has decreed that whenever you give a customer change you must use the smallest possible number of coins, so as not to wear out the image of him so lovingly engraved on each coin.

   (a) In the U.S., there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin which is not too large and recursively give change for the reminder. El Generalissimo does not approve of American capitalist greed, however. Show that there is a set of coin values for which the greedy algorithm does not always give the smallest possible number of coins.

   Solution: Consider coins of value 1, 3 and 4, with a target of 6. Greedy will return 3 coins $(4 + 1 + 1)$, but we can return 2 coins $(3 + 3)$ and do better than greedy, so it cannot be optimal. ■

[1] See https://www.npr.org/2025/02/10/nx-s1-5292082/trump-penny-mint-treasury

(b) Now suppose El Generalissimo decides to impose a currency system where the coin denominations are consecutive powers $b^0, b^1, b^2, \ldots, b^k$ of some integer $b \geq 2$. Prove that despite El Generalissimo's disapproval, the greedy algorithm described in part (a) does make optimal change in this system as well.

(Hint: You can start with powers of 2, if that makes things more concrete, but then be sure to justify why it generalizes for larger bases as well!)

Solution: We begin with a simple fact, which we'll prove via induction on k: Given denominations of $b^0, \ldots, b^k$, replacing any coin of value b^k will require at least b smaller coins. (Note: I did not require a formal inductive proof, but did require some justification of this fact in your solution.)

Proof: Base case: Suppose $k = 1$. Replacing a single coin of value $b = b^1$ with coins of value $1 = b^0$ will take exactly b 1 cent coins, since we have no other available higher coins.

Inductive hypothesis: Assume for some value $k-1$, we need at least b coins of smaller values $b^0, \ldots, b^{k-2}$ to replace a coin of value b^{k-1}

Inductive step: Consider a coin of value b^k. Our only options to make change are of value $b^0, \ldots b^{k-1}$. If we choose coins of value b^{k-1}, we will need exactly $\frac{b^k}{b^{k-1}} = b$. If we choose to use smaller values of coins, we will need to remove at least one coin of value b^{k-1} and make up the difference with smallercoints; by the induction hypothesis, each coin of value b^{k-1} replaced with smaller coins will require at least b more, meaning in total we will need more than b smaller coins. Either way, we need at least b coins. □

Now consider a greedy solution we make change for a target value n using g_i coins of value b^i, so $n = \sum_{i=1}^{k} g_i b^i$. Suppose for the purposes of contradiction that the optimal solution is not greedy, and uses o_i coins for target coin value b^i, so $n = \sum_{i=0}^{k} o_i b^i$. Going from largest coin to smallest in these solution, consider they first place they differ: so the first i where $g_i \neq o_i$. Since greedy chose as many as possible, we know that in fact we must have $o_i < g_i$ at this point, as opt could not choose more without exceeding the target value. But this means there is at least one coin of value b^i which we must make up for in optimal using smaller coins, since any larger coin will be too large (and greedy took the maximum possible for each of those coin denominations, so opt cannot include more.). By our fact above, this means opt must use at least b more coins than greedy, which means it is not optimal, as greedy will use fewer coins. ■

3. You've been hired to store a sequence of n books on shelves in a library. The order of the books is fixed by the cataloging system and cannot be changed; each shelf must store a contiguous interval of the given sequence of books. You are given two arrays $H[1..n]$ and $T[1...n]$, where $H[i]$ and $T[i]$ are respectively the height and thickness of the i^{th} book in the sequence. All shelves in this library have the same length L; the total thickness of all books on any single shelf cannot exceed L.

(a) Suppose all the books have the same height h and the shelves have height larger than h, so every book fits on every shelf. Describe and analyze a greedy algorithm to store the books in as few shelves as possible. [Hint: The algorithm is obvious, but why is it correct?]

Solution: The "obvious" greedy algorithm is to maximize the number of books per shelf, only moving to the next shelf if the book is too large to fit. More formally:

```
MinShelves(T[1..n], L):
  shelves ← 1
  currentThickness ← 0
  for i ← 1 to n
      if currentThickness +T[i] ≤ L
          currentThickness + = T[i]
      else
          shelves ← shelves +1
          currentThickness ← T[i]
  return shelves
```

The for loops repeats n times with work $O(1)$ per iteration, so total time is $O(n)$

Correctness: Suppose the optimal solution is not this greedy one, so suppose optimal has some placement of books onto shelves that uses k shelves total, where k is smaller than the greedy number of shelves g. If the solutions are different, there must be some first book i which optimal must place differently than greedy. Consider this book i, which we say was placed on shelf j: we know greedy continues on the same shelf until there is no room, and we are not allowed to reorder books, so optimal must place this book on shelf j while greedy put it on shelf $j-1$. We can modify optimal as follows: move book i back up to shelf $j-1$. The book fits (because greedy placed it there), and this new solution must be viable, because all we have done is decrease the contents of shelf j by $T[i]$, so it is still less than L. This means we have a new optimal which did not differ from greedy at book i, contradicting our assumption.

(Alternately, instead of the contradiction, we could note that at any book i where optimal and greedy differ, we can make this exchange, and if we continue this swapping for all books i that are different, we will end with a greedy solution that is no worse than optimal.)

■

(b) That was a nice warmup, but now here's the real problem. In fact the books have different heights, but you can adjust the height of each shelf to match the tallest book on that shelf. (In particular, you can change the height of any empty shelf to zero.) Now your task is to store the books so that the sum of the heights of the shelves is as small as possible. Show that your greedy algorithm from part (a) does not always give the best solution to this problem. [Note: if you wanted to solve this optimally, you'd want to try dynamic programming!]

Solution: Consider 3 books, all of width 1, and of heights $[1, 2, 2]$, where max shelf width is 2. Greedy will put books 1 and 2 on a single shelf and 3 on a new shelf, for a total height of 4, but optimal would be to place 1 on its own shelf and 2 and 3 on a second shelf, for a total height of 3. ■

4. **Sample solved problem:**

Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month.

Each license is currently selling for a price of \$100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100 \cdot r_j^t$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the same price of \$100).

The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible?

Give an algorithm that takes the n rates of price growth $r_1, r_2, \ldots, r_n$, and computes an order in which to buy the licenses so that the total amount of money spent is minimized. The running time of your algorithm should be polynomial in n.

Solution:

Two natural guesses for a good sequence would be to sort the r_i in decreasing order, or to sort them in increasing order. Faced with alternatives like this, it's perfectly reasonable to work out a small example and see if the example eliminates at least one of them. Here we could try $r_1 = 2, r_2 = 3$, and $r_3 = 4$. Buying the licenses in increasing order results in a total cost of

$$100(2 + 3^2 + 4^3) = 7,500,$$

while buying them in decreasing order results in a total cost of

$$100(4 + 3^2 + 2^3) = 2,100.$$

This tells us that increasing order is not the way to go. (On the other hand, it doesn't tell us immediately that decreasing order is the right answer, but our goal was just to eliminate one of the two options.)

Let's try proving that sorting the r_i in decreasing order in fact always gives the optimal solution. When a greedy algorithm works for problems like this, in which we put a set of things in an optimal order, we've seen in the text that it's often effective to try proving correctness using an exchange argument.

To do this here, let's suppose that there is an optimal solution O that differs from our solution S. (In other words, S consists of the licenses sorted in decreasing order.) So this optimal solution O must contain an inversion—that is, there must exist two neighboring months t and $t + 1$ such that the price increase rate of the license bought in month t (let us denote it by r_t) is less than that bought in month $t+1$ (similarly, we use r_{t+1} to denote this). Given that these must not be greedily ordered, we have $r_t < r_{t+1}$.

We claim that by exchanging these two purchases, we can strictly improve our optimal solution, which contradicts the assumption that O was optimal. Therefore if we succeed in showing this, we will successfully show that our algorithm is indeed the correct one.

Notice that if we swap these two purchases, the rest of the purchases are identically priced. In O, the amount paid during the two months involved in the swap is $100(r_t^t + r_{t+1}^{t+1})$. On

the other hand, if we swapped these two purchases, we would pay $100(r_{t+1}^t + r_t^{t+1})$. Since the constant 100 is common to both expressions, we want to show that the second term is less than the first one. So essentially, we want to show that $r_{t+1}^t + r_t^{t+1} < r_t^t + r_{t+1}^{t+1}$. If we rearrange to group like terms, we get the equivalent statement $r_t^{t+1} - r_t^t < r_{t+1}^{t+1} + r_{t+1}^t$, which is then equivalent to: $r_t(r_t - 1) < r_{t+1}^t(r_{t+1} - 1)$. This last inequality is true simply because $r_t > 1$ for all i, and because we started by assuming $r_t < r_{t+1}$, which implies the original desired inequality is also true since they are equivalent.

Our algorithm is then quite simple: sort by decreasing order and output the list. The algorithm's runtime is $O(n \log n)$ since that is the running time to sort.

Side Note: It's interesting to note that things become much less straightforward if we vary this question even a little. Suppose that instead of buying licenses whose prices increase, you're trying to sell off equipment whose cost is depreciating. Item i depreciates at a factor of $r_i < 1$ per month, starting from \$100, so if you sell it t months from now you will receive $100 \cdot r_i^t$. (In other words, the exponential rates are now less than 1, instead of greater than 1.) Ifyou can only sell one item per month, what is the optimal order in which to sell them? Here, it turns out that there are cases in which the optimal solution doesn't put the rates in either increasing or decreasing order (as in the input $\frac{3}{4}, \frac{1}{2}, \frac{1}{100}$).