Kaleb Kim
11/20/24

CSDS 310 Assignment 4

Note: Arrays are zero-indexed.

Problem 1

a) Counterexample: Consider activities $a_1 = [1, 4], a_2 = [2, 3]$, and $a_3 = [3, 4]$. The optimal solution is $\{a_2, a_3\}$. This greedy approach selects a_1 first, which creates the unoptimal set $\{a_1\}$.
b) Counterexample: Consider activities $a_1 = [1, 3], a_2 = [3, 5], a_3 = [2, 6]$. The optimal solution is $\{a_1\}$ Selecting a_1(shortest duration) excludes a_2, and a_3 The optimal solution is $\{a_1, a_2\}$
c) Counterexample: there isn't one this one is true

—

Problem 2

Pseudocode

procedure MAX_PROFIT(A, B, n):
  r-quicksort A
  r-quicksort B
  profit $\leftarrow$ 1
  for $0 \leq i < n$:
    profit $\leftarrow$ profit $\times$ $A[i]^{B[i]}$
  return profit

Proof

Pairing larger values amplifies the exponentiation result.

- Example: $A = [3, 1], B = [2, 4]$ Reordering $A = [3, 1], B = [4, 2]$maximizes the result: $3^4 c \cdot 1^2 = 81$versus other arrangements.

Runtime

Randomized quick sort takes $O(n \log n)$ time in the worst case. It also sorts in place, using $O(1)$ extra space. The for loop runs for $\Theta(n)$ time. Thus, we have:

$$\text{Time complexity: } O(n \log n)$$
$$\text{Space complexity: } O(1)$$

Problem 3

Algorithm

- Sort activities by their deadlines $d_i$in ascending order.
- Start scheduling from $t = 0$, assigning $s_i = \max(t, 0)$and updating $t = s_i + t_i$

Example

- Input: $t = [10, 5, 6, 2], d = [11, 6, 12, 20]$
- Sorted by deadlines: $\{(5, 6), (10, 11), (6, 12), (2, 20)\}$
- Scheduled order: $[2, 1, 3, 4]$
- Starting/Finishing times: $\left[\frac{0}{5}, \frac{5}{15}, \frac{15}{21}, \frac{21}{23}\right]$

- Maximum delay: $\Delta = \max(-1, 4, 9, -3) = 9$

Explanation

- Sorting by deadlines minimizes delays, ensuring earlier deadlines are prioritized.

Problem 4

a) With the counterexample $c = [1, 3, 4]$ and $n = 6$, the optimal solution is 2 coins ($\{3, 3\}$). However, the greedy choice is 3 coins ($\{4, 1, 1\}$).

b) We must prove that if the coin denominations are powers of 2, then this greedy choice leads to the optimal solution. Let a represent the coins needed to make n based on the greedy choice.

- Base case:
  When $n = 1$, we have $c = [1]$ so $a = 1$ (coins = $\{1\}$).
  When $n = 2$, we have $c = [1, 2]$ so that $a = 1$ (coins = $\{2\}$).
  When $n = 2$, we have $c = [1, 2]$ so that $a = 1$ (coins = $\{2, 1\}$).
- Inductive step: Having proven that a is optimal for $1 \leq n \leq b$ such that $b = 3$, we must prove $b + 1$. From our base case, we notice that for the largest coin c_k, we have $k = \lfloor \log_2(b) \rfloor$. Then, we have $k' = \lfloor \log_2(b + 1) \rfloor$. This means that if $b + 1$ can be expressed as 2^d with $d \in \mathbb{Z}$, we have $a' = a - c_{k'}$, another optimal solution. This also works for cases in which $b + 1$ cannot be expressed as such. Therefore, the greedy solution is always optimal for n.