Homework 3

● Graded

Student

Abhishek Soundalgekar

Total Points

60 / 60 pts

Question 1

Q1 **15** / 15 pts

+ 0 pts Incorrect

✔ **+ 15 pts** Correct

+ 3 pts establishing the base case (k=1) correctly.

+ 8 pts Inductive Hypothesis and Step

+ 4 pts explaining the k-th step

Question 2

Q2 **15** / 15 pts

✔ **+ 15 pts** Full points

+ 0 pts Incorrect

+ 5 pts Greedy algorithm

+ 2 pts (part a) Stay-ahead argument" of the proof
– Induction base: 2 pts

+ 3 pts (part a) Stay-ahead argument" of the proof
– Induction hypothesis: 3pts

+ 3 pts (part a) Stay-ahead argument" of the proof
– Induction step: 3 pts

+ 2 pts Completing the argument of optimality in the proof (part b)

+ 7.5 pts Wrong question

Question 3

Q3 **15** / 15 pts

✔ **– 0 pts** Correct algorithm

– 3 pts for not mentioning sorting in descending order of B.

– 2 pts for not mentioning Computer A follows the sorted order.

– 0 pts Correct proof

– 2 pts for not defining what the inversions in this case are

– 2 pts for not generalizing the proof

– 1 pt for not showing that the algorithm is now the same as our greedy algorithm

Question 4

Q4 a **10** / 10 pts

+ **0 pts** Incorrect

✔ + **4 pts** Algorithm

✔ + **3 pts** Claim and proof

✔ + **3 pts** Final proof with induction

Question 5

Q4 b **5** / 5 pts

✔ + **5 pts** Correct example

– **5 pts** Click here to replace this description.

Question assigned to the following page: 1

CSCI 570 Spring 2025

Homework 3

Name: Abhishek Sowdalgekar

USC ID: 2089011000

Question 1

Answer 1

To show that the greedy algorithm minimizes the number of trucks required, we use a 'staying ahead' argument paired with a contradiction-based analysis. This proof formalizes why no alternative packing strategy can outperform the greedy approach under the problem's constraints.

According to the Greedy Algorithm the boxes are packed in the order they arrive. Each truck is filled until the next box exceeds the weight limit W, at which point the truck departs.

Optimal solution would be a hypothetical strategy that uses the fewest possible trucks while respecting the order of boxes.

Question assigned to the following page: 1

Departure Index - For a truck, this is the index of the last box it carries. Let s_i denote the departure index of the i^{th} truck in the greedy solution and t_i for the i^{th} truck in the optimal solution.

We prove by induction that every greedy truck departs with a later (or equal) box index compared to the corresponding optimal truck. Formally $s_i \geq t_i$ for all $i \leq k$ [$k \rightarrow$ number of trucks in the optimal solution]

1. Base case ($i=1$)

The first greedy truck G_1 packs boxes starting from index 1 and departs at s_1, the largest index where the total weight $\leq W$. Since the optimal solution must also start at index 1, it cannot pack beyond s_1 without violating W. Thus $s_1 \geq t_1$.

2. Inductive step :

Assume $s_{i-1} \geq t_{i-1}$ for the $(i-1)^{th}$ truck. The i^{th} greedy truck G_i starts at box $s_{i-1}+1$, while the optimal's i^{th} truck O_i starts at $t_{i-1}+1$. Since $s_{i-1} \geq t_{i-1}$, the greedy truck G_i begins

Question assigned to the following page: 1

no earlier than O_i. By design, the greedy packs as many boxes as possible from its starting point, so b_i must depart with $s_i \geq t_i$.

Contradiction for Optimality

Suppose the greedy algorithm uses more trucks than the optimal solution ($g > o$). By the staying ahead property, after o trucks, the greedy solution would have covered: $s_o \geq t_o = n$

$n \rightarrow$ number of total boxes. This implies the greedy algorithm already shipped all boxes in o trucks, contradicting $g > o$. Thus, $g \leq o$. Since o is minimal by definition $g = o$.

The order constraint matters.

The requirement to ship boxes in arrival order is critical. Any alternative strategy that reorders boxes could potentially use fewer trucks, but such reordering is not allowed. The greedy algorithm's adherence to the order ensures it never 'holds back' a box to optimize future trucks - a restriction that the staying ahead argument formalizes.

Question assigned to the following page: 2

Question 2
Answer 2

Greedy Algorithm
The algorithm will follow a greedy strategy to go as far as possible before stopping for gas. Suppose we are at the i^{th} gas station, if we have enough gas to go to the $(i+1)^{th}$ gas station, then we skip the i^{th} gas station. Otherwise stop at the i^{th} station and fill up the tank.

Algorithm for Minimizing Gas Stops:

You start at USC with a full tank that can drive up to P miles. There are n gas stations at distances $d_1 \leq d_2 \leq \ldots \leq d_n$ from USC. The distances between USC and the first station, neighboring stations, and the last station to Santa Monica are all $\leq P$. The goal is to reach Santa Monica with the fewest gas stops.

Question assigned to the following page: 2

Greedy Algorithm

1. Initialize:
   1.1 Start at USC with a full tank (fuel range = p).
   1.2 Let current_position = 0 (USC)
   1.3 Let stops = 0

2. Iterate through stations:
   2.1 For each gas station i (from 1 to n):
   2.2 If the distance to the next station exceeds the current fuel range:
      2.2.1 Refuel at station i:
         2.2.1.1 Increment stops
         2.2.1.2 Reset the fuel range to p.
   2.3 Update current_position to d_i.

3. Termination:
   3.1 If Santa Monica is reachable from the last station with the remaining fuel, return stops.

Question assigned to the following page: 2

Proof of Optimality :

The greedy algorithm always refuels at the farthest possible station to minimize stops. We can prove this strategy is optimal using induction.

Part (a) : Greedy stops are not earlier than optimal stops

Let $g_1, g_2, \ldots, g_m$ be the stations where the greedy algorithm stops, and $h_1, h_2, \ldots, h_k$ ($k \leq m$) be an optimal solution. We show $h_i \leq g_i$ for all $i \leq k$.

- Base case : $h_1 \leq g_1$.

→ The greedy algorithm stops at g_1 because it is the farthest station reachable from USC without refueling. Any optimal solution must stop at or before g_1; otherwise, the car cannot reach beyond g_1.

Question assigned to the following page: 2

- Inductive step: Assume $h_c \leq g_c$.

→ After stopping at h_c, the optimal solution has fuel to reach stations up to $h_c + p$. The greedy algorithm, stopping at $g_c \geq h_c$, has fuel to reach at least $g_c + p \geq h_c + p$. Thus, the next optimal stop h_{c+1} cannot be later than the greedy's next stop g_{c+1}, since g_{c+1} is the farthest station reachable from g_c.

Part b: Greedy uses no more stops than the optimal solution.

Suppose the optimal solution $h_1, \ldots h_k$ has $k < m$ stops. By part (a), $h_k \leq g_k$. However, the greedy algorithm requires g_{k+1} because the distance from g_k to Santa Monica exceeds p. Since $h_k \leq g_k$, the optimal solution would also fail to reach Santa Monica after h_k, contradicting its feasibility. Thus, $k \geq m$, proving the greedy solution is optimal.

Time complexity:

Total time: $O(n)$, as there are n stations and constant time operations per station.

Question assigned to the following page: 3

Question 3
Answer 3

Tasks : N tasks, each with two parts : a_i (first part on computer A) and b_i (second part on computer B).

Constraints :
→ Computer A processes tasks sequentially (one at a time)
→ Computer B processes all second parts in parallel.
→ The second part b_i of a task can only start after the first part a_i completes on A.

Goal : Minimize the total completion time (Makespan)

* Algorithm
1. Sort : Arrange tasks in decreasing order of b_i (longest b_i first).
2. Execute on A : Process the first parts a_i sequentially on computer A in this sorted order.
3. Execute on B : For each task i, start b_i immediately after a_i finishes on A. Computer B handles all b_i in parallel.

Question assigned to the following page: 3

Proof of optimality:

We use an exchange argument to prove that the greedy schedule G (sorted by decreasing b_i) is optimal.

→ Inversion: A pair of tasks J_k and J_l forms an inversion if $b_k \leq b_l$, but J_k is scheduled before J_l.

⇒ Step 1: Removing Inversions:
Assume there exists an optimal schedule $S \neq G$ with at least one inversion. Let J_k and J_l be adjacent tasks in S where J_k precedes J_l but $b_k \leq b_l$. Swap J_k and J_l to create a new schedule S'.

- Original Completion Times:
  J_k: $C_{prev} + a_k + b_k$
  J_l: $C_{prev} + a_k + a_l + b_l$
  Makespan: $\max(C_{prev} + a_k + b_k,\ C_{prev} + a_k + a_l + b_l)$.

- After Swapping
  J_l: $C_{prev} + a_l + b_l$
  J_k: $C_{prev} + a_l + a_k + b_k$
  Makespan: $\max(C_{prev} + a_l + b_l,\ C_{prev} + a_l + a_k + b_k)$.

Question assigned to the following page: 3

Since $b_\ell \geq b_k$, we get

$C_{prev} + a_\ell + b_\ell \leq C_{prev} + a_k + a_\ell + b_\ell$

and

$C_{prev} + a_\ell + a_k + b_k \leq C_{prev} + a_k + a_\ell + b_\ell$.

Therefore the makespan of S' is no worse than S.

⇒ Step 2: Iterative Inversion Removal

- Repeatedly swap adjacent inversions in S until no inversions remain. This converts S into G.
- Since each swap does not increase the makespan, the final schedule G is at least as good as S.

Any optimal schedule S can be transformed into G without increasing the makespan. Therefore, G is optimal.

Time complexity:

Sorting: $O(n \log n)$ to sort tasks by b_i

Execution: $O(n)$ to compute cumulative sums & track the maximum $(\text{sum}(a_1 \ldots a_i) + b_i)$.

Total: $O(n \log n)$.

Question assigned to the following page: 4

Question 4
Answer 4

[a] Greedy algorithm for making change with coins

Algorithm:
1. Initialize: Start with n cents
2. Iterate:
2.1 At each step, select the largest coin c_k such that $c_k \leq n$.
2.2 Add c_k to the solution set S.
2.3 Subtract c_k from n.
3 Terminate: Repeat until $n = 0$.

Proof of Optimality:
An optimal solution must satisfy the following constraints:
1. Pennies: At most 4 pennies (since 5 pennies can be replaced by 1 nickel).
2. Nickels: At most 1 nickel (since 2 nickels can be replaced by 1 dime).
3. Dimes: At most 2 dimes (since 3 dimes can be replaced by 1 quarter & 1 nickel).
4. Dimes + Nickels: If 2 dimes are used, no nickels are allowed (since 2 dimes + 1 nickel = 25 cents, equivalent to 1 quarter).

Question assigned to the following page: 4

→ For $1 \leq n \leq 5$: Use pennies

→ For $5 \leq n < 10$: Use 1 nickel (using pennies would require ≥ 5, violating penny constraint).

→ For $10 \leq n < 25$: Use 1 dime (using nickels / pennies would exceed 9 cents)

→ For $n \geq 25$: Use 1 quarter (using dimes / nickels / pennies would exceed 24 cents).

Inductive Argument :
by always choosing the largest coin $c_k \leq n$, the greedy algorithm reduces n to $n - c_k$, which is optimally solved by the same strategy. Since each step adheres to the constraints above, the solution is optimal.

[b] Coin set where Greedy Fails
$\{1, 3, 4\}$

Greedy solution :
1. Take the largest coin ≤ 6 : 4. Remaining $6 - 4 = 2$
2. Take 1-cent coins twice
Total coins $1 + 2 = 3$.

Optimal solution :
$2 \times$ 3-cent coins
Total coins : 2

Question assigned to the following page: 4

[The greedy algorithm works for coins as each denomination is a multiple of the smaller ones. For arbitrary coin sets it may fail, and DP. is required for optimality.]

Reason for Failure:
The greedy algorithm prioritizes the largest coin (4) first, but the remaining 2 cents must be made with pennies.
The optimal solution uses smaller coins (3) that divide in evenly.

★ Another set of coins could be {1, 15, 20} (includes a penny for completeness).

counterexample ⇒ n = 30 cents.

Greedy Solution:
1. Take 1×20-cent coin. Remaining: 30−20 = 10.
2. Use 10×1-cent coins
Total coins: 1 + 10 = 11.

Optimal Solution:
2×15 cent coins
Total coins: 2

The greedy algorithm prioritizes the largest coin (20) first, but the remaining 10 cents can only be made with pennies. In contrast the optimal solution uses smaller coins (15) that better divide the total amount.
Violates 'greedy choice property' because coin denominations not multiples of each other (eg. 20 is not a multiple of 15).