Analysis of Greedy Algorithms

September 13, 2025

Question 1: Interval Coloring

Problem 1. *Let X be a set of n intervals on the real line. A coloring of X assigns a color to each interval so that any two overlapping intervals are differently colored. Design and analyze a greedy algorithm to compute the minimum number of colors needed to color X. Prove its correctness and optimality.*

The Greedy Algorithm

The greedy strategy is as follows:

1. Sort the intervals in X by their start times in increasing order.
2. Iterate through the sorted intervals. For each interval, assign it the smallest possible integer color (i.e., color 1, 2, 3, ...) that is not currently being used by any of its neighbors that it overlaps with.

Proof of Correctness

Definition 1 (Correct Coloring)**.** A coloring of a set of intervals is **correct** if for any two intervals I_a and I_b that overlap, they are assigned different colors.

Theorem 1. *The greedy algorithm, which sorts intervals by start times and assigns each interval the smallest available color, produces a correct coloring.*

Proof. Let's consider any two arbitrary intervals that overlap, denoted I_a and I_b. We must show that the algorithm assigns them different colors, i.e., $\text{color}(I_a) \neq \text{color}(I_b)$.

Without loss of generality, assume that I_a starts before or at the same time as I_b. This means $s(I_a) \leq s(I_b)$.

Since the algorithm processes intervals in increasing order of their start times, it will process and assign a color to I_a before it processes I_b. Let the color assigned to I_a be c_a.

Later, when the algorithm considers interval I_b, it must determine a color for it. We know the following two facts:

1. I_a and I_b overlap.
2. $s(I_a) \leq s(I_b)$.

These two facts imply that interval I_a has not finished when interval I_b begins. Therefore, at the moment the algorithm is assigning a color to I_b, the interval I_a is an active, overlapping neighbor.

By its definition, the greedy algorithm must assign to I_b the smallest-indexed color that is not currently in use by any of its active, overlapping neighbors. Since I_a is such a neighbor and it is using color c_a, the color c_a is unavailable for I_b.

Consequently, the algorithm must assign a different color, c_b, to I_b, where $c_b \neq c_a$. Since this holds for any pair of overlapping intervals, the coloring produced by the algorithm is correct. □

Proof of Optimality

Definition 2 (Optimality)**.** A coloring is **optimal** if it is correct and uses the minimum possible number of colors.

Definition 3 (Depth, k)**.** Let the **depth** of a set of intervals be the maximum number of intervals that overlap at any single point on the real line. Let this value be k.

Theorem 2. *The greedy algorithm is optimal and uses exactly k colors.*

Direct Proof. We prove this in two parts. First, we establish a lower bound on the number of colors required, showing at least k colors are necessary. Second, we establish an upper bound, showing that the greedy algorithm uses at most k colors.

Part 1: Lower Bound (Necessity)

By the definition of depth, there exists some point p on the real line where k intervals mutually overlap. Let this set of intervals be $\{I_1, I_2, \ldots, I_k\}$.

Since all k of these intervals contain the point p, every interval in this set overlaps with every other interval in the set. According to the rules of a correct coloring, each of these k intervals must be assigned a unique color.

Therefore, any correct coloring of the set of intervals must use at least k colors. This establishes that the optimal number of colors, OPT, satisfies OPT $\geq k$.

Part 2: Upper Bound (Sufficiency)

Next, we show that our greedy algorithm uses at most k colors. We will use a direct proof.

Consider an arbitrary interval, I_v, being processed by the algorithm. Assume the algorithm assigns it color c. The reason it chose color c is that colors $\{1, 2, \ldots, c-1\}$ were unavailable.

This means that at the moment of coloring I_v, there were $c-1$ other intervals, let's call them $\{I_{u_1}, I_{u_2}, \ldots, I_{u_{c-1}}\}$, that met the following criteria:

1. They were already colored with colors $\{1, 2, \ldots, c-1\}$ respectively.
2. They all overlap with I_v.

Because the algorithm sorts by start times, we know that $s(I_{u_j}) \leq s(I_v)$ for all $j \in \{1, \ldots, c-1\}$. Because each I_{u_j} overlaps with I_v, its end time must be after I_v begins, i.e., $e(I_{u_j}) > s(I_v)$.

This implies that at the specific point in time $s(I_v)$, all $c-1$ of these intervals are active. The interval I_v itself is also active at this point. Thus, we have found a point on the real line, $s(I_v)$, where at least c intervals are mutually overlapping.

By the definition of depth, the maximum number of intervals overlapping at any point is k. Therefore, the number of overlaps we just found, c, must be less than or equal to k. That is, $c \leq k$.

Since we chose I_v arbitrarily, this shows that any color c assigned by the algorithm will be at most k. Thus, the algorithm uses no more than k colors.

Conclusion

From Part 1, we know the optimal number of colors is at least k (OPT $\geq k$). From Part 2, we know our algorithm produces a correct coloring using at most k colors. Combining these, we have that our algorithm's result, k_{alg}, satisfies $k_{alg} \leq k \leq$ OPT. Since k_{alg} is a valid solution, it must be that $k_{alg} \geq$ OPT. The only way for this to be true is if $k_{alg} = k =$ OPT. Thus, the greedy algorithm is optimal. □

Question 2: Change-Making Problem

Problem 2. *Given a set of integer coin denominations $D = \{d_1, d_2, \ldots, d_k\}$ and a target integer value V, find a set of coins $C = \{c_1, c_2, \ldots, c_m\}$ such that each $c_i \in D$, the sum of the coins is exactly V ($\sum_{i=1}^{m} c_i = V$), and the number of coins, m, is minimized. Analyze the greedy algorithm for the specific case of the US currency system, $D = \{1, 5, 10, 25\}$.*

The Greedy Algorithm

The greedy approach to solve the Change-Making Problem is as follows:

1. Initialize an empty solution set, $C = \emptyset$.
2. Initialize the remaining value to be made, $V_{rem} = V$.
3. While $V_{rem} > 0$:
   (a) Find the largest denomination $d_{\max} \in D$ such that $d_{\max} \leq V_{rem}$.
   (b) Add $d_{\max}$ to the solution set C.
   (c) Update the remaining value: $V_{rem} = V_{rem} - d_{\max}$.
4. Return the set C.

Proof of Optimality for US Currency

We will demonstrate the optimality of the greedy algorithm for the US currency system $D = \{1, 5, 10, 25\}$ using three different proof techniques.

Theorem 3. *For the coin system $D = \{1, 5, 10, 25\}$, the greedy algorithm is optimal.*

Proof 1: Proof by Contradiction

Proof by Contradiction. **1. The Assumption**
Assume for the sake of contradiction that the greedy algorithm is **not** optimal. This means there exists some target value V for which the Greedy Solution (G) uses more coins than an actual Optimal Solution (O). Let $|G|$ be the number of coins in the greedy solution and $|O|$ be the number of coins in the optimal solution. Our assumption is:

$$|O| < |G|$$

2. The Analysis
Let's compare the coins in the two solutions, sorted from largest to smallest. Since the solutions are different but sum to the same value V, there must be a first coin where they differ. Let this be at index j. Let g_j be the coin chosen by the greedy algorithm and o_j be the coin in the optimal solution.

By the nature of the greedy algorithm, it always picks the largest possible coin for the remaining value. Since the first $j-1$ coins were the same in both solutions, the remaining value was identical when the j^{th} coin was chosen. The greedy algorithm chose g_j. This means g_j was the largest possible coin choice. Therefore, the coin o_j chosen by the optimal solution at this step must be smaller.

$$g_j > o_j$$

3. Reaching the Contradiction
Let's look at the remaining sum V' that needs to be made after the first $j-1$ coins are taken. The greedy solution G pays g_j towards this sum, while the optimal solution O must form the value g_j (or more) using its coins from index j onwards, all of which are smaller than g_j.

- If $g_j = 25$**c**: The optimal solution O must use coins smaller than 25c. Any combination of dimes, nickels, and pennies summing to 25c requires at least 3 coins (e.g., $\{10, 10, 5\}$).

- If $g_j = 10\mathbf{c}$: The optimal solution O must use coins smaller than 10c. Any combination of nickels and pennies summing to 10c requires at least 2 coins (e.g., $\{5, 5\}$).
- If $g_j = 5\mathbf{c}$: The optimal solution O must use 1c coins, requiring 5 coins.

In every case, to make up the value of the single greedy coin g_j, the supposedly optimal solution O must use *more coins*. By replacing that group of smaller coins in O with the single coin g_j, we could construct a new solution O' that is also valid for value V but has fewer coins than O. This contradicts the assumption that O was optimal.

4. Conclusion

Our assumption that an optimal solution exists with fewer coins than the greedy solution leads to a logical impossibility. Therefore, the initial assumption must be false. The greedy algorithm is optimal for this coin system. □

Proof 2: Proof by Induction

Proof by Induction. **Inductive Proposition.** Let $P(V)$ be the proposition that the greedy algorithm is optimal for making change for any value from 1 to V.

Base Case

For $V = 1, 2, 3, 4$, the greedy algorithm is trivially optimal (it uses only pennies). So $P(4)$ is true.

Inductive Hypothesis

Assume that $P(k)$ is true for all $k < V$. That is, for any value less than V, the greedy algorithm produces an optimal solution.

Inductive Step

We must show $P(V)$ is true. Let $d_{\max}$ be the largest denomination $\leq V$. The greedy algorithm, G, will choose $d_{\max}$ and then solve for the subproblem $V - d_{\max}$. The size of the greedy solution is $|G| = 1 + |G(V - d_{\max})|$.

By the inductive hypothesis, $|G(V - d_{\max})|$ is optimal for the value $V - d_{\max}$.

Now, consider an optimal solution O for value V. Let c_1, c_5, c_{10}, c_{25} be the number of pennies, nickels, dimes, and quarters in O, respectively. To prove optimality, we must show $|O| \geq |G|$. We can use the following properties of the US coin system:

- Any optimal solution for $V \geq 5$ must have $c_1 \leq 4$. If it had ≥ 5 pennies, they could be replaced by one nickel for a better solution.
- Any optimal solution for $V \geq 10$ must have $c_5 \leq 1$. If it had ≥ 2 nickels, they could be replaced by one dime for a better solution.
- Any optimal solution for $V \geq 25$ must have $c_{10} \leq 2$. If it had ≥ 3 dimes, they could be replaced by a quarter and a nickel, which is fewer coins.
- Any optimal solution for $V \geq 25$ must have $c_5 + 2c_{10} \leq 4$. The value of nickels and dimes must be less than 25 cents.

These properties imply that for any value V, an optimal solution must use coins in a way that is identical to the greedy strategy for the amount paid in coins smaller than the largest denomination. For example, if $V \geq 25$, an optimal solution cannot contain change worth 25 cents or more in dimes and nickels. Therefore, an optimal solution for V must use the coin $d_{\max}$, and then solve optimally for the remainder $V - d_{\max}$. This is exactly what the greedy algorithm does.

Since the greedy algorithm makes the same first choice as an optimal solution ($d_{\max}$), and then, by the inductive hypothesis, solves the subproblem optimally, the full solution must be optimal. Thus $P(V)$ is true.

Conclusion

By the principle of mathematical induction, the greedy algorithm is optimal for all values V. □

Proof 3: Direct Proof (Greedy Choice Property)

Direct Proof. We prove that at each step, the greedy choice is part of some optimal solution. This is known as the **greedy choice property**.

Let V be the value we need to make change for. Let $d_{\max}$ be the largest denomination such that $d_{\max} \leq V$. The greedy algorithm chooses $d_{\max}$ as its first coin. We must show that there exists an optimal solution for V that also includes at least one coin of denomination $d_{\max}$.

Let O be any optimal solution for V.

- **Case 1:** $V \in [1, 4]$. $d_{\max} = 1$. Any solution must use pennies, so the greedy choice is part of the optimal solution.
- **Case 2:** $V \in [5, 9]$. $d_{\max} = 5$. Assume an optimal solution O does not use a nickel. Then it must make the value V using only pennies. This would require at least 5 pennies. We can replace these 5 pennies with a single nickel, resulting in a solution with fewer coins. This contradicts the optimality of O. Therefore, O must contain a nickel.
- **Case 3:** $V \in [10, 24]$. $d_{\max} = 10$. Assume an optimal solution O does not use a dime. It must make the value V using nickels and pennies. To sum to at least 10, it must use at least two coins (e.g., two nickels). We can replace these two coins with a single dime, reducing the number of coins and contradicting the optimality of O. Thus, O must contain a dime.
- **Case 4:** $V \geq 25$. $d_{\max} = 25$. Assume an optimal solution O does not use a quarter. It must make the value V using dimes, nickels, and pennies. To sum to at least 25, it must use at least three coins (e.g., 10c, 10c, 5c). We can replace these three coins with a single quarter, reducing the number of coins and contradicting the optimality of O. Thus, O must contain a quarter.

In every case, the greedy first choice ($d_{\max}$) is part of some optimal solution. Let this optimal solution be $O^* = \{d_{\max}\} \cup O'$, where O' is an optimal solution for the remaining value $V' = V - d_{\max}$.

Since the greedy algorithm makes an optimal first choice and then optimally solves the remaining subproblem, its overall solution is optimal. This holds for each subsequent step. □

Question 3: Interval Scheduling

Problem Statement

Problem 3. *Given a set of n requests (intervals) $S = \{1, 2, \ldots, n\}$, where each request i has a **start time** s_i and a **finish time** f_i, find a subset of mutually compatible (non-overlapping) requests, $A \subseteq S$, such that the size of the subset, $|A|$, is maximized.*

Two requests, i and j, are considered **compatible** if their time intervals do not overlap, meaning either $f_i \leq s_j$ or $f_j \leq s_i$.

The Greedy Algorithm: Earliest Finish First

The greedy strategy is as follows:

1. Sort the requests in increasing order of their **finish times**. Let the sorted requests be $r_1, r_2, \ldots, r_n$.
2. Initialize an empty solution set, $A = \emptyset$.
3. Add the first request, r_1, to the solution set A.
4. Iterate from $j = 2$ to n. If request r_j is compatible with the most recently added request in A, add r_j to A.

Proof of Optimality

We will prove the optimality of the "Earliest Finish First" algorithm using three different approaches.

Theorem 4. *The "Earliest Finish First" greedy algorithm yields an optimal solution for the Interval Scheduling Problem.*

Proof 1: Proof by Induction

Proof by Induction. **Inductive Proposition.** Let $P(k)$ be the proposition that for any set of k requests, the greedy algorithm produces an optimal solution.

Base Case

For a set of size $k = 1$, the algorithm selects that single request. This is trivially an optimal solution containing one request. Thus, $P(1)$ is true.

Inductive Hypothesis

Assume that for all sets of requests of size $i < k$, the greedy algorithm produces an optimal solution. That is, assume $P(i)$ is true for all $i < k$.

Inductive Step

We must now prove that $P(k)$ is true for a set of k requests. Let the set of k requests, sorted by finish times, be $R = \{r_1, r_2, \ldots, r_k\}$.

1. **The Greedy Choice.** The greedy algorithm's first choice is request r_1, as it has the earliest finish time. The solution produced by the greedy algorithm is $G = \{r_1\} \cup G'$, where G' is the greedy solution for the subproblem $R' \subset R$ containing all requests from R that are compatible with r_1.
2. **The Greedy Choice Property.** We show that some optimal solution contains the greedy choice, r_1. Let $O = \{o_1, o_2, \ldots, o_m\}$ be any optimal solution for R, with its requests also sorted by finish times.
    - If $o_1 = r_1$, the optimal solution already contains our greedy choice.

- If $o_1 \neq r_1$, we construct a new solution $O' = \{r_1\} \cup \{o_2, \ldots, o_m\}$. Since r_1 is the request with the absolute earliest finish time in the entire set R, we know that $f(r_1) \leq f(o_1)$. Because all requests $\{o_2, \ldots, o_m\}$ in O were compatible with o_1 (meaning their start times were all $\geq f(o_1)$), they must also be compatible with r_1.

Therefore, O' is a valid, compatible set of requests. It has the same size as O, so it is also optimal. This proves that an optimal solution containing the greedy choice r_1 always exists.

3. **The Optimal Substructure.** Let this optimal solution containing r_1 be $O^* = \{r_1\} \cup O'$, where O' is an optimal solution for the subproblem R'. The size of this optimal solution is $|O^*| = 1 + |O'|$. The size of the subproblem R' is $|R'| = i$, where $i < k$.

   By our **inductive hypothesis**, the greedy algorithm finds an optimal solution for any problem of size $i < k$. Therefore, the greedy solution for the subproblem R', which we called G', must be optimal for R'. This means $|G'| = |O'|$.

   Comparing the sizes of the full solutions:

   - Size of Greedy Solution: $|G| = 1 + |G'|$
   - Size of Optimal Solution: $|O^*| = 1 + |O'|$

   Since $|G'| = |O'|$, it follows that $|G| = |O^*|$.

Thus, the greedy solution for the set of k requests is optimal.

Conclusion

By the principle of mathematical induction, the algorithm is optimal. □

Proof 2: Proof by Contradiction

Proof by Contradiction. **1. The Assumption**
Assume for the sake of contradiction that the greedy algorithm is **not** optimal. This means its solution, $G = \{g_1, g_2, \ldots, g_k\}$, is smaller than some optimal solution, $O = \{o_1, o_2, \ldots, o_m\}$, where $k < m$. Both sets of intervals are sorted by their finish times.
2. The Analysis
Since G and O are different, there must be a first index i where they differ, so $g_i \neq o_i$.

- For $i = 1$: The greedy algorithm selects g_1 because it has the earliest finish time of all intervals. The optimal solution selects o_1. By definition of the greedy choice, $f(g_1) \leq f(o_1)$. If they are not the same interval, we can create a new solution $O' = \{g_1, o_2, \ldots, o_m\}$. Since $f(g_1) \leq f(o_1)$ and $s(o_2) \geq f(o_1)$, it must be that $s(o_2) \geq f(g_1)$. So g_1 is compatible with o_2. O' is a valid solution of size m, and is therefore also optimal. This shows there is always an optimal solution that agrees with the first greedy choice.

- Let's now assume G and O are two solutions where the first element, $g_1 = o_1$, is the same. There must be a first index $j > 1$ where $g_j \neq o_j$.

Let's find the first index i where $g_i \neq o_i$. We know an optimal solution exists where $g_1 = o_1, g_2 = o_2, \ldots, g_{i-1} = o_{i-1}$.

The greedy algorithm selected g_i because it was the first available interval to start after g_{i-1} (and thus o_{i-1}) finished. The optimal solution selected o_i which also must start after o_{i-1} finished. By the greedy choice rule (earliest finish time), it must be that $f(g_i) \leq f(o_i)$.
3. Reaching the Contradiction
As before, we can construct a new optimal solution $O' = \{g_1, \ldots, g_{i-1}, g_i, o_{i+1}, \ldots, o_m\}$. This solution is valid because g_i is compatible with $g_{i-1} = o_{i-1}$. And since $f(g_i) \leq f(o_i)$ and $s(o_{i+1}) \geq f(o_i)$, it follows that $s(o_{i+1}) \geq f(g_i)$, so g_i is compatible with o_{i+1}.

We can repeat this argument for every index. For each i from 1 to k, we can show that there exists an optimal solution of the form $\{g_1, g_2, \ldots, g_i, \ldots\}$. This implies that the greedy solution G is a prefix of some

optimal solution O^*. But our initial assumption was that $|G| < |O^*|$, i.e., $k < m$. This means that after the last interval g_k is chosen, there is at least one more interval, o_{k+1}, in the optimal solution O^*. The interval o_{k+1} must be compatible with $o_k = g_k$. But the greedy algorithm only terminates when there are no more compatible intervals left. If o_{k+1} exists and is compatible with g_k, the greedy algorithm would have selected it (or another compatible interval). This is a contradiction.

4. Conclusion

The assumption that there is an optimal solution larger than the greedy solution leads to a contradiction. Thus, the greedy algorithm is optimal. □

Proof 3: Direct Proof ("Stays Ahead" Argument)

Direct Proof. This proof shows that the greedy algorithm "stays ahead" of any optimal solution. Let the greedy solution be $G = \{g_1, g_2, \ldots, g_k\}$ and any optimal solution be $O = \{o_1, o_2, \ldots, o_m\}$, both sorted by finish times. We will prove by induction that for all $i \leq k$, the i-th interval chosen by the greedy algorithm finishes no later than the i-th interval in the optimal solution.

Proposition. For all $i \in \{1, \ldots, k\}$, we have $f(g_i) \leq f(o_i)$.

Base Case ($i = 1$)

The greedy algorithm chooses g_1 because it has the earliest finish time among all intervals. The interval o_1 is just one of these intervals. Therefore, by definition of the greedy choice, $f(g_1) \leq f(o_1)$. The base case holds.

Inductive Step

Assume that for some $i-1 < k$, the proposition holds: $f(g_{i-1}) \leq f(o_{i-1})$. We must show that $f(g_i) \leq f(o_i)$.

- By definition of a valid schedule, the interval o_i must start after o_{i-1} has finished. So, $s(o_i) \geq f(o_{i-1})$.
- From our inductive hypothesis, we know $f(g_{i-1}) \leq f(o_{i-1})$.
- Combining these gives $s(o_i) \geq f(o_{i-1}) \geq f(g_{i-1})$.

This last inequality, $s(o_i) \geq f(g_{i-1})$, shows that the interval o_i is compatible with g_{i-1} and all previous intervals in the greedy solution. Therefore, o_i was one of the available candidates for the greedy algorithm to choose as its i-th interval.

The greedy algorithm chooses g_i from the set of all candidates compatible with g_{i-1} because it has the *earliest finish time*. Since o_i was one of these candidates, the chosen interval g_i must have a finish time that is less than or equal to o_i's finish time. That is, $f(g_i) \leq f(o_i)$. The induction holds.

Conclusion of Optimality

We have shown that $f(g_i) \leq f(o_i)$ for all $i \leq k$. Now we use this to prove optimality, again by contradiction. Assume that the optimal solution O has more intervals than the greedy solution G, so $m > k$. This would imply there is an interval o_{k+1} in the optimal solution. This interval must start after o_k finishes, i.e., $s(o_{k+1}) \geq f(o_k)$. From our "stays ahead" proof, we know that $f(g_k) \leq f(o_k)$. Therefore, $s(o_{k+1}) \geq f(g_k)$. This means that the interval o_{k+1} is compatible with the last greedy interval g_k (and all prior greedy intervals). However, the greedy algorithm terminated after selecting g_k. This means that there were no more intervals in the original set that were compatible with g_k. This is a direct contradiction to the existence of o_{k+1}. Therefore, the assumption that $m > k$ must be false. Since an optimal solution must be at least as large as any valid solution, we must have $m \geq k$. Combining $m \not> k$ and $m \geq k$ implies that $m = k$. The greedy solution has the same size as any optimal solution, and is therefore optimal. □

Question 4: Proving a Greedy Algorithm Incorrect

Problem 4. *Consider the **0/1 Knapsack Problem**: You have a knapsack with a maximum weight capacity W and a set of n items. Each item i has a weight w_i and a value v_i. You can either take an entire item or leave it behind; you cannot take fractions of items. The goal is to choose a subset of items that maximizes the total value without exceeding the weight capacity.*

A proposed greedy algorithm for this problem is as follows:

1. *Calculate the value-to-weight ratio (v_i/w_i) for each item.*
2. *Sort the items in descending order based on this ratio.*
3. *Iterate through the sorted items and add an item to the knapsack if it fits.*

***Your Task:** Prove that this greedy algorithm is **not optimal**. Provide a specific counterexample consisting of a weight capacity W and a set of items (with their weights and values). Show the result of the greedy algorithm and compare it to the true optimal solution for your example.*

Solution via Counterexample

We can prove the proposed greedy algorithm is not optimal by providing a counterexample where it fails to produce the best solution.

The Counterexample

Let the knapsack have a weight capacity of $\mathbf{W = 50}$.
Consider the following three items:

- **Item 1:** $w_1 = 10$, $v_1 = 60$ (Ratio: $60/10 = 6$)
- **Item 2:** $w_2 = 20$, $v_2 = 100$ (Ratio: $100/20 = 5$)
- **Item 3:** $w_3 = 30$, $v_3 = 120$ (Ratio: $120/30 = 4$)

Applying the Greedy Algorithm

The greedy algorithm sorts the items by their value-to-weight ratio in descending order: Item 1, Item 2, Item 3.

1. **Choose Item 1** (ratio 6). It fits in the knapsack.
   - Remaining capacity: $50 - 10 = 40$.
   - Current value: 60.
2. **Choose Item 2** (ratio 5). It fits in the knapsack.
   - Remaining capacity: $40 - 20 = 20$.
   - Current value: $60 + 100 = 160$.
3. **Consider Item 3** (ratio 4). Its weight is 30, which is greater than the remaining capacity of 20. We cannot add it.

The greedy algorithm terminates. The chosen items are {Item 1, Item 2}.

- **Greedy Solution Value: \$160**
- **Greedy Solution Weight: 30**

The Optimal Solution

A different choice of items yields a better result. Consider choosing {Item 2, Item 3}:

- Total weight: $w_2 + w_3 = 20 + 30 = 50$. This fits exactly in the knapsack.
- Total value: $v_2 + v_3 = 100 + 120 = 220$.

The optimal choice of items is {Item 2, Item 3}.

- **Optimal Solution Value: $220**
- **Optimal Solution Weight: 50**

Conclusion

The greedy algorithm produced a solution with a total value of $160, while the optimal solution has a total value of $220.

$$\$160 \text{ (Greedy)} < \$220 \text{ (Optimal)}$$

Since the greedy algorithm did not find the optimal solution for this instance, the algorithm is proven to be incorrect for the 0/1 Knapsack Problem.

Question 5: Non-Crossing Connections on Parallel Lines

Problem 5. *You have n red points on a line $y = 1$ and n blue points on a line $y = 0$. The x-coordinates of all $2n$ points are distinct. You must connect each red point to exactly one blue point with a straight line segment. Devise a greedy algorithm to find a pairing that ensures no two line segments cross. Prove that your algorithm is correct.*

Question 6: Pairing Points on a Line

Problem 6. *You are given $2n$ points on the real number line. The cost of pairing two points (p_i, p_j) is the distance $|p_i - p_j|$. Devise a greedy algorithm to partition the points into n pairs that minimizes the sum of the costs of all pairs. Prove that your algorithm is optimal.*

Question 7: Minimizing Lateness

Problem 7. *You have a single machine and a set of n jobs. Each job i requires processing time t_i and has a deadline d_i. The lateness of a job is defined as $L_i = \max(0, f_i - d_i)$, where f_i is its finish time. Devise a greedy algorithm to find a schedule that minimizes the **maximum lateness** over all jobs ($L_{\max} = \max_i L_i$). Prove that your algorithm is optimal.*

Question 8: Minimum Spanning Tree

Problem 8. *Given a connected, undirected, and weighted graph, a spanning tree is a subgraph that connects all the vertices together without any cycles. Devise a greedy algorithm to find a spanning tree with the minimum possible total edge weight. Prove that your algorithm is optimal.*

Question 9: The Fractional Knapsack

Problem 9. *You have a knapsack with a maximum weight capacity W and a set of items. Each item i has a weight w_i and a value v_i. You are allowed to take fractions of items. Devise a greedy algorithm to maximize the total value of items placed in the knapsack without exceeding the weight capacity W. Prove that your algorithm is optimal.*

Question 10: Djikstra for negative-weight edges

Problem 10. *Prove that Dijkstra's algorithm is not guaranteed to find the correct shortest paths in graphs containing negative edge weights.*

(a) *First, provide a rigorous theoretical proof. Your argument should not use a specific counterexample. Instead, focus on the core greedy principle or loop invariant of Dijkstra's algorithm. Explain what this principle is, why it holds true for graphs with non-negative edge weights, and precisely how the presence of a negative edge weight can violate this principle, thereby invalidating the algorithm's guarantee of correctness.*

(b) *Second, prove the assertion by providing a concrete counterexample. Devise a simple, weighted, directed graph with a designated source vertex that includes at least one negative edge. Then, demonstrate the algorithm's failure by:*

   (a) *Showing the step-by-step execution of Dijkstra's algorithm on your graph.*

   (b) *Stating the final (and incorrect) shortest path distances calculated by the algorithm.*

   (c) *Stating the true shortest path distances to show the discrepancy.*

Question 11: Maximum Spanning Trees

Problem 11. *A Maximum Spanning Tree is a spanning tree of a weighted, connected, undirected graph that has the largest possible total edge weight. Devise a simple transformation on the edge weights of the graph that would allow a standard Minimum Spanning Tree algorithm to find the Maximum Spanning Tree of the original graph. Prove that your transformation works.*