Greedy algortihm

Greedy Algorithms — Exam Notes

1. Fractional Knapsack

Problem:

- Knapsack capacity = WWW.
- NNN items, each with weight wi>0w_i > 0wi>0 and value vi>0v_i > 0vi>0.
- You may take **any fraction** of an item. Value and weight scale proportionally.
- Goal: maximize total value.

Solution (Greedy):

1. Compute value-to-weight ratio ri=vi/wir_i = v_i / w_iri=vi/wi.
2. Sort items by descending rir_iri.
3. Take items in that order until the knapsack is full.
    - If current item doesn't fit, take as much fraction as possible.

Correctness Proof (exchange argument):

- Suppose solution includes some of item iii (lower ratio) but not all of item jjj (higher ratio).
- Swapping a small weight δ\deltaδ of iii for the same δ\deltaδ of jjj increases total value (since vj/wj>vi/wiv_j/w_j > v_i/w_ivj/wj>vi/wi).
- Therefore an optimal solution must always take higher-ratio items first.

Complexity:

- Sort items: O(NlogN)O(N \log N)O(NlogN).

- Take items greedily: O(N)O(N)O(N).
- Total: O(NlogN)O(N \log N)O(NlogN).

Pseudo-code:

```
def frac_knapsack(W, items):
    # items = list of (weight, value)
    items.sort(key=lambda x: x[1]/x[0], reverse=True)  # sort by ratio
    total_value, weight = 0, 0
    for wi, vi in items:
        if weight + wi <= W:
            total_value += vi
            weight += wi
        else:
            remain = W - weight
            total_value += vi * (remain / wi)
            break
    return total_value
```

---

2. What is a Greedy Strategy?

- A **greedy strategy** makes a **locally optimal choice** at each step, hoping it leads to a global optimum.
- Needs **optimal substructure**: after making a greedy choice, the remainder of the problem is a smaller instance of the same type.
- Correctness often proven via:
    - **Exchange argument** (like in fractional knapsack).
    - **Proof by bubble sort** (for ordering problems).

Greedy is used for:

- Problems where simple “short-sighted” choices lead to global optimality.

- Faster and simpler than DP when it works (often $O(N \log N)$ vs DP's $O(NW)$, etc).

---

3. Activity Selection Problem

Problem:

- Set of activities $A = \{a_1, ..., a_n\}$, each interval $[s_i, f_i)$.
- Two activities are compatible if they don't overlap.
- Find the **maximum set of mutually compatible activities**.

Modeling:

- Subproblem: maximum compatible set from time t.
- Base case: if t is after all finish times, result is empty set.
- Recurrence: choose a next activity, then recurse after its finish time.

Greedy strategy:

- Always choose the activity that finishes earliest among those that start after current time.

Why correct?

- Suppose optimal picks activity a_i finishing later than a_j.
- Replacing a_i with earlier-finishing a_j leaves more room for future activities → never worse.
- Therefore the earliest-finishing activity is always safe to take.

Complexity:

- Sort activities by finish time: $O(N \log N)$.
- Scan through once to pick: $O(N)$.
- Total: $O(N \log N)$.

---

4. Task Scheduling Problem

Problem:

- Tasks S={(li,di)}S = \{(l_i, d_i)\}S={(li,di)} where lil_ili = length, did_idi = deadline.
- Start at time 0, tasks done sequentially.
- Completing task iii at time ttt gives reward di−td_i - tdi−t. (Positive if early, negative if late).
- Goal: maximize total reward.

Observation:

- Only ordering matters (no idle time is ever good).
- Consider two adjacent tasks sk=(lk,dk)s_k = (l_k,d_k)sk=(lk,dk) and sk+1=(lk+1,dk+1)s_{k+1} = (l_{k+1},d_{k+1})sk+1=(lk+1,dk+1).
- Swapping them changes reward by lk−lk+1l_k - l_{k+1}lk−lk+1.
- If lk>lk+1l_k > l_{k+1}lk>lk+1, reward increases by swapping.
- Therefore tasks must be sorted by **ascending length**.

Greedy strategy:

- Sort all tasks by lil_ili (shortest job first).

Complexity:

- Sorting: O(NlogN)O(N \log N)O(NlogN).
- Scheduling: O(N)O(N)O(N).

---

5. Proof by Bubble Sort

- Technique for proving greedy orderings.
- Idea: If optimal solution had two adjacent items in the wrong order, swapping them increases optimality.
- Keep swapping until sequence is sorted → proves the sorted order is optimal.
- Called “proof by bubble sort” because the argument resembles bubbling the elements into correct order.

✅ With this, you now have:

- Fractional Knapsack (greedy ratio strategy, correctness proof, complexity, code).
- Greedy strategies (definition, correctness proofs, uses).
- Activity Selection (earliest finish time rule).
- Task Scheduling (shortest job first, proven via bubble sort).
- Proof by bubble sort explained.

📘 Greedy Algorithms — Exam Notes

1. Fractional Knapsack

Problem:

- Knapsack capacity = WWW.
- NNN items, each with weight wi>0w_i > 0wi>0 and value vi>0v_i > 0vi>0.
- You may take **any fraction** of an item. Value and weight scale proportionally.
- Goal: maximize total value.

Solution (Greedy):

1. Compute value-to-weight ratio ri=vi/wir_i = v_i / w_iri=vi/wi.
2. Sort items by descending rir_iri.
3. Take items in that order until the knapsack is full.
    - If current item doesn't fit, take as much fraction as possible.

Correctness Proof (exchange argument):

- Suppose solution includes some of item iii (lower ratio) but not all of item jjj (higher ratio).

- Swapping a small weight δ\deltaδ of iii for the same δ\deltaδ of jjj increases total value (since vj/wj>vi/wiv_j/w_j > v_i/w_ivj/wj>vi/wi).
- Therefore an optimal solution must always take higher-ratio items first.

Complexity:

- Sort items: O(NlogN)O(N \log N)O(NlogN).
- Take items greedily: O(N)O(N)O(N).
- Total: O(NlogN)O(N \log N)O(NlogN).

Pseudo-code:

```
def frac_knapsack(W, items):
    # items = list of (weight, value)
    items.sort(key=lambda x: x[1]/x[0], reverse=True)  # sort by ratio
    total_value, weight = 0, 0
    for wi, vi in items:
        if weight + wi <= W:
            total_value += vi
            weight += wi
        else:
            remain = W - weight
            total_value += vi * (remain / wi)
            break
    return total_value
```

---

2. What is a Greedy Strategy?

- A **greedy strategy** makes a **locally optimal choice** at each step, hoping it leads to a global optimum.
- Needs **optimal substructure**: after making a greedy choice, the remainder of the problem is a smaller instance of the same type.
- Correctness often proven via:

- **Exchange argument** (like in fractional knapsack).
- **Proof by bubble sort** (for ordering problems).

Greedy is used for:

- Problems where simple "short-sighted" choices lead to global optimality.
- Faster and simpler than DP when it works (often $O(N \log N)$ vs DP's $O(NW)$, etc).

---

3. Activity Selection Problem

Problem:

- Set of activities $A = \{a_1, ..., a_n\}$, each interval $[s_i, f_i)$.
- Two activities are compatible if they don't overlap.
- Find the **maximum set of mutually compatible activities**.

Modeling:

- Subproblem: maximum compatible set from time t.
- Base case: if t is after all finish times, result is empty set.
- Recurrence: choose a next activity, then recurse after its finish time.

Greedy strategy:

- Always choose the activity that finishes earliest among those that start after current time.

Why correct?

- Suppose optimal picks activity a_i finishing later than a_j.
- Replacing a_i with earlier-finishing a_j leaves more room for future activities → never worse.
- Therefore the earliest-finishing activity is always safe to take.

Complexity:

- Sort activities by finish time: $O(N \log N)$.

- Scan through once to pick: O(N)O(N)O(N).
- Total: O(NlogN)O(N \log N)O(NlogN).

---

4. Task Scheduling Problem

Problem:

- Tasks S={(li,di)}S = \{(l_i, d_i)\}S={(li,di)} where lil_ili = length, did_idi = deadline.
- Start at time 0, tasks done sequentially.
- Completing task iii at time ttt gives reward di−td_i - tdi−t. (Positive if early, negative if late).
- Goal: maximize total reward.

Observation:

- Only ordering matters (no idle time is ever good).
- Consider two adjacent tasks sk=(lk,dk)s_k = (l_k,d_k)sk=(lk,dk) and sk+1=(lk+1,dk+1)s_{k+1} = (l_{k+1},d_{k+1})sk+1=(lk+1,dk+1).
- Swapping them changes reward by lk−lk+1l_k - l_{k+1}lk−lk+1.
- If lk>lk+1l_k > l_{k+1}lk>lk+1, reward increases by swapping.
- Therefore tasks must be sorted by **ascending length**.

Greedy strategy:

- Sort all tasks by lil_ili (shortest job first).

Complexity:

- Sorting: O(NlogN)O(N \log N)O(NlogN).
- Scheduling: O(N)O(N)O(N).

---

5. Proof by Bubble Sort

- Technique for proving greedy orderings.

- Idea: If optimal solution had two adjacent items in the wrong order, swapping them increases optimality.
- Keep swapping until sequence is sorted → proves the sorted order is optimal.
- Called “proof by bubble sort” because the argument resembles bubbling the elements into correct order.

✅ With this, you now have:

- Fractional Knapsack (greedy ratio strategy, correctness proof, complexity, code).
- Greedy strategies (definition, correctness proofs, uses).
- Activity Selection (earliest finish time rule).
- Task Scheduling (shortest job first, proven via bubble sort).
- Proof by bubble sort explained.