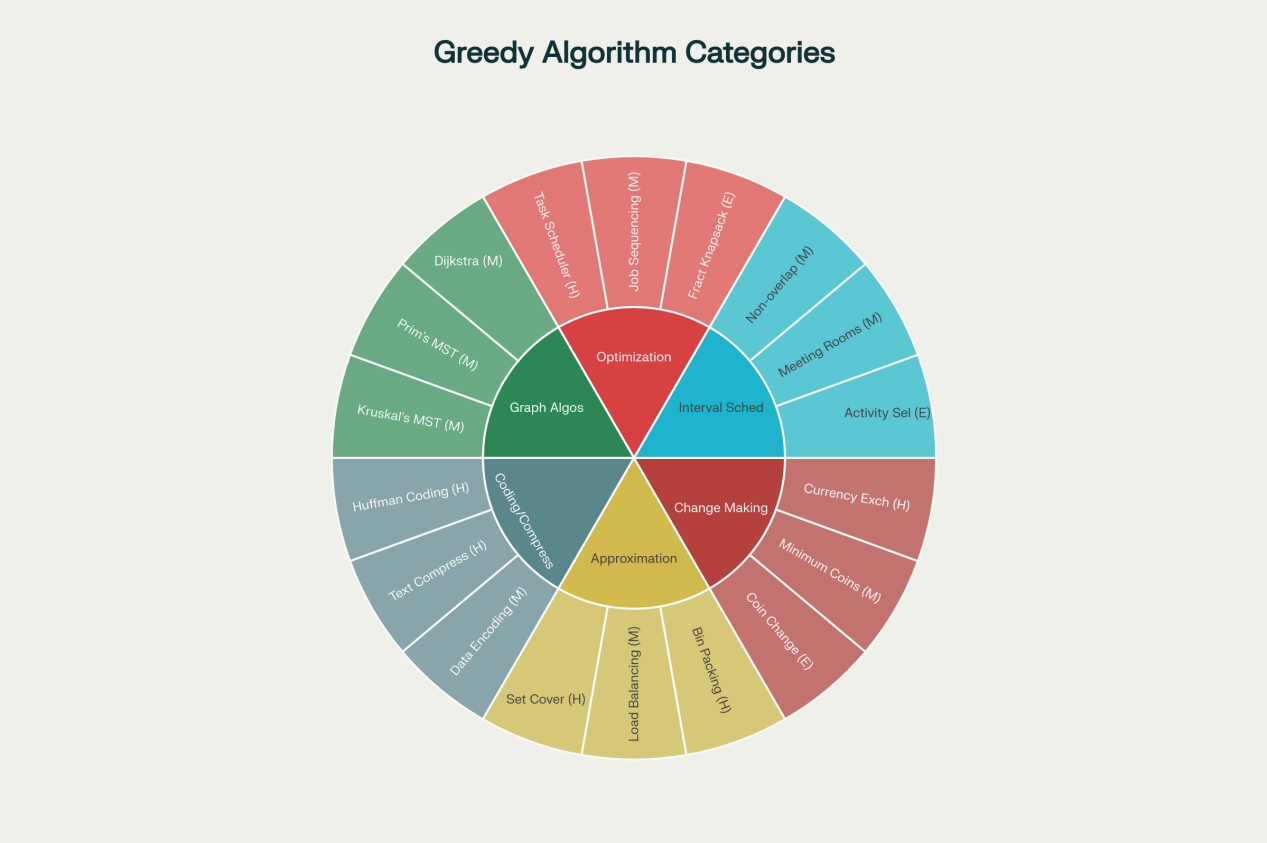
Complete Greedy DSA Handbook

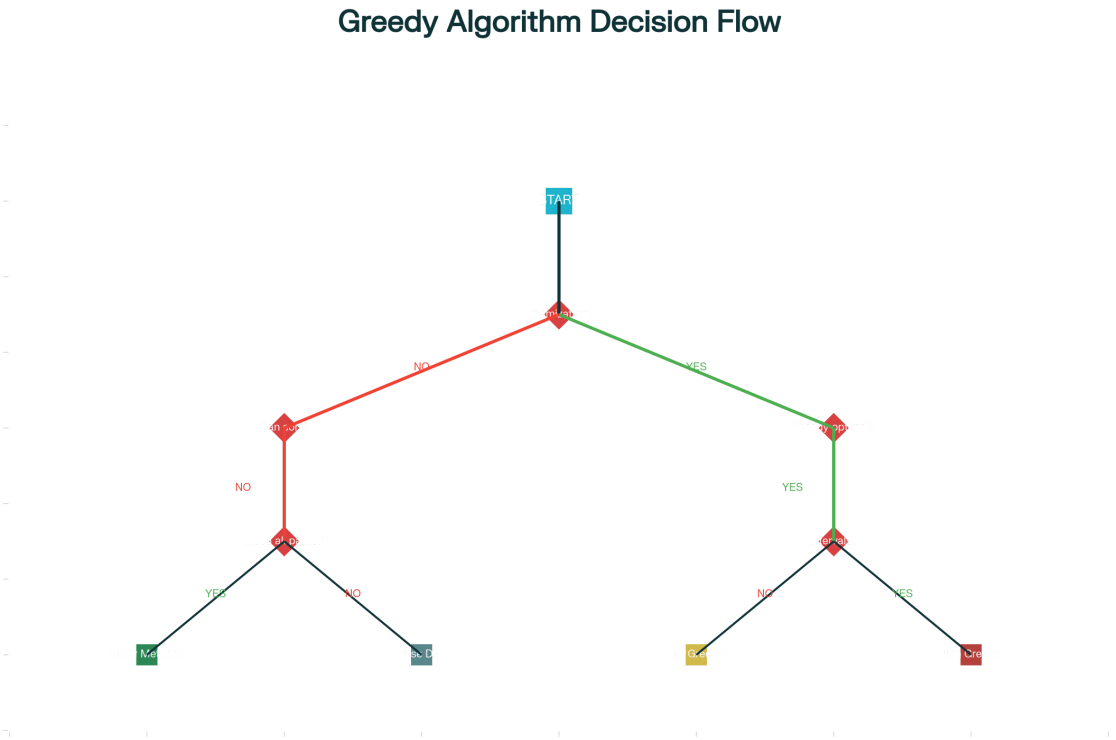
As an expert DSA tutor, I've created a comprehensive handbook to help you master **all types of greedy problems**. This systematic guide addresses your specific challenges with identifying greedy solutions and understanding different greedy strategies.



Comprehensive Overview of Greedy Algorithm Problem Categories

Understanding When to Use Greedy vs Other Approaches

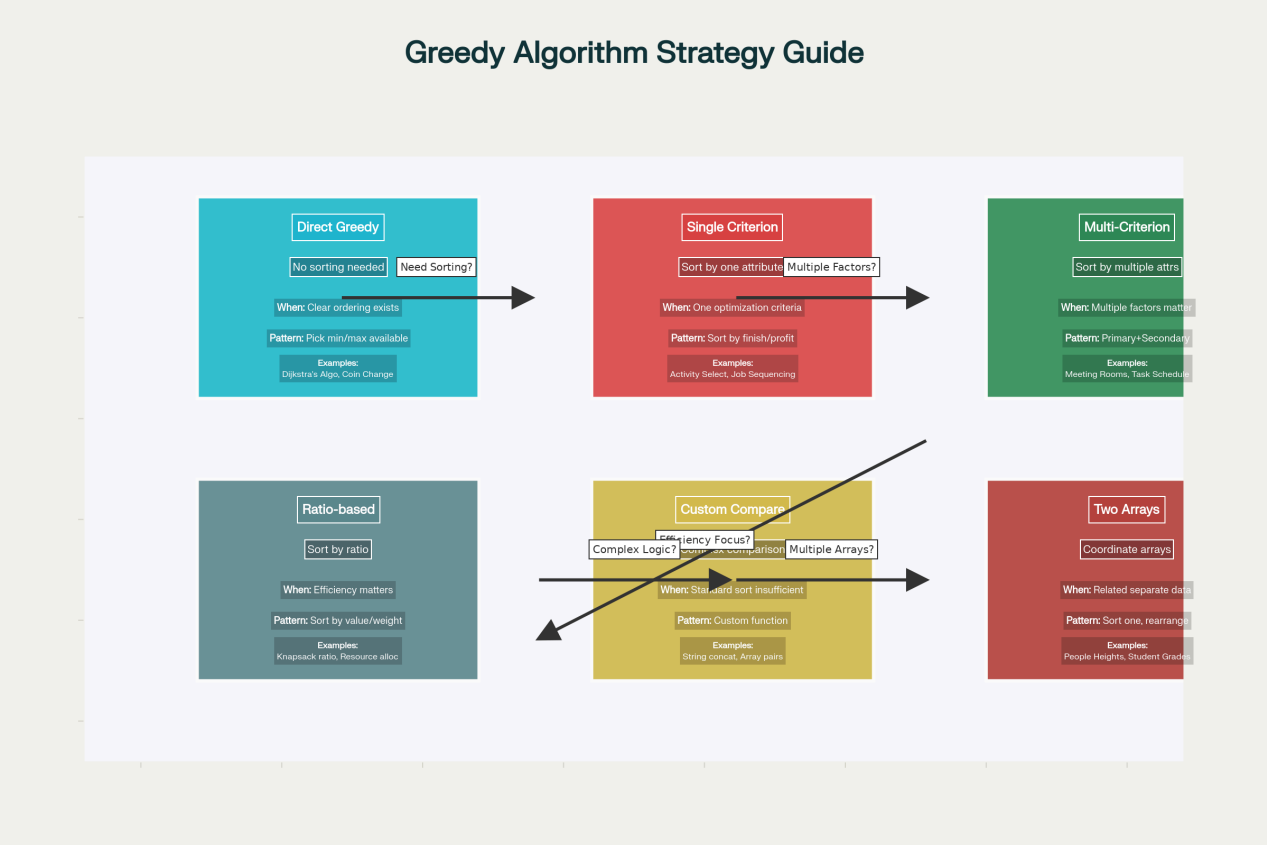
Your concern about not knowing when a problem requires greedy versus other approaches is very common. Here's a systematic decision framework:



Decision Flowchart: When to Use Greedy Algorithms vs Other Approaches

Types of Greedy Strategies and Sorting Approaches

You asked about different types of greedy algorithms (directed greedy, sorting-based, etc.). Here's a comprehensive breakdown:



Types of Greedy Strategies and Sorting Approaches

Complete Problem Categories with Real-World Applications

1. Interval Scheduling Problems

Real-world analogy: Meeting room booking system

Core greedy strategy: Sort by end time, always pick the activity that finishes earliest
Why greedy works:

- **Greedy choice property**: Picking earliest-ending activity leaves maximum room for future activities
- **Optimal substructure**: If activity A is in optimal solution, the remaining problem is optimal for the rest

Step-by-step approach:

```
def activity_selection(start, end):
    # Step 1: Combine and sort by end time
    activities = sorted(zip(start, end), key=lambda x: x[^1])

    # Step 2: Greedy selection
    selected = [activities[^0]]  # Always select first
    last_end = activities[^0][^1]

    for s, e in activities[1:]:
        if s >= last_end:  # No overlap (>= not >)
```

```
            selected.append((s, e))
            last_end = e

    return len(selected)
```

Complexity: O(n log n) - dominated by sorting
Common variations: Meeting Rooms I/II, Non-overlapping Intervals, Minimum Arrows

2. Fractional Knapsack Type

Real-world analogy: Resource allocation where you can take fractions (like liquids, grains)

Core greedy strategy: Sort by value-to-weight ratio (efficiency), take highest ratio first
Why greedy works: Taking most efficient items first always maximizes total value

Step-by-step approach:

```
def fractional_knapsack(values, weights, capacity):
    # Step 1: Calculate ratios and sort
    items = [(values[i]/weights[i], weights[i], values[i]) for i in range(len(values))]
    items.sort(reverse=True)  # Highest ratio first

    total_value = 0
    for ratio, weight, value in items:
        if capacity >= weight:
            # Take whole item
            capacity -= weight
            total_value += value
        else:
            # Take fraction
            total_value += ratio * capacity
            break

    return total_value
```

Complexity: O(n log n)

3. Job Sequencing with Deadlines

Real-world analogy: Project management with profit and deadlines

Core greedy strategy: Sort by profit (descending), schedule each job as late as possible before its deadline
Why greedy works: Higher profit jobs should be prioritized, and scheduling late maximizes flexibility

Step-by-step approach:

```
def job_sequencing(jobs):  # jobs = [(profit, deadline, id)]
    # Step 1: Sort by profit (highest first)
    jobs.sort(key=lambda x: x[^0], reverse=True)
```

```
    # Step 2: Create timeline slots
    max_deadline = max(job[^1] for job in jobs)
    timeline = [False] * (max_deadline + 1)

    total_profit = 0
    selected_jobs = []

    # Step 3: Greedily assign jobs
    for profit, deadline, job_id in jobs:
        # Find latest available slot before deadline
        for slot in range(min(deadline, max_deadline), 0, -1):
            if not timeline[slot]:
                timeline[slot] = True
                total_profit += profit
                selected_jobs.append(job_id)
                break

    return total_profit, selected_jobs
```

Complexity: O(n^2) - for each job, search for available slot

4. Graph Algorithms

Minimum Spanning Tree - Prim's Algorithm

Real-world analogy: Connecting cities with minimum total cable cost

Core greedy strategy: Always add minimum weight edge that connects a new vertex to the growing tree

```
def prim_mst(graph):
    import heapq
    visited = set()
    min_heap = [(0, 0)]  # (weight, vertex)
    mst_cost = 0

    while min_heap:
        weight, u = heapq.heappop(min_heap)

        if u in visited:
            continue

        visited.add(u)
        mst_cost += weight

        for v, w in graph[u]:
            if v not in visited:
                heapq.heappush(min_heap, (w, v))

    return mst_cost
```

Dijkstra's Shortest Path

Real-world analogy: GPS navigation finding shortest route

Core greedy strategy: Always visit the nearest unvisited vertex

```
def dijkstra(graph, start):
    import heapq
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_dist, current = heapq.heappop(pq)

        if current_dist > distances[current]:
            continue

        for neighbor, weight in graph[current]:
            distance = current_dist + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances
```

5. Huffman Coding

Real-world analogy: Text compression for efficient storage

Core greedy strategy: Always merge two nodes with smallest frequencies

```
def huffman_coding(frequencies):
    import heapq
    heap = [[freq, char] for char, freq in frequencies.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)

        merged = [left[^0] + right[^0], left, right]
        heapq.heappush(heap, merged)

    return heap[^0]  # Huffman tree root
```

6. Coin Change (Canonical Systems Only)

Real-world analogy: Making change with fewest coins

Core greedy strategy: Always use largest denomination possible
⚠ **Warning**: Only works for canonical coin systems (like US currency: 1, 5, 10, 25, 100)

```
def coin_change_greedy(coins, amount):
    coins.sort(reverse=True)  # Largest first
    result = []

    for coin in coins:
        count = amount // coin
        result.extend([coin] * count)
        amount %= coin

    return result if amount == 0 else None
```

When it fails: Non-canonical systems like for amount 6 [1] [2] [3]

- Greedy: 4 + 1 + 1 = 3 coins
- Optimal: 3 + 3 = 2 coins

7. Approximation Algorithms

Set Cover Problem

Real-world analogy: Selecting minimum radio stations to cover all cities

Core greedy strategy: Always pick the set that covers the most uncovered elements

```
def set_cover_greedy(universe, sets):
    uncovered = set(universe)
    selected_sets = []

    while uncovered:
        best_set = None
        best_coverage = 0

        for i, s in enumerate(sets):
            coverage = len(uncovered & set(s))
            if coverage > best_coverage:
                best_coverage = coverage
                best_set = i

        if best_set is not None:
            selected_sets.append(best_set)
            uncovered -= set(sets[best_set])

    return selected_sets
```

Approximation factor: ln(n) where n is universe size

Common Variations and Pitfalls

Pitfall 1: Wrong Sorting Criteria

- **Activity Selection**: ✗ Sort by duration → ✓ Sort by end time
- **Fractional Knapsack**: ✗ Sort by value → ✓ Sort by value/weight ratio
- **Job Sequencing**: ✗ Sort by deadline → ✓ Sort by profit

Pitfall 2: Boundary Conditions

- Use `>=` not `>` for interval comparisons (meeting ending at time t, next can start at time t)
- Handle empty inputs and single elements
- Check for integer overflow in profit calculations

Pitfall 3: Greedy Doesn't Always Work

- **0/1 Knapsack**: Needs dynamic programming
- **Longest Common Subsequence**: Needs DP
- **Traveling Salesman**: NP-hard, needs approximation or exact algorithms

Advanced Tricks and Tips

What to Memorize by Heart

Essential Templates (Practice Daily Until Automatic):

1. **Activity Selection Template**
2. **Fractional Knapsack Template**
3. **Job Sequencing Template**
4. **Dijkstra's Algorithm**
5. **Prim's MST Algorithm**

Sorting Strategy Mnemonics:

- **Intervals** → Sort by **END** time (earlier finish = more room)
- **Ratios** → Sort by **VALUE/WEIGHT** (efficiency first)
- **Jobs** → Sort by **PROFIT** (money talks)
- **Graphs** → Use **priority queues** (always pick best)

Problem Recognition Keywords:

- **"Maximum/Minimum"** + **"Selection"** → Likely greedy
- **"Schedule"** + **"Optimal"** → Check for interval problems
- **"Ratio/Efficiency"** → Fractional knapsack pattern
- **"Shortest path/MST"** → Standard graph algorithms

Example Problem Progression (Easy → Hard)

Easy (Start Here)

1. **LeetCode 455 - Assign Cookies**: Two arrays, sort both
2. **LeetCode 860 - Lemonade Change**: Direct greedy, no sorting needed
3. **LeetCode 1221 - Split Balanced Strings**: Simple counting greedy

Medium

1. **LeetCode 435 - Non-overlapping Intervals**: Classic interval scheduling
2. **LeetCode 452 - Minimum Arrows**: Interval processing with slight twist
3. **LeetCode 621 - Task Scheduler**: Frequency-based greedy with constraints

Hard

1. **LeetCode 1520 - Maximum Non-Overlapping Substrings**: Complex interval scheduling
2. **LeetCode 1675 - Minimize Deviation**: Priority queue with transformations

Competitive Programming

- **Codeforces**: Search problems tagged "greedy" with difficulty 800-1200 (beginner)
- **HackerRank**: Complete "Greedy Algorithms" section in Interview Preparation Kit
- **AtCoder**: Beginner Contest greedy problems

Study Schedule

Week 1: Master the three core templates (Activity Selection, Fractional Knapsack, Job Sequencing)
Week 2: Learn graph algorithms (Dijkstra, Prim, Kruskal)
Week 3: Practice medium-level problems and learn to identify patterns
Week 4: Solve contest problems and debug common mistakes

Remember: **Greedy algorithms are about pattern recognition and intuition**. The more you practice identifying when local optimal choices lead to global optimal solutions, the better you'll become at applying greedy strategies effectively!

⁂

1. https://drpress.org/ojs/index.php/HBEM/article/view/16401
2. https://aapm.onlinelibrary.wiley.com/doi/10.1002/mp.17624
3. http://research.ijcaonline.org/volume90/number17/pxc3894715.pdf
4. https://www.nature.com/articles/s41467-019-09115-y
5. https://www.cs.williams.edu/~shikha/teaching/spring20/cs256/lectures/Lecture09.pdf
6. https://www.geeksforgeeks.org/dsa/huffman-coding-greedy-algo-3/
7. https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
8. https://en.wikipedia.org/wiki/Dijkstra's_algorithm
9. https://www.programiz.com/dsa/huffman-coding
10. https://www.scaler.in/prims-and-kruskal-algorithm/
11. https://www.geeksforgeeks.org/dsa/is-dijkstra-a-greedy-algorithm/
12. https://www.w3schools.com/dsa/dsa_ref_huffman_coding.php
13. https://byjus.com/gate/difference-between-prims-and-kruskal-algorithum/
14. https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php
15. https://arxiv.org/pdf/2101.09700.pdf
16. https://www.youtube.com/watch?v=co4_ahEDCho
17. https://www.youtube.com/watch?v=4ZlRH0eK-qQ
18. https://www.shiksha.com/online-courses/articles/understanding-dijkstras-algorithm/
19. https://en.wikipedia.org/wiki/Huffman_coding
20. https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/
21. https://www.geeksforgeeks.org/dsa/dijkstras-shortest-path-algorithm-greedy-algo-7/
22. https://www.youtube.com/watch?v=uDS8AkTAcIU
23. https://en.wikipedia.org/wiki/Kruskal's_algorithm
24. https://arxiv.org/abs/2411.18137
25. https://www.semanticscholar.org/paper/d536f05e313d0fd8060e50ec926094981b2ef937
26. http://arxiv.org/pdf/2408.08935.pdf
27. http://www.ijcaonline.org/archives/volume178/number15/mehmood-2019-ijca-918787.pdf
28. https://www.semanticscholar.org/paper/cb940dcd80288a252f82b4cc41591409a09ab8f7
29. http://ieeexplore.ieee.org/document/5254395/
30. https://dl.acm.org/doi/10.1145/2498328.2500095
31. https://ieeexplore.ieee.org/document/9210380/
32. https://www.semanticscholar.org/paper/8cbea0052eb3e36d4be6d8edd853107869048e2f
33. https://scholarcommons.usf.edu/ujmm/vol11/iss1/5/
34. https://arxiv.org/abs/2407.20422
35. https://arxiv.org/pdf/0809.0400.pdf
36. http://arxiv.org/pdf/0801.0120v2.pdf
37. http://arxiv.org/pdf/2410.09434.pdf
38. https://arxiv.org/pdf/2111.12392.pdf

39. http://arxiv.org/pdf/2407.19137.pdf
40. https://arxiv.org/pdf/2203.11457.pdf
41. https://arxiv.org/pdf/2208.14700.pdf
42. https://arxiv.org/pdf/2412.10884.pdf
43. https://cyberenlightener.com/?page_id=222
44. https://spectra.mathpix.com/article/2022.03.00818/approximation-algorithms-load-balancing-and-bin-packing
45. https://courses.grainger.illinois.edu/cs583/sp2018/Notes/covering.pdf
46. https://www.baeldung.com/cs/min-number-of-coins-algorithm
47. https://thesai.org/Publications/ViewPaper?Volume=5&Issue=2&Code=IJACSA&SerialNo=9
48. http://arxiv.org/pdf/2502.13432.pdf
49. https://www.geeksforgeeks.org/dsa/greedy-approximate-algorithm-for-set-cover-problem/
50. https://www.log2base2.com/algorithms/greedy/greedy-algorithm.html
51. https://otfried.org/courses/cs500/slides-approx.pdf
52. https://www.cs.williams.edu/~shikha/teaching/spring20/cs256/lectures/Lecture31.pdf
53. https://courses.grainger.illinois.edu/cs598csc/sp2009/lectures/lecture_5.pdf
54. https://www.cs.dartmouth.edu/~deepc/LecNotes/Appx/1. Greedy Algorithm for Set Cover.pdf
55. https://www.reddit.com/r/algorithms/comments/pbq3zw/coin_change_problem_dynamic_vs_greedy_approach/
56. https://www.cs.williams.edu/~shikha/teaching/fall19/cs256/lectures/Lecture31.pdf
57. https://www.cs.jhu.edu/~mdinitz/classes/ApproxAlgorithms/Spring2019/Lectures/lecture3.pdf
58. https://www.geeksforgeeks.org/dsa/greedy-algorithm-to-find-minimum-number-of-coins/
59. https://arxiv.org/pdf/2202.10267.pdf
60. https://www.cs.princeton.edu/courses/archive/spr05/cos423/lectures/11approx-alg.pdf
61. https://www.tutorialspoint.com/data_structures_algorithms/dsa_set_cover_problem.htm
62. https://leetcode.com/problems/coin-change/
63. https://web.stanford.edu/class/msande319/Approximation Algorithm/lec1.pdf
64. https://www.semanticscholar.org/paper/7fb66090fe4b7e9a56a91bf977e6f40b49d569ac
65. https://dl.acm.org/doi/10.1145/3637528.3671827
66. https://ieeexplore.ieee.org/document/8476612/
67. https://www.mdpi.com/1999-4893/17/11/478
68. https://www.semanticscholar.org/paper/f1032737f72de19c4f49c014c26c57b25b53ce9d
69. https://www.mathnet.ru/eng/vuu904
70. http://arxiv.org/pdf/2405.16609.pdf
71. https://moitvivt.ru/ru/journal/pdf?id=1508
72. https://www.mdpi.com/2076-3417/12/4/1965
73. https://ieeexplore.ieee.org/document/9662236/
74. https://jurnal.unimed.ac.id/2012/index.php/cess/article/view/30625
75. http://arxiv.org/pdf/2311.14021.pdf

76. http://arxiv.org/pdf/1907.09064.pdf
77. https://arxiv.org/pdf/1308.3732.pdf
78. http://arxiv.org/pdf/0802.2184.pdf
79. http://arxiv.org/pdf/2407.20422.pdf
80. https://arxiv.org/pdf/1907.08142.pdf
81. http://arxiv.org/pdf/1801.07413.pdf
82. https://arxiv.org/pdf/2405.20939.pdf
83. http://arxiv.org/pdf/2001.09103.pdf
84. https://www.reddit.com/r/codeforces/comments/1fyj9c7/is_there_a_problemset_for_greedy_problems/
85. https://www.youtube.com/watch?v=b1TgpWPNEs4
86. https://www.youtube.com/watch?v=-WTslqPbj7I
87. https://www.youtube.com/watch?v=cPYk--7AC68
88. https://prepinsta.com/hackerrank-technical-section-questions__trashed/greedy-alogorithm/
89. https://leetcode.com/problem-list/greedy/
90. https://www.youtube.com/watch?v=gib-yC5NmuM
91. https://leetcode.com/discuss/interview-question/3972722/Top-Greedy-Questions-helpful-for-OA-and-Interviews
92. https://arxiv.org/pdf/2009.13998.pdf
93. https://www.hackerrank.com/interview/interview-preparation-kit/greedy-algorithms/challenges
94. https://leetcode.com/discuss/general-discussion/669996/greedy-for-beginners-problems-sample-solutions
95. https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/practice-problems/
96. https://leetcode.com/discuss/study-guide/5330283/Mastering-Greedy-Algorithms-with-LeetCode-Problems./
97. https://www.hackerrank.com/blog/dynamic-programming-definition-questions/
98. https://leetcode.com/discuss/general-discussion/1061059/ABCs-of-Greedy
99. https://www.hackerrank.com/domains/algorithms
100. https://leetcode.com/discuss/interview-question/5915635/DP-or-Greedy-or-String-or-Graph-or-Tree-or-Binary-Search-or-Sliding-Window-Topic-Wise/
101. https://www.hackerrank.com/domains/algorithms/greedy/difficulty:true/page:3
102. https://seanprashad.com/leetcode-patterns/
103. https://arxiv.org/pdf/2110.04719.pdf
104. https://www.hackerrank.com/challenges/greedy-florist/problem
105. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/c4f59d27c0e91e36a9123a3ae38704b8/a1c3c1a5-55fc-44c6-8f80-fd112e4ac0dd/c628e4dc.py
106. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/c4f59d27c0e91e36a9123a3ae38704b8/cac6445e-9d70-40b9-9fb9-641386e570fe/6165c41d.md
107. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/c4f59d27c0e91e36a9123a3ae38704b8/3063b13f-62af-4280-abf7-7e84c18b32de/bf1755c4.py

108. https://ppl-ai-code-interpreter-files.s3.amazonaws.com/web/direct-files/c4f59d27c0e91e36a9123a3ae38704b8/be08ff58-7561-4cb2-b1fc-e23e52ee7bd0/66780b83.py
109. https://www.shs-conferences.org/articles/shsconf/pdf/2022/14/shsconf_stehf2022_03009.pdf
110. https://en.wikipedia.org/wiki/Greedy_algorithm
111. https://brilliant.org/wiki/greedy-algorithm/
112. https://www.geeksforgeeks.org/dsa/greedy-algorithms-general-structure-and-applications/
113. https://www.w3schools.com/dsa/dsa_ref_greedy.php
114. https://javascript.plainenglish.io/i-found-these-10-patterns-to-solve-greedy-problems-faster-after-grinding-1000-leetcode-problems-4a05bf7dfb3f
115. https://www.sciencedirect.com/science/article/pii/S0167642303000340
116. https://www.upgrad.com/blog/greedy-algorithm/
117. https://www.codechef.com/practice/greedy-algorithms
118. https://www.codecademy.com/article/greedy-algorithm-explained
119. https://www.geeksforgeeks.org/dsa/greedy-algorithms/
120. https://www.jmlr.org/papers/volume4/mannor03a/mannor03a.pdf
121. https://www.jntua.ac.in/gate-online-classes/registration/downloads/material/a159239288590.pdf
122. https://www.khoury.northeastern.edu/home/vip/teach/Algorithms/5_greedy/notes/greedy.pdf
123. https://www.geeksforgeeks.org/dsa/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/
124. https://www.ewadirect.com/proceedings/ace/article/view/16674
125. https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/
126. https://www.sciencedirect.com/science/article/pii/S0167642303000340/pdf?md5=1d4a20a155eb65ba08086309bcf7921f&pid=1-s2.0-S0167642303000340-main.pdf&_valck=1
127. https://www.wscubetech.com/resources/dsa/greedy-algorithms
128. https://www.semanticscholar.org/paper/bc498529573226bbf84935ffeb00e9aefb418a6b
129. https://ieeexplore.ieee.org/document/11089084/
130. https://www.ewadirect.com/proceedings/ace/article/view/13702
131. https://www.nature.com/articles/s41598-023-47729-x
132. https://ieeexplore.ieee.org/document/9835149/
133. https://ieeexplore.ieee.org/document/10121391/
134. https://www.techscience.com/cmc/v82n1/59254
135. https://ieeexplore.ieee.org/document/10863331/
136. https://ieeexplore.ieee.org/document/10152800/
137. https://ieeexplore.ieee.org/document/10566935/
138. https://www.ewadirect.com/proceedings/tns/article/view/7969
139. http://arxiv.org/pdf/2501.09091.pdf
140. https://journals.sagepub.com/doi/10.1177/10591478251359808
141. http://arxiv.org/pdf/2407.20941.pdf
142. https://arxiv.org/pdf/2208.06815.pdf

143. http://arxiv.org/pdf/2303.06127.pdf
144. https://arxiv.org/abs/2307.00949
145. http://arxiv.org/pdf/1805.05436.pdf
146. http://arxiv.org/pdf/2404.11879.pdf
147. https://www.degruyter.com/document/doi/10.1515/math-2018-0026/html
148. http://thesai.org/Downloads/Volume13No1/Paper_46-A_Greedy_based_Algorithm_in_Optimizing_Student's.pdf
149. https://pmc.ncbi.nlm.nih.gov/articles/PMC11031591/
150. https://en.wikipedia.org/wiki/Activity_selection_problem
151. https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12708/2684033/Text-data-based-casual-analysis-of-initial-delay-of-high/10.1117/12.2684033.full
152. https://www.boardinfinity.com/blog/fractional-knapsack/
153. https://www.slideshare.net/slideshow/job-sequencing-with-deadlines/242580435
154. https://www.tutorialspoint.com/activity-selection-problem
155. https://www.slideshare.net/slideshow/fractional-knapsack-problem/217153823
156. https://www.scaler.com/topics/job-sequencing-with-deadlines/
157. https://www.educative.io/answers/what-is-the-activity-selection-problem
158. https://www.scaler.in/fractional-knapsack-problem/
159. https://byjus.com/gate/job-sequencing-with-deadlines/
160. https://www.geeksforgeeks.org/dsa/activity-selection-problem-greedy-algo-1/
161. https://www.tutorialspoint.com/data_structures_algorithms/fractional_knapsack_problem.htm
162. https://onepetro.org/SPEATCE/proceedings/18ATCE/18ATCE/D021S011R006/213690
163. https://www.geeksforgeeks.org/dsa/job-sequencing-problem/
164. https://www.studytonight.com/data-structures/activity-selection-problem
165. https://www.biyanicolleges.org/knapsack-problem-using-greedy-method/
166. http://ggn.dronacharya.info/CSE2Dept/Downloads/QuestionBank/Even/VI sem/ADA/Section-B/job-scheduling1.pdf
167. https://heycoach.in/blog/activity-selection-problem-proof/
168. https://www.geeksforgeeks.org/dsa/fractional-knapsack-problem/
169. https://www.tutorialspoint.com/data_structures_algorithms/job_sequencing_with_deadline.htm
170. https://www.geeksforgeeks.org/problems/activity-selection-1587115620/1
171. https://takeuforward.org/data-structure/fractional-knapsack-problem-greedy-approach/
172. https://ijournals.in/wp-content/uploads/2020/09/IJSHRE-8905-Sunmin-Lee-compressed.pdf
173. https://www.hindawi.com/journals/complexity/2023/9567183/
174. https://journal.yasib.com/index.php/enigma/article/view/24
175. https://www.mdpi.com/1099-4300/24/10/1447
176. https://ieeexplore.ieee.org/document/10125304/
177. https://www.semanticscholar.org/paper/677c944512c93a1b94d102b28849e8d6ea73b698
178. https://arxiv.org/abs/2210.04013

179. http://ma.fme.vutbr.cz/archiv/9_2/ma_9_2_foldes_final.pdf
180. https://www.semanticscholar.org/paper/f45176dc7deb50df9176ea7bc9845ef698b10409
181. https://dl.acm.org/doi/10.1145/2445196.2445458
182. http://ieeexplore.ieee.org/document/149517/
183. https://arxiv.org/pdf/2210.04013.pdf
184. https://link.springer.com/10.1007/s10462-021-10073-5
185. http://arxiv.org/pdf/2405.16753.pdf
186. http://arxiv.org/pdf/0906.2466.pdf
187. https://www.mdpi.com/1099-4300/24/10/1447/pdf?version=1665485370
188. https://arxiv.org/abs/1901.11343v4
189. https://arxiv.org/pdf/1907.07216.pdf
190. http://arxiv.org/pdf/1810.12861.pdf
191. https://pmc.ncbi.nlm.nih.gov/articles/PMC9602054/
192. https://arxiv.org/abs/2503.11107
193. https://www.cs.miami.edu/home/odelia/teaching/csc317.sp15/syllabus/Algorithms8bClass_greedy.pdf
194. https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/