Module 5 of 12

Greedy algorithms and exchange proofs

Sort, sweep, commit: activity selection by earliest finish, event sweeps for platforms and rooms, ordering by an exchange argument — and the counterexamples that send a problem to DP instead.

A greedy algorithm makes the locally best choice and never revisits it, which usually means one sort and one sweep — the fastest solution a problem can have. It is also the easiest to get wrong: "earliest start", "shortest first" and "largest coin first" all sound right and all fail. This module teaches the habit that separates a lucky greedy from a correct one: guess a rule, hunt for a small counterexample, then prove it with an **exchange argument** — swap one choice of an optimal answer for the greedy's and show nothing got worse.

The lesson plays the exchange argument out on activity selection, frame by frame, then covers the rules that fail (and why coin change needs DP) and the sweep family: events sorted with a deliberate tie order. The exercises are activity selection and minimum platforms at 2 500 intervals with times up to 10^9. The timed checkpoint orders tasks to maximise deadline minus finish time, where the proof decides the sort key and the total needs `long long`.

CONTENTS

1	Greedy algorithms — choose, then prove by exchange	14 min
✓	Checkpoint — Greedy algorithms and exchange proofs	45 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 14 MIN

Greedy algorithms — choose, then prove by exchange

A greedy algorithm makes the choice that looks best right now and never takes it back. It is the fastest kind of solution — usually a sort and one sweep — and the riskiest, because a plausible rule is often wrong. In a contest you want both halves: the rule, and a reason to believe it.

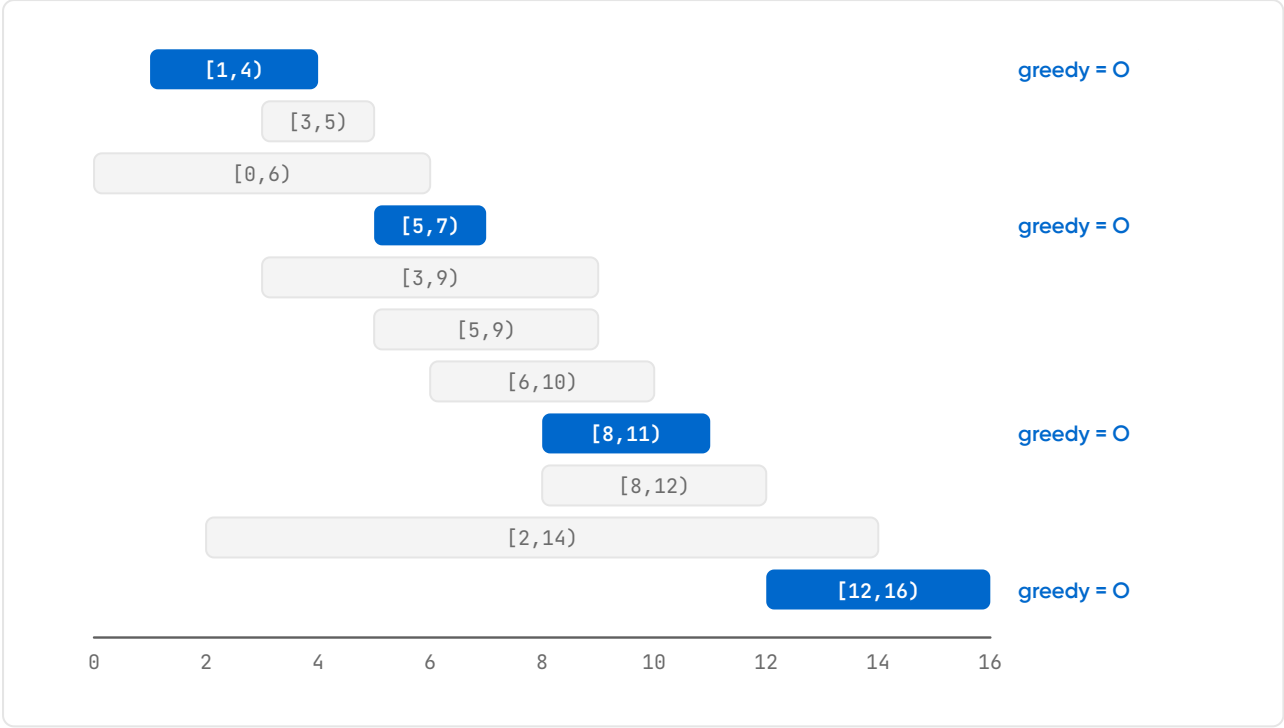
The recipe

1. **Guess a rule** for the next choice: earliest end, smallest weight, largest ratio.
2. **Hunt for a counterexample** on tiny inputs — on paper, or with a brute force over every subset.
3. **Prove it with an exchange**: take any optimal answer, swap one of its choices for the greedy's, and show nothing got worse.
4. **Implement**: sort by the rule, sweep once. O(n log n).

Activity selection, proved

Choose the most activities `[start, end)` with no two overlapping. The rule: **always take the one that finishes first** among those starting after the last one taken.

Why earliest finish is optimal: exchange an optimum into the greedy



1. 11 activities, each [start, end), drawn one per lane in order of finishing time. Pick as many as possible with no two overlapping; one may start exactly when another ends.
2. The greedy: always take the activity that finishes first among those that start after the last one taken. It takes [1,4), [5,7), [8,11), [12,16): 4 activities.
3. Is 4 the best? Take any optimal schedule O, say [3,5), [5,7), [8,12), [12,16) — also 4, sharing only 2 of its activities with the greedy's.
4. Swap O's activity 1, [3,5), for the greedy's [1,4). The greedy's ends at 4, no later than 5, so it still finishes before [5,7) starts. Still valid, still 4.
5. Swap O's activity 3, [8,12), for the greedy's [8,11). The greedy's ends at 11, no later than 12, so it still finishes before [12,16) starts. Still valid, still 4.
6. After the swaps O is exactly the greedy's schedule and never lost an activity. Since O was any optimum, the greedy's 4 is optimal. That is an exchange argument.

Why the swap is safe: the greedy's first pick ends no later than *any* activity, so putting it in place of O's first one cannot collide with O's second. Repeat for each position and O becomes the greedy's answer without shrinking.

```
sort(iv.begin(), iv.end(), [](auto& a, auto& b) { return a.second < b.second;
ll freeAt = LLONG_MIN;
int taken = 0;
for (auto [s, e] : iv)
    if (s ⩾ freeAt) { ++taken; freeAt = e; }
```

Rules that look right and are not

- **Earliest start**: one long activity starting at 0 blocks everything.
- **Shortest first**: a short activity overlapping two compatible ones costs you one.
- **Largest coin first**: coins {1, 3, 4}, target 6 → greedy 4+1+1, best 3+3.

The fix for coin change and 0/1 knapsack is dynamic programming. The fractional knapsack *is* greedy (best value per kilogram first).

The sweep family

Many interval problems are one sort plus one sweep over **events**:

- **Minimum platforms** / **meeting rooms**: turn each interval into +1 at its start and −1 at its end, sort the events, track the running count's maximum. Decide the tie order deliberately — does a train arriving at 10:00 need a fresh platform if another leaves at 10:00?
- **Order by a key that the exchange argument picks**: sort tasks by duration to maximise "deadline − finish time", because swapping two adjacent tasks out of that order never helps.

GAME · QUICK FIRE

Greedy or not?

For each problem, pick the approach that is actually correct.

1/5

Most non-overlapping meetings in one room.

A Sort by end time, take each that starts after the last taken

B Sort by start time, take each that fits

C Sort by length, take each that fits

More practice

- Non-overlapping Intervals — activity selection, counted the other way.
- Minimum Number of Arrows to Burst Balloons — earliest end again.
- Minimum Number of Platforms — the event sweep.
- Assign Cookies — sort both sides, match greedily.

- Gas Station — a greedy with a one-line proof.

EXERCISE 1

Activity selection

You are given `n` activities, each a half-open interval `[s, e)`. Choose as many as possible so that no two overlap. An activity that starts exactly when another ends does **not** overlap it. Print the largest number you can choose.

Input: `n` (`1 ≤ n ≤ 2·10^5`), then `n` lines `s e` (`0 ≤ s < e ≤ 10^9`). **Output:** one integer.

```
11
1 4
3 5
0 6
5 7
3 9
5 9
6 10
8 11
8 12
2 14
12 16
```

prints

```
4
```

For example `[1,4)`, `[5,7)`, `[8,11)`, `[12,16)`.

INPUT 1

```
11
1 4
3 5
0 6
5 7
3 9
5 9
6 10
8 11
8 12
2 14
12 16
```

OUTPUT 1

```
4
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<pair<ll, ll>> iv(n);   // {start, end}
    for (auto& [s, e] : iv) cin >> s >> e;

    // Earliest finish first: it leaves the most room for the rest (exchange argumer
    sort(iv.begin(), iv.end(), [](const auto& a, const auto& b) {
        return a.second != b.second ? a.second < b.second : a.first < b.first;
    });
    ll freeAt = LLONG_MIN;
    int taken = 0;
    for (const auto& [s, e] : iv) {
        if (s >= freeAt) {
            ++taken;
            freeAt = e;
        }
    }
    cout << taken << '\n';
    return 0;
}
```

EXERCISE 2

Minimum platforms

`n` trains use a station. Train `i` occupies a platform from its arrival `a_i` to its departure `d_i`, **both inclusive** — so a train arriving at the very moment another departs needs a different platform. Print the minimum number of platforms the station needs.

Input: `n` (`1 ≤ n ≤ 2·10^5`), then `n` lines `a d` (`0 ≤ a ≤ d ≤ 10^9`). **Output:** one integer.

```
6
900 910
940 1200
950 1120
1100 1130
1500 1900
1800 2000
```

prints

```
3
```

From 1100 to 1120 the trains arriving at 940, 950 and 1100 are all in.

INPUT 1

```
6
900 910
940 1200
950 1120
1100 1130
1500 1900
1800 2000
```

OUTPUT 1

```
3
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<pair<ll, int>> events;   // {time, 0 = arrival / 1 = departure}
    events.reserve(2 * n);
    for (int i = 0; i < n; ++i) {
        ll a, d;
        cin >> a >> d;
        events.push_back({a, 0});
        events.push_back({d, 1});
    }
    // Times are inclusive, so at a tie the arrival must be counted before the
    // departure frees its platform: pair ordering puts 0 before 1.
    sort(events.begin(), events.end());
    int here = 0, most = 0;
    for (const auto& [t, kind] : events) {
        if (kind == 0) most = max(most, ++here);
        else --here;
    }
    cout << most << '\n';
    return 0;
}
```

QUIZ

1. In the exchange argument for activity selection, why can the greedy's first activity replace an optimal schedule's first activity?

A. It is the shortest activity

B. Every optimal schedule already contains it

C. It ends no later than any activity, so it cannot overlap the optimal schedule's second one

D. It starts earliest, so it is always in every optimal schedule

2. Coins {1, 3, 4}, amount 6. How many coins does largest-coin-first use, and what is the optimum?

A. 2 coins; the optimum is 2

B. 3 coins; the optimum is 3

C. 4 coins; the optimum is 2

D. 3 coins; the optimum is 2

3. Arrival and departure times are inclusive. One train departs at 10 and another arrives at 10. In the event sweep, which order at time 10 gives the right answer?

A. Arrival before departure

B. Departure before arrival

C. Either order gives the same answer

D. Merge them into one event

4. Which greedy rule for picking the most non-overlapping intervals is correct?

A. Fewest overlaps first, without recomputing

B. Earliest end time first

C. Earliest start time first

D. Shortest interval first

ANSWER KEY · GREEDY ALGORITHMS — CHOOSE, THEN PROVE BY EXCHANGE

1. C The greedy's pick has the earliest end of all activities. The optimal schedule's second activity starts after its first ends, which is no earlier than the greedy's end, so the swap leaves a valid schedule of the same size. The greedy pick need not be in every optimum — only swappable into one.

2. D Greedy takes 4, then 1, then 1: three coins. 3 + 3 uses two. Coin systems like {1, 5, 10, 25} happen to be greedy-safe, but an arbitrary one needs a DP over amounts.

3. A With inclusive times both trains are present at 10, so the arrival must be counted before the departure frees its platform. With half-open times the opposite order would be right — the tie rule follows the statement.

4. B Earliest end is the one with an exchange proof. Earliest start fails on one long interval that starts first; shortest first fails when a short interval overlaps two compatible ones.

CHECKPOINT · 45 MIN

Checkpoint — Greedy algorithms and exchange proofs

A timed checkpoint on greedy algorithms: choosing a rule, proving it with an exchange argument, sweeping events with the right tie order and recognising when a problem needs DP instead.

How it works. Ten questions and one program, in **45 minutes** from when you start the timer. Clear 70% of the questions and get the program accepted on every hidden case to complete the module.

Before you start, make sure you can answer these from memory:

- The exchange argument for activity selection, in two sentences.
- Why earliest start and shortest first fail for activity selection.
- How the tie order of events changes between inclusive and half-open intervals.
- A coin system where largest-coin-first is wrong, and what solves it.
- Why shortest-duration-first maximises the sum of deadline minus finish time.

EXERCISE

Tasks and deadlines

You must do `n` tasks one after another, starting at time 0, in any order you choose. Task `i` takes `a_i` units of time and has deadline `d_i`. If it finishes at time `f`, you earn `d_i - f` points — negative when it is late. Print the largest total you can earn.

Input: `n` (`1 ≤ n ≤ 2·10^5`), then `n` lines `a d` (`1 ≤ a, d ≤ 10^6`). **Output:** one integer (possibly negative).

```
3
6 10
8 15
5 12
```

prints

```
2
```

The order 5, 6, 8 finishes at 5, 11, 19 and earns 7 − 1 − 4 = 2. Finish times add up past 10^{11} on the hidden cases.

INPUT 1

```
3
6 10
8 15
5 12
```

OUTPUT 1

```
2
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<pair<ll, ll>> t(n);   // {duration, deadline}
    for (auto& [a, d] : t) cin >> a >> d;

    // sum(d) is the same in every order; sum(finish) is smallest with the
    // shortest task first (swapping an adjacent longer-then-shorter pair
    // lowers it), so sort by duration.
    sort(t.begin(), t.end());
    ll now = 0, total = 0;
    for (const auto& [a, d] : t) {
        now += a;
        total += d - now;
    }
    cout << total << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. What does an exchange argument show?

A. That every solution is optimal

B. That the input can be sorted in O(n log n)

C. That an optimal solution can be changed into the greedy's solution without getting worse

D. That the greedy is faster than dynamic programming

2. Tasks and deadlines: why does the order of the deadlines not matter?

A. Because deadlines are sorted in the input

B. Because late tasks earn nothing

C. It does matter — sort by deadline

D. Every task's deadline is added exactly once whatever the order

3. Two adjacent tasks with durations 5 then 2, starting at time T. Swapping them to 2 then 5 changes the sum of their two finish times by how much?

A. It drops by 3

B. It drops by 7

C. It rises by 3

D. It does not change

4. Which problem is solved by a greedy algorithm?

A. Longest increasing subsequence by always extending the last element

B. Fractional knapsack

C. 0/1 knapsack

D. Coin change with arbitrary coin values

5. What is the cheapest way to test a greedy rule before a contest submission?

A. Check it on the sample only

B. Measure its running time

C. Compare it with a brute force on many small random inputs

D. Submit it and read the verdict

6. Activity selection: intervals `[1,3)`, `[3,5)`, `[2,4)`. How many can be chosen?

A. 1

B. 3

C. 0

D. 2

7. What is the time complexity of the minimum-platforms event sweep for n trains?

A. O(n log n)

B. O(n)

C. $O(n^2)$

D. O(T) where T is the largest time

8. Arrival/departure intervals are half-open `[a, d)`: a train leaving at 10 frees its platform for one arriving at 10. At a tie, which event goes first?

A. Whichever train has the smaller index

B. The departure

C. The arrival

D. It does not matter

9. Why does the tasks-and-deadlines total need `long long`?

A. Because sort requires 64-bit keys

B. It does not; int is enough

C. Finish times are summed over up to 2·10^5 tasks, each up to about 2·10^11

D. Deadlines can be negative

10. Your greedy passes the samples but you cannot find an exchange argument, and a brute force finds a counterexample with 4 items. What next?

A. Submit anyway

B. Sort in the opposite order

C. Add a special case for 4 items

D. Look for a dynamic programming formulation

ANSWER KEY · CHECKPOINT — GREEDY ALGORITHMS AND EXCHANGE PROOFS

1. C Swap the optimal solution's choices for the greedy's one at a time, each swap keeping it valid and no worse. At the end it is the greedy's solution, which is therefore optimal.

2. D The total is Σ d_i − Σ f_i. Σ d_i is a constant; only the finish times depend on the order, and they are smallest with the shortest tasks first.

3. A 5 then 2: finishes T+5 and T+7, sum 2T+12. 2 then 5: T+2 and T+7, sum 2T+9. The later task finishes at T+7 either way; the first finishes 3 earlier.

4. B When items can be cut, taking the best value per unit weight first is optimal. 0/1 knapsack and arbitrary coin change need DP.

5. C A brute force over every subset or order is easy to write for n ≤ 8 and finds a counterexample in seconds if the rule is wrong — far cheaper than a wrong-answer penalty.

6. D Sorted by end: [1,3), [2,4), [3,5). Take [1,3); [2,4) starts at 2 < 3, skip; [3,5) starts at 3 ≥ 3, take. Touching is allowed, so 2.

7. A Sorting the 2n events dominates; the sweep itself is O(n). A per-moment array would be O(T), and T reaches 10^9.

8. B Half-open means the leaving train is gone at 10, so its −1 must be counted before the arriving train's +1. The inclusive version of the exercise needs the opposite order.

9. C The last task finishes at up to 2·10^5 × 10^6 = 2·10^11, and the total subtracts many such finish times. int overflows long before.

10. D A counterexample means the local choice can rule out a better combination. That is the signal to keep more state — a DP over prefixes, capacities or amounts.

CodeKairo · Competitive Programming study plan · Module 5: Greedy algorithms and exchange proofs. Solve the exercises online to track your progress and earn XP.