Lab 5: Interval Scheduling Problems

Title: Lab 5: Interval Scheduling Problems
Date: 10/19/2025
Name: Robert Rice
Student ID: 657588340

---

1. Introduction

Objective: Analyze and solve two classic interval-scheduling variants using a principled greedy approach: (i) remove the minimum number of intervals so the remainder are non-overlapping; (ii) shoot the minimum number of vertical arrows to burst all balloons represented by horizontal intervals.
Problem Statement: Given a set of closed intervals $[s_i, e_i]$ on the real line, (1) find the minimum number of intervals to remove so that the remaining intervals are pairwise non-overlapping (touching at endpoints is allowed); (2) partition the intervals into the fewest 'arrow positions' so that each arrow x satisfies $s_i \leq x \leq e_i$ for every interval it covers.

2. Lab Results

All Goals Achieved: Yes.
Both tasks were solved by sorting intervals by their right endpoints and applying a single left-to-right greedy pass. The input of the samples provided matched the expected output.

3. Algorithm Framework

Primary approach: greedy to obtain the most completion time (interval scheduling). Sorting by end time leaves maximal room for future selections, which maximizes the number of intervals we can keep (hence, minimizes deletions) and groups overlapping intervals into the smallest number of arrow shots.

A. Minimum Removals to Avoid Overlap

- **Sort** intervals by increasing end time (break the ties by starting if desired).
- **Initialize** lastEnd $= -\infty$, kept $= 0$.
- **Scan** each interval $[s, e]$ in ordered:
  - If $s \geq$ lastEnd (touching allowed), **keep** it and set lastEnd $\leftarrow e$, kept $\leftarrow$ kept $+ 1$.
  - Otherwise, **skip** it (equivalently, *remove* it).
- **Answer:** removals $= n -$ kept.

Key insight: Keeping the interval that finishes earliest never reduces future options; by an exchange argument, there is an optimal solution that follows this rule.

B. Minimum Number of Arrows to Burst Balloons

- **Sort** intervals by increasing end time.
- **Initialize** arrows = 1, arrowX = end of the first interval.
- **Scan** next interval $[s, e]$:
  - If $s \leq$ arrowX, it is already covered by the current arrow (touch count), do nothing.
  - Else, **shoot a new arrow** at e: arrows $\leftarrow$ arrows + 1, arrowX $\leftarrow e$.
- **Answer:** arrows.

Key insight: Placing the arrow at *current earliest end* maximizes overlap with upcoming intervals; any earlier placement cannot cover more intervals than shooting at the earliest end.

Edge Cases Considered

- Touching endpoints (e.g. $[1, 2]$ and $[2, 3]$) are treated as nonoverlapping / discoverable by a single arrow.
- Negative coordinates and wide ranges are handled since comparisons are numeric and sorting is total.
- Duplicates or nested intervals are naturally handled by the greedy rule.

4. Complexity Analysis

Both algorithms share the same complexity profile, since they perform a sort and a linear scan.

- **Time Complexity (Worst Case):** $O(n \log n)$ due to sorting; the subsequent pass is $O(n)$.
- **Space Complexity:** $O(1)$ extra beyond input (when sorting is done). If a copy is made before sorting, it is $O(n)$.

Correctness Sketch: Consider any optimal solution that does not choose the earliest-end interval among those starting before its chosen arrow/kept boundary. Replacing that choice with the earliest-end interval cannot reduce future feasibility (it frees at least as much room for the remaining intervals), preserving optimality. Repeated application of this exchange yields a solution identical to the greedy one.

5. Lessons Learned, Feedback, and Conclusion

- **Challenges Faced:** Disentangling "touching" from "that overlap" and ensuring the comparator aligns with that rule; verifying off-by-one comparisons ($s \geq$ lastEnd vs. $s >$ lastEnd).

- **Insights Gained:** Many problems reduce to maximizing non-overlapping intervals via earliest-finish-time; “minimum removals” equals $n-\#$kept; “minimum arrows” equals the number of disjoint overlap groups induced by the same ordering.
- **Feedback:** A side-by-side comparison of classic interval variants (scheduling, removals, coloring/partitioning, stabbing with points) highlights how a single greedy invariant unifies solutions.

Conclusion: Sorting by right endpoints and making the locally optimal choice (keep compatible, or shoot at the current earliest end) yields globally optimal solutions for both tasks.

6. References and Use of Tools

References: Kleinberg & Tardos, *Algorithm Design*, Ch. 4 (Greedy Algorithms); CLRS, *Introduction to Algorithms*, interval scheduling notes.