Interval Coloring (a.k.a. Interval Partitioning):
Input: n intervals $I_1 = (s_1, f_1), \ldots, I_n = (s_n, f_n)$
Goal: Color all the intervals with as few colors as possible so that intervals having the same color do not intersect.

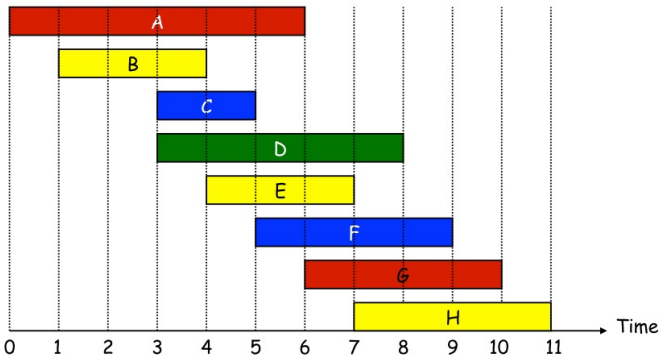

Figure 1: An interval coloring problem. 4 colors is used in this figure.

An optimal algorithm: Surprisingly the EST(earliest starting time) algorithm that considers intervals with ordering $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$ (which was arbitrarily bad for interval scheduling) now leads to an optimal greedy algorithm for interval coloring.

The algorithm: Having sorted the intervals by non-decreasing starting times, we assign each interval the smallest numbered color that is feasible given the intervals already colored. Note that this algorithm is equivalent to LFT (latest finishing time first).

Proof idea: The style of argument used to prove optimality is different here. Essentially, the argument is to show some intrinsic lower bound (in terms of some parameter of the input instance) for any allowable solution and then show that the greedy solution achieves this bound. In this case, **the bound (k) is the maximum number of intervals that can intersect at any point of time**. That is, $|G| = k \leqslant |OPT|$ where G is the set of colors used by the greedy algorithm EST and OPT is any solution. So an optimal algorithm must then use exactly k colors (as does the greedy algorithm). We show that the greedy algorithm will never use more than this number of colors.

Proof sketch: Recall that we have sorted the intervals by nondecreasing starting time (i.e. earliest start time first).
Let k be the maximum number of intervals in the input set that intersect at any given point.
Suppose for a contradiction that the algorithm used more than k colors.
Consider the first time (say on some interval l) that the greedy algorithm would have used $k + 1$ of colors.

- Then it must be that there are k intervals intersecting l.
- Let s_l be the starting time of interval I_l.
- These intersecting intervals must all include s_l.
- Hence, there are $k + 1$ intervals intersecting at s_l!

Minimum Spanning Tree (MST):
Input: Connected undirected weighted graph $G = (V, E)$ with weight (or cost) $c(e)$ for each edge $e \in E$.
Output: A spanning tree $T \subseteq E$ such that $cost(T)$ (sum of the costs of edges in T) is minimal.

Terminology:

- Spanning tree: acyclic connected subset of edges.
- Acyclic: does not contain any cycle.
- Connected: contains a path between any two vertices.

Properties: For all spanning trees T of a graph G,

- T contains exactly $n - 1$ edges;
- the removal of any edge from T disconnects it into two disjoint subtrees (otherwise T would contain a cycle);
- the addition of any edge to T creates exactly one cycle (otherwise T would be disconnected).

Note: We explain three **optimal** algorithms: Kruskal, Prim, Reverse-Delete

1- Kruskal's algorithm: Repeatedly put in smallest-cost edge remaining, as long as it doesn't create a cycle.

Pseudo-code:

Algorithm 1: Kruskal's algorithm

```
Sort edges such that c(e_1) ⩽ c(e_2) ⩽ ... ⩽ c(e_m)
T := {}
for k in [1, 2, ..., m] do
    let (u, v) = e_k
    if T does not contain a path between u and v then
        T = T ∪ {e_k}
```

Runtime? $\Theta(m \log m)$ for sorting; main loop involves sequence of m Union and FindSet operations on n elements which is $\Theta(m \log n)$. Total is $\Theta(m \log n)$ since $\log m$ is $\Theta(\log n)$.

Correctness: Using the exchange argument technique and induction. Similar to the optimality proof of EFT for the Interval Scheduling problem.

2- Prim's algorithm: Start with some vertex $s \in V$ (pick arbitrarily) and at each step, add lowest-cost edge that connects a new vertex to the existing partial tree.

Runtime? $\Theta(m \log n)$ using a heap to implement a priority queue of candidate edges to pick from next.

Correctness: Using the exchange technique with induction.

3- Reverse-Delete algorithm: Start with full graph; repeatedly delete highest-cost edge remaining, as long as it doesn't disconnect the graph we currently have.

Cut property: Assume that all edge costs are distinct. Let S be a subset of V such that neither S nor $V - S$ is empty. Let e be the edge with minimum cost with one endpoint in S and one endpoint in $V - S$. Then edge e is part of every minimum spanning tree.