2

CSE 421

Alg Design by Induction, Dynamic Programming

Shayan Oveis Gharan

Q/A

- I did terrible in my midterm what can I do?
  - First see if you have fundamental problems or simple mistakes?
  - Try to spend time on your areas of weakness.
  - Try more exercises: there are lots of exercise in the book
  - See https://train.usaco.org/usacogate

- Grades are not important after you leave school
  - Make sure you learn the material so you can use it for the rest of your life

- How to think, how to write?
  - Many cases it is better to spend more time on thinking than writing.

Sample Soln of Problem 2 Midterm

In HW2 we designed an O(m+n) time algorithm to find a vertex v_1 in a connected graph G such that $G - v_1$ is connected.

Repeat this k times. Since $G - v_1$ is connected it has a vertex v_2 such that $G - v_1 - v_2$ is connected. So after k times we will find k vertices $v_1, \dots, v_k$ such that $G - v_1 - \cdots v_k$ is connected.

The algorithm runs in time O(k(m+n)) which is polynomial.

Sample Soln of Problem 3 Midterm

ALG: Use ALG from class to find connected components of G. If G has a connected component such that number of edges is less than number of vertices output No otherwise Yes.

Runtime: ALG runs in time O(m+n) since we can find all components in this time.

Correctness: First suppose for every connected comp of G #edges >= #vertices. Then, by HW3-P1 we can orient every component of G s.t. indegree of every vertex is 1.

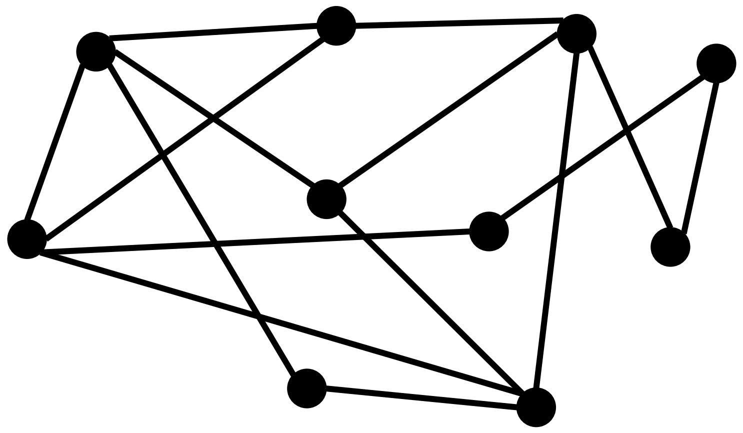
Now, suppose G has a component s.t.,

e:=#edges < #vertices=:v.

We claim that there is no orientation for this component. This is because by orienting each edge we increase sum of indegrees by 1. So the sum of indegrees of this comp = e. But if the indegree of every vertex is >=1 then sum of indegrees will be at least v. Contradiction.

Vertex Cover

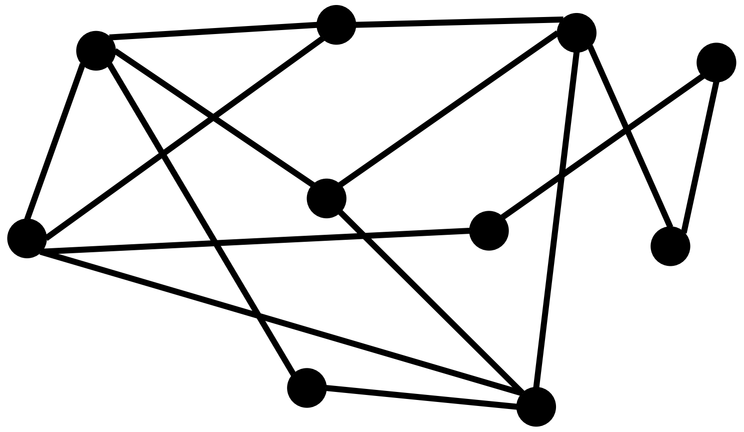
Given a graph G=(V,E), Find smallest set of vertices touching every edge



A Different Greedy Rule

Greedy 2: Iteratively, pick both endpoints of an uncovered edge.

Vertex cover = 6



Greedy (2) gives 2-approximation

Thm: Size of greedy (2) vertex cover is at most twice as big as size of optimal cover

Pf: Suppose Greedy (2) picks endpoints of edges $e_1, \dots, e_k$. Since these edges do not touch, every valid cover must pick one vertex from each of these edges!

i.e., $OPT \geq k$.

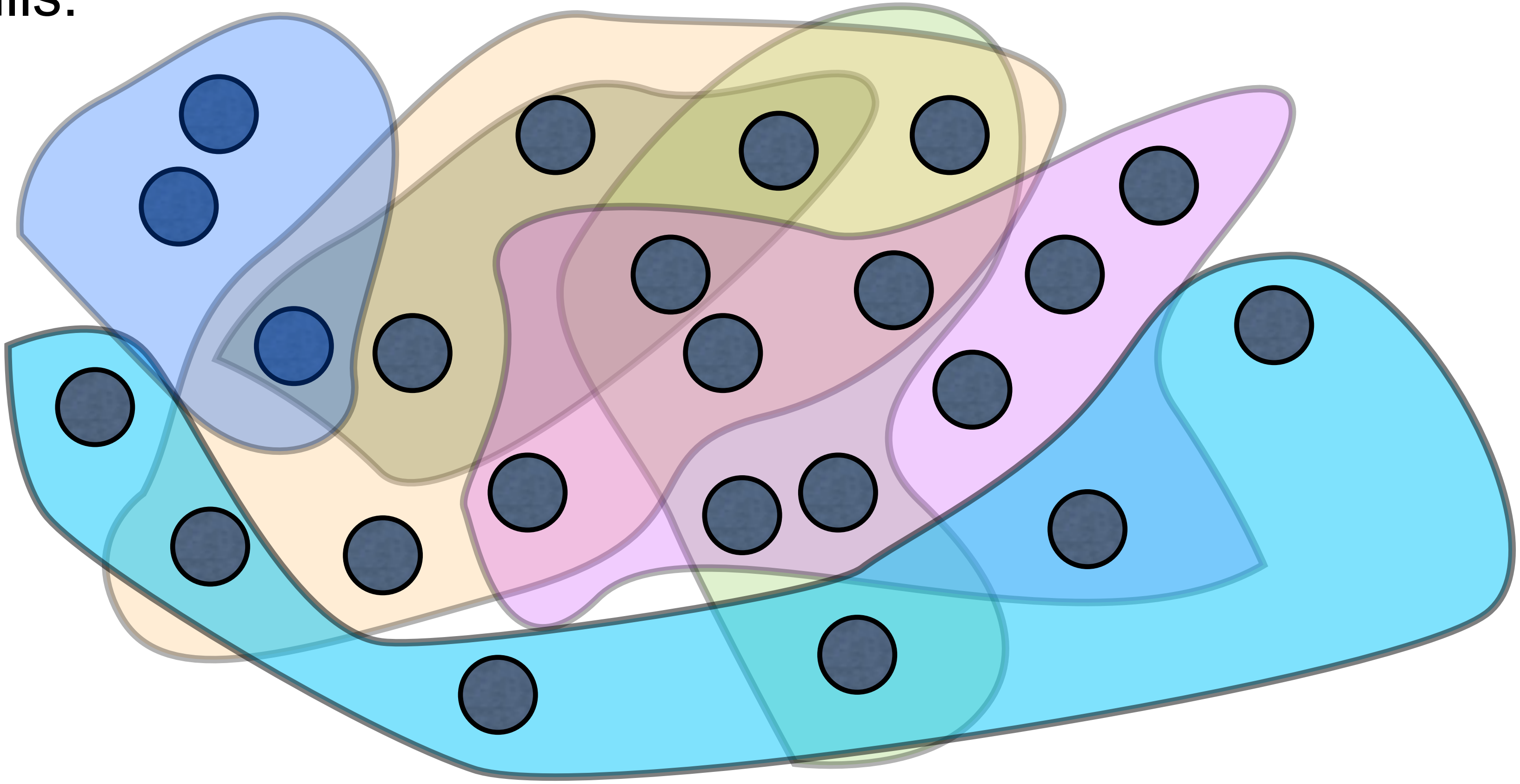
But the size of greedy cover is 2k. So, Greedy is a 2-approximation.

Set Cover

Given a number of sets on a ground set of elements,

Goal: choose minimum number of sets that cover all.

e.g., a company wants to hire employees with certain skills.

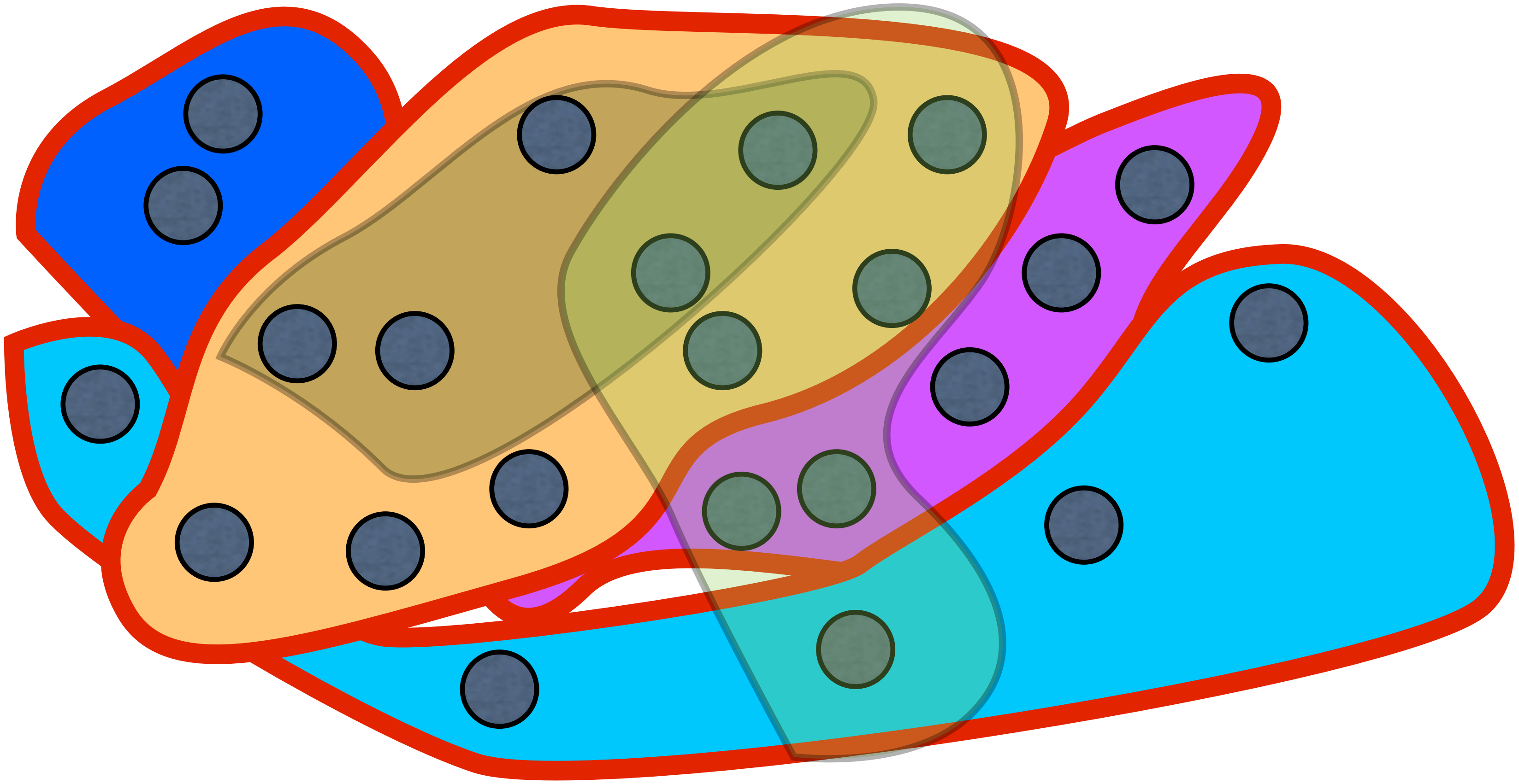


Set Cover

Given a number of sets on a ground set of elements,

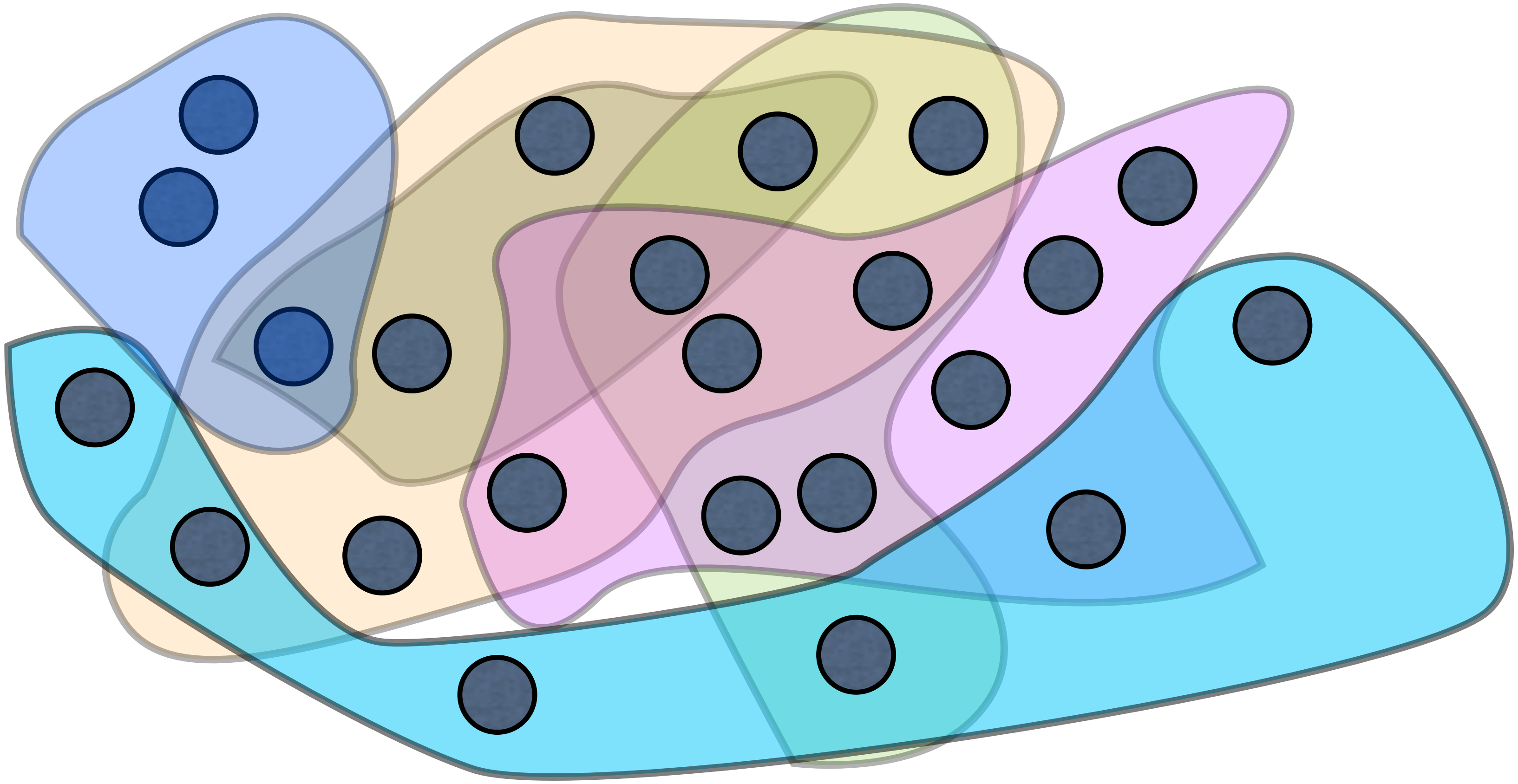
Goal: choose minimum number of sets that cover all.

Set cover = 4



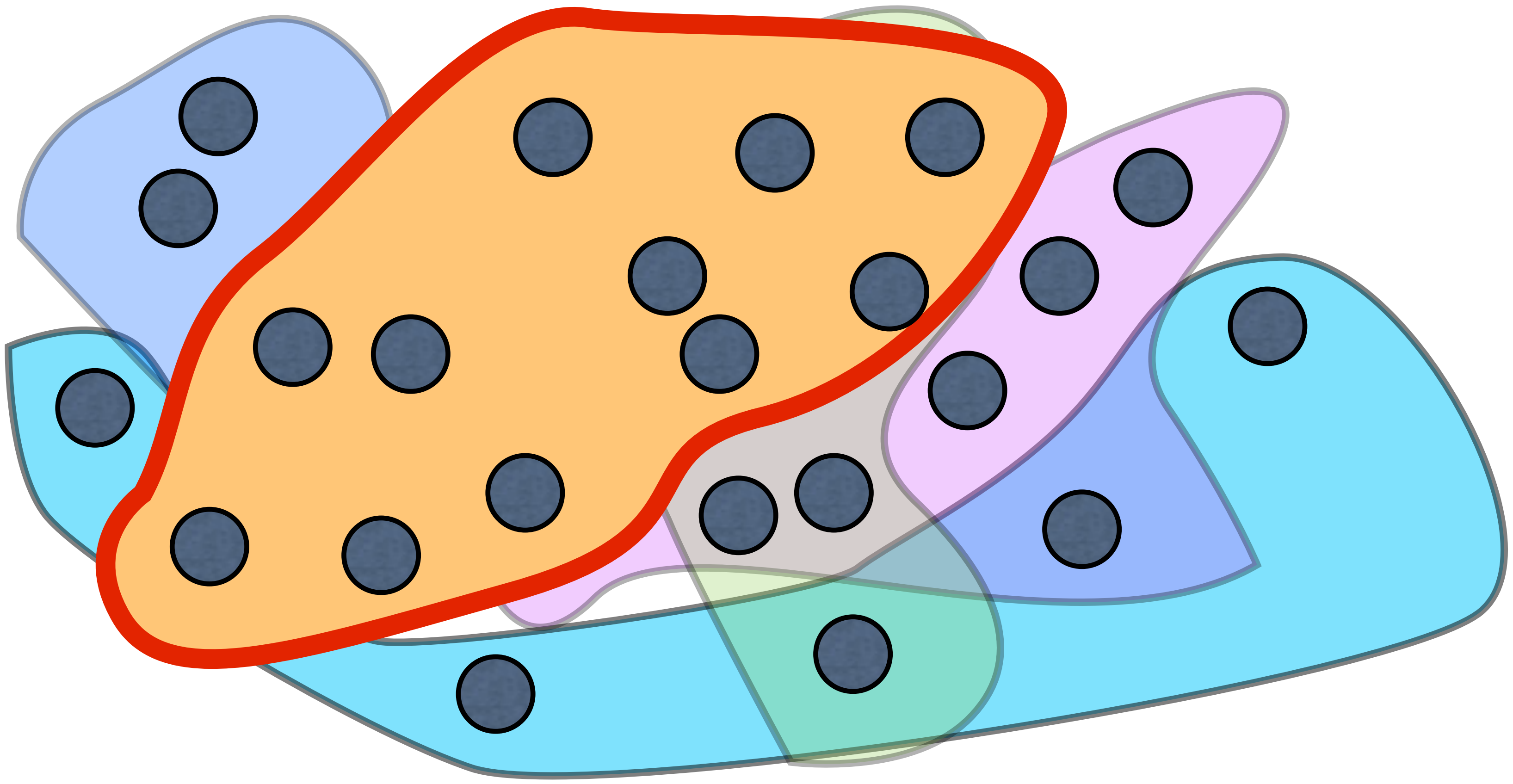
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



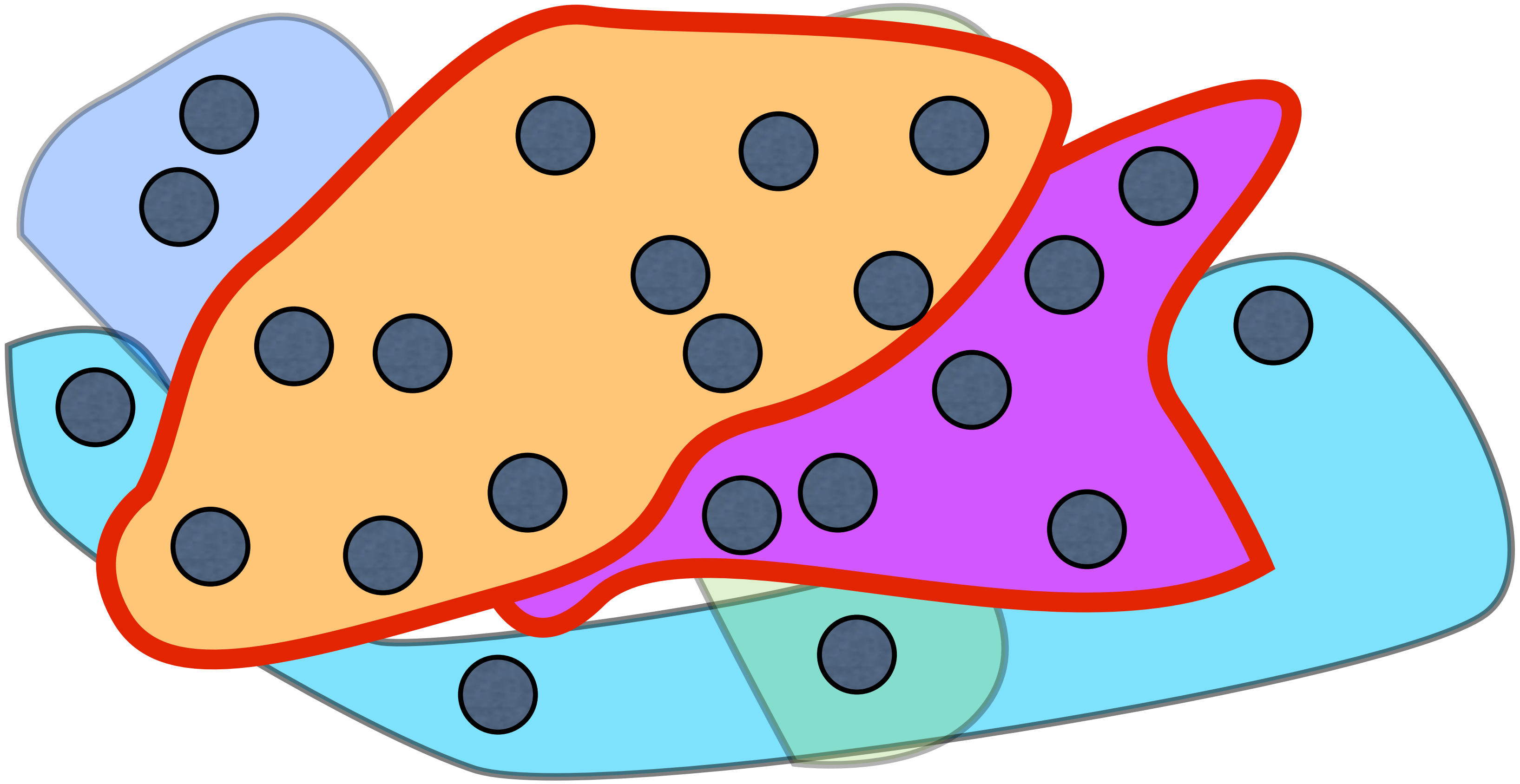
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



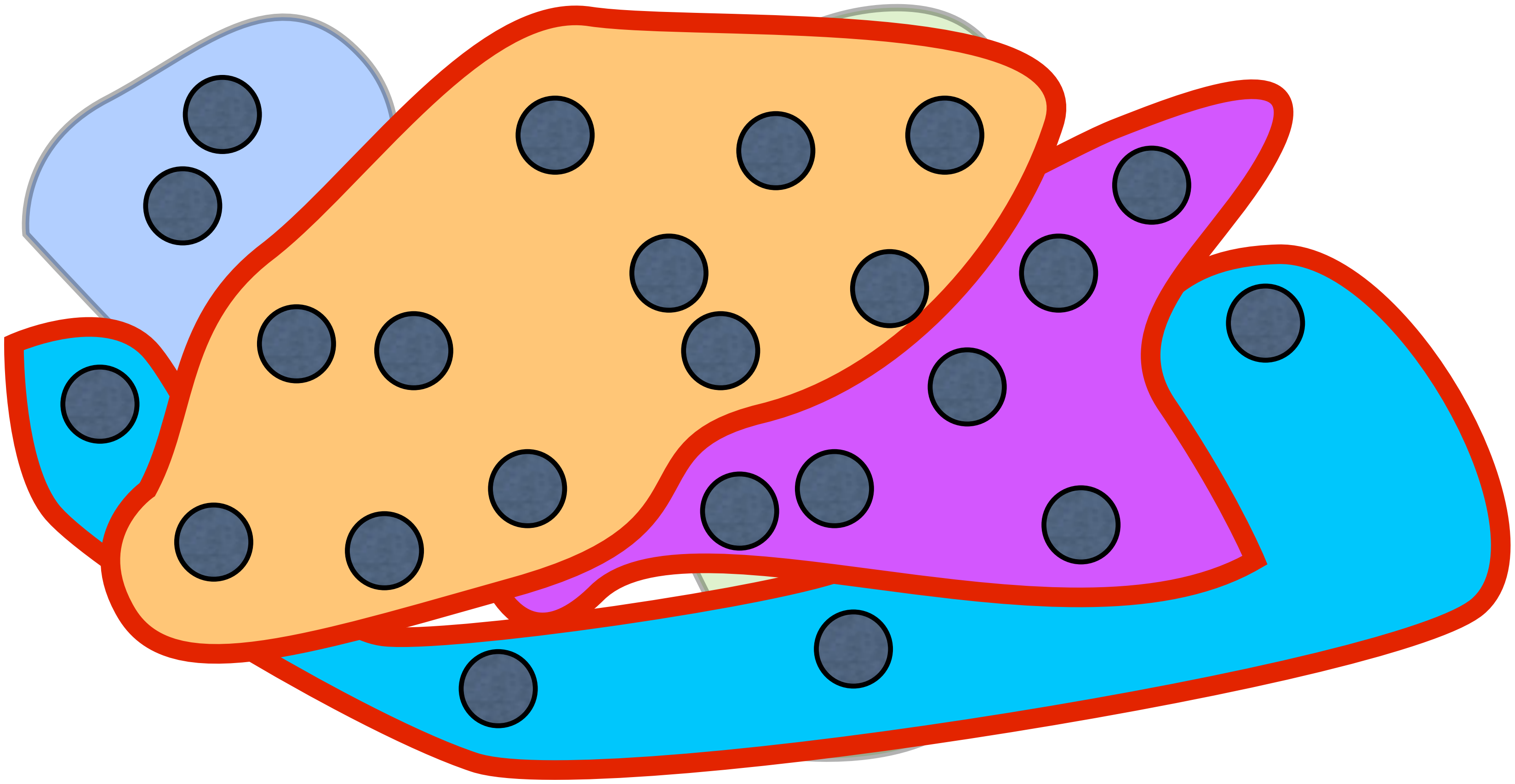
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered



A Greedy Algorithm

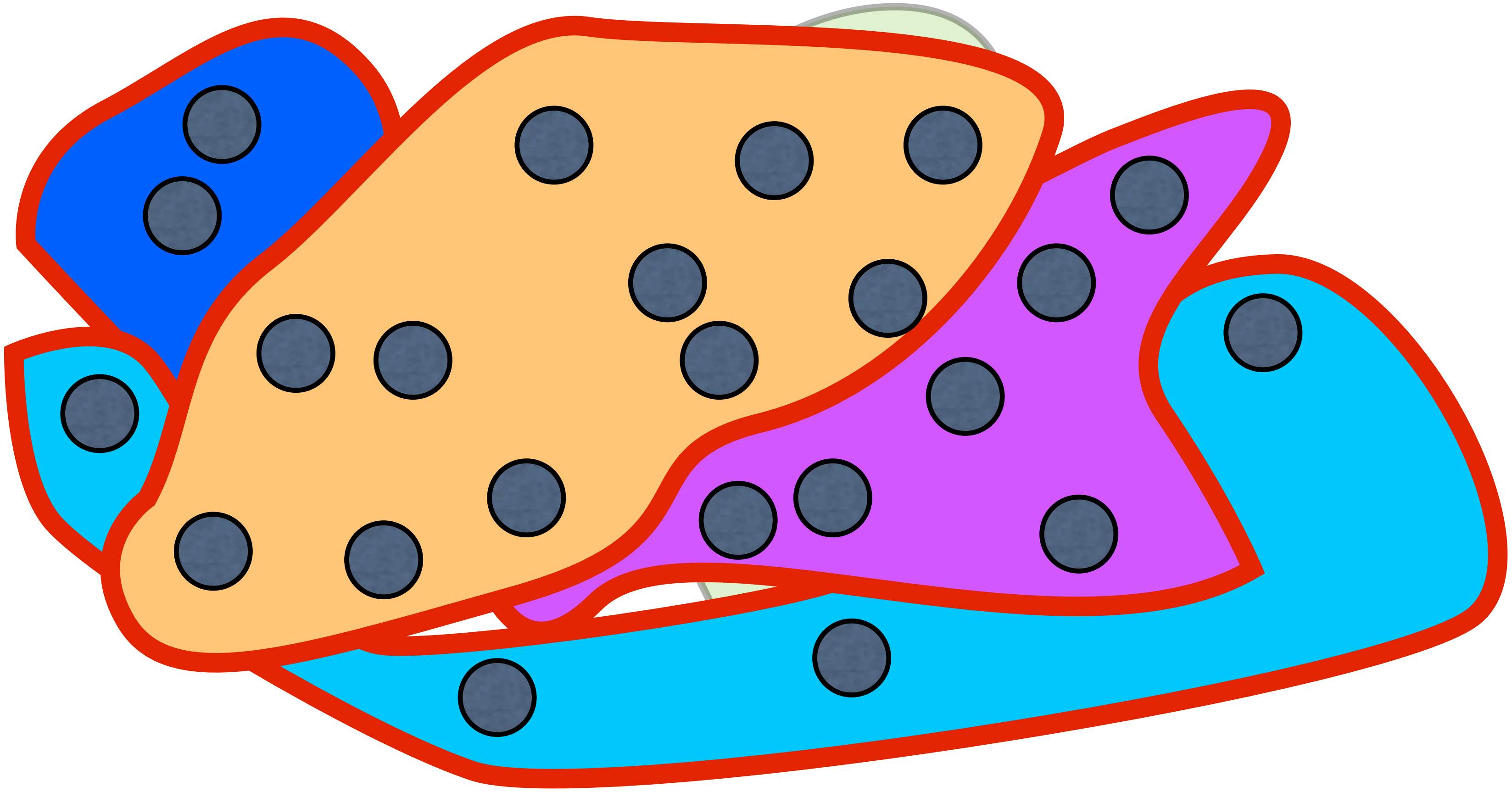
Strategy: Pick the set that maximizes # new elements covered



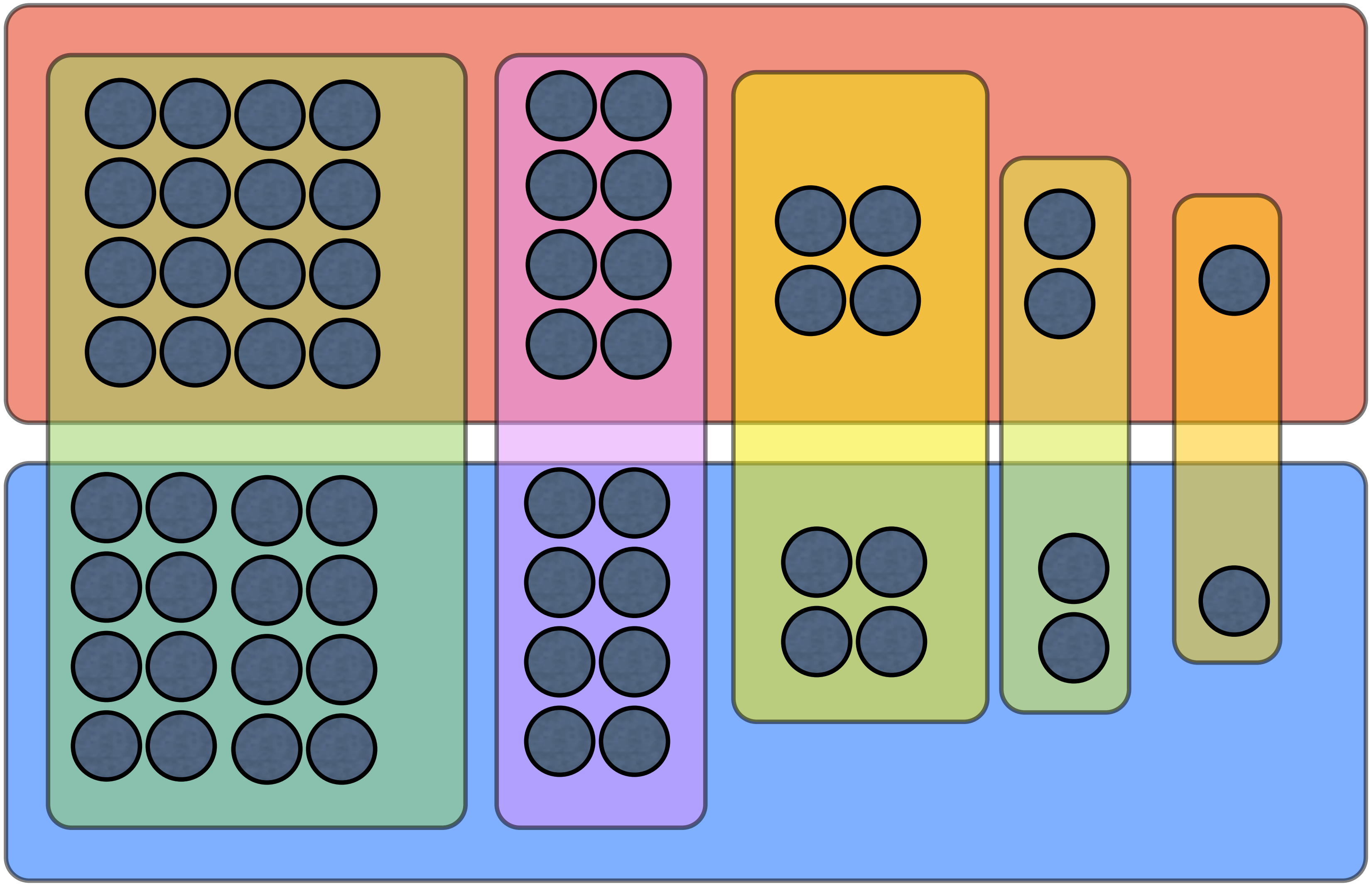
A Greedy Algorithm

Strategy: Pick the set that maximizes # new elements covered

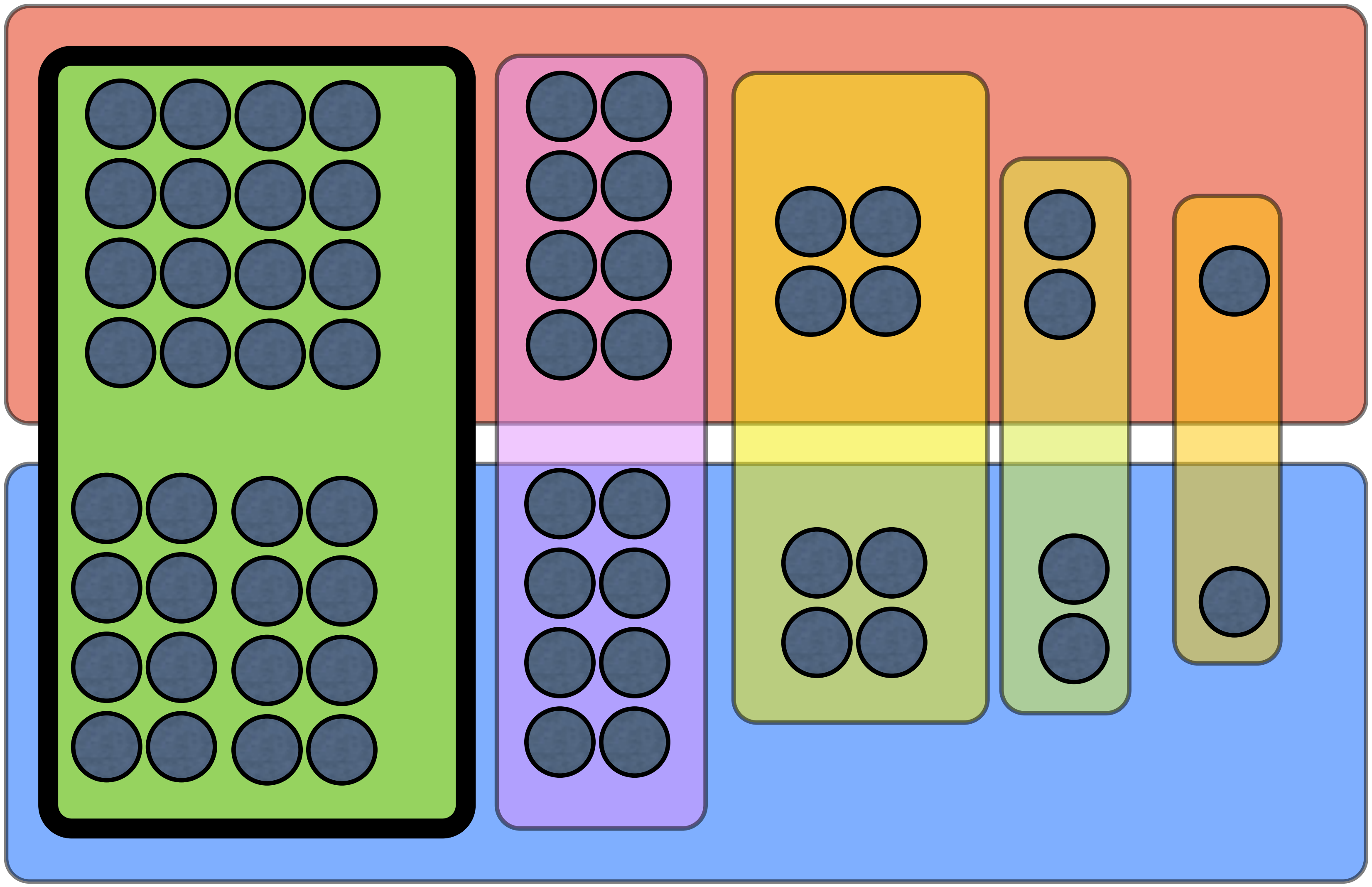
Thm: Greedy has ln n approximation ratio



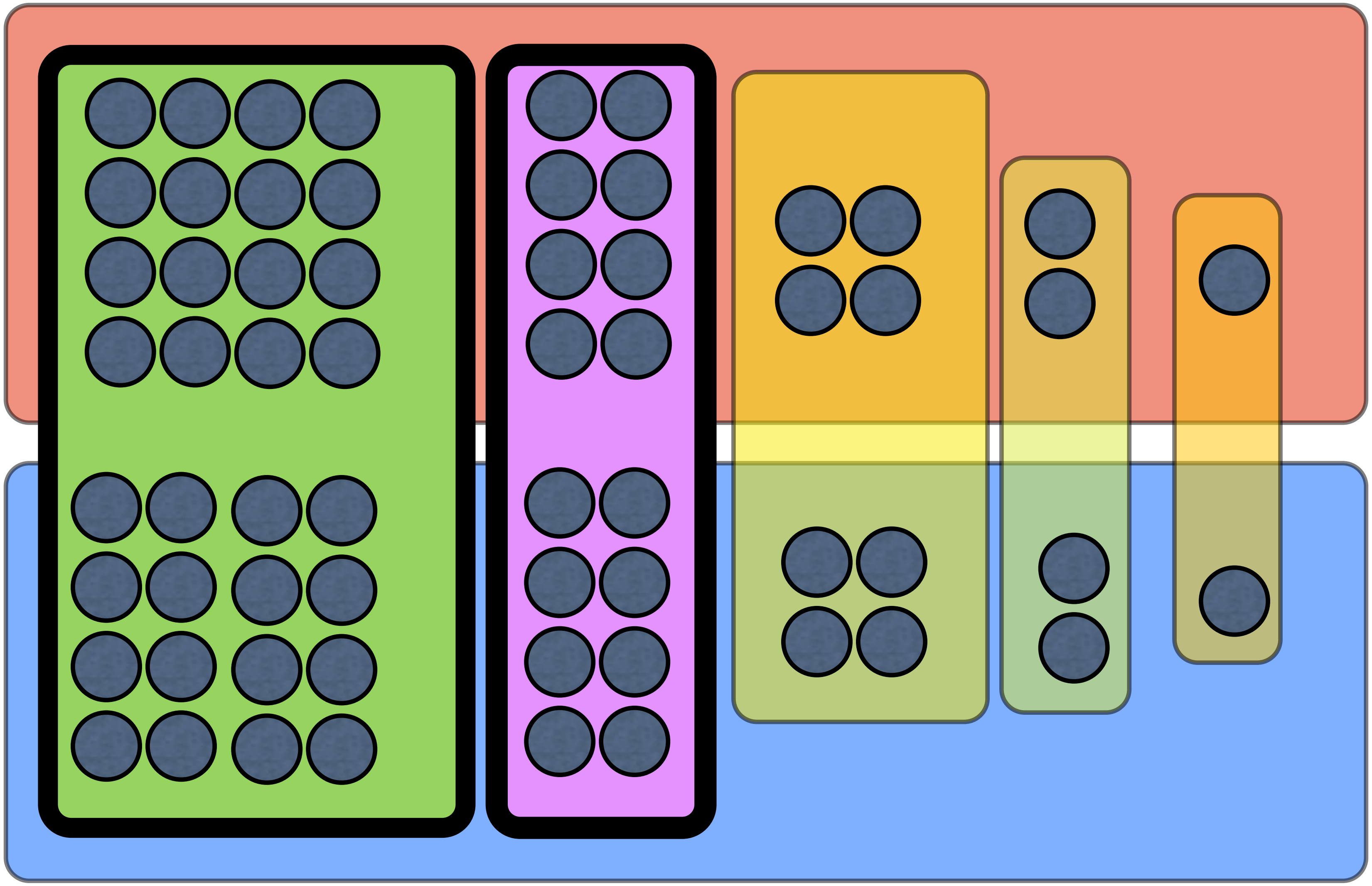
A Tight Example for Greedy



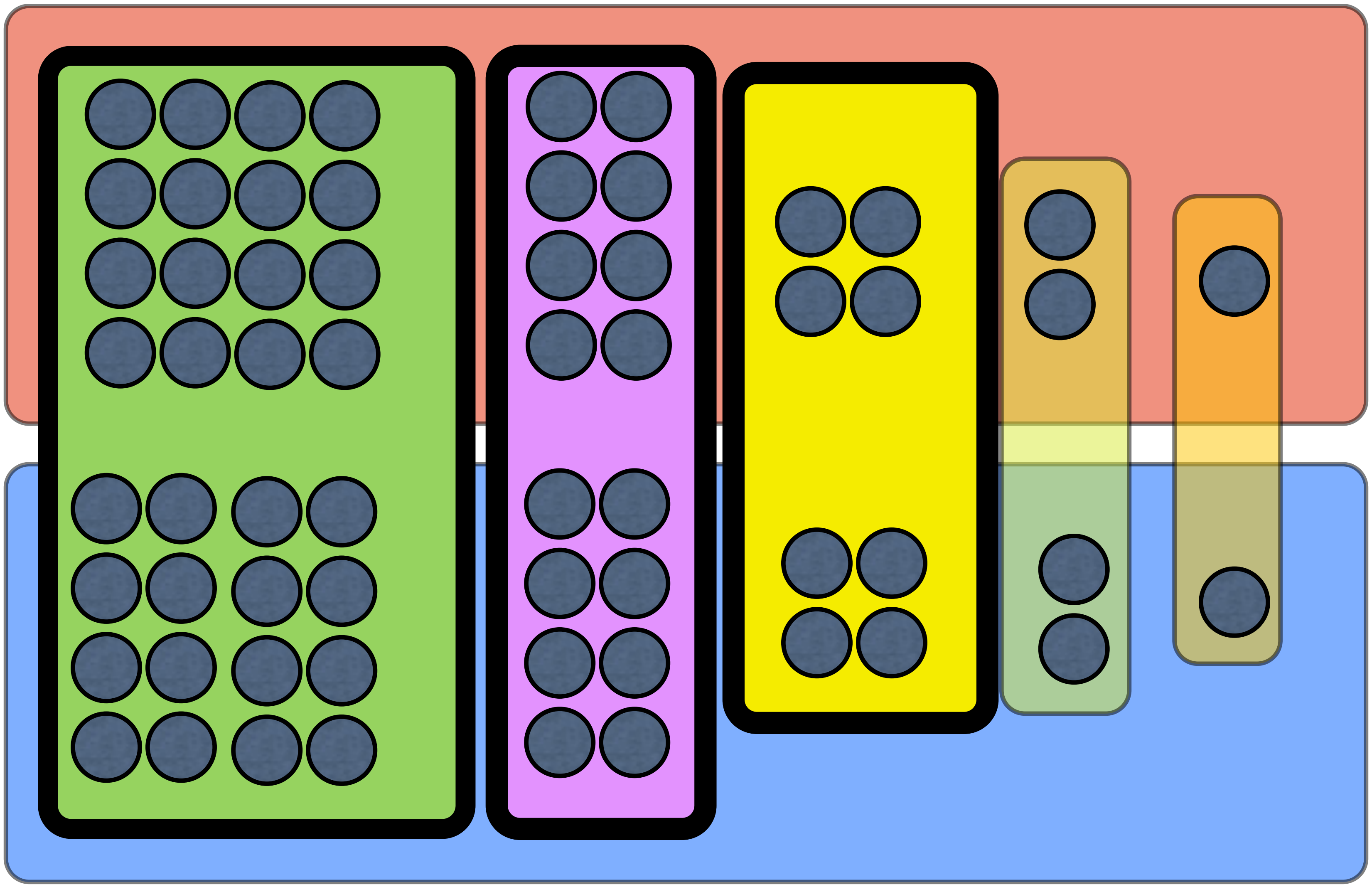
A Tight Example for Greedy



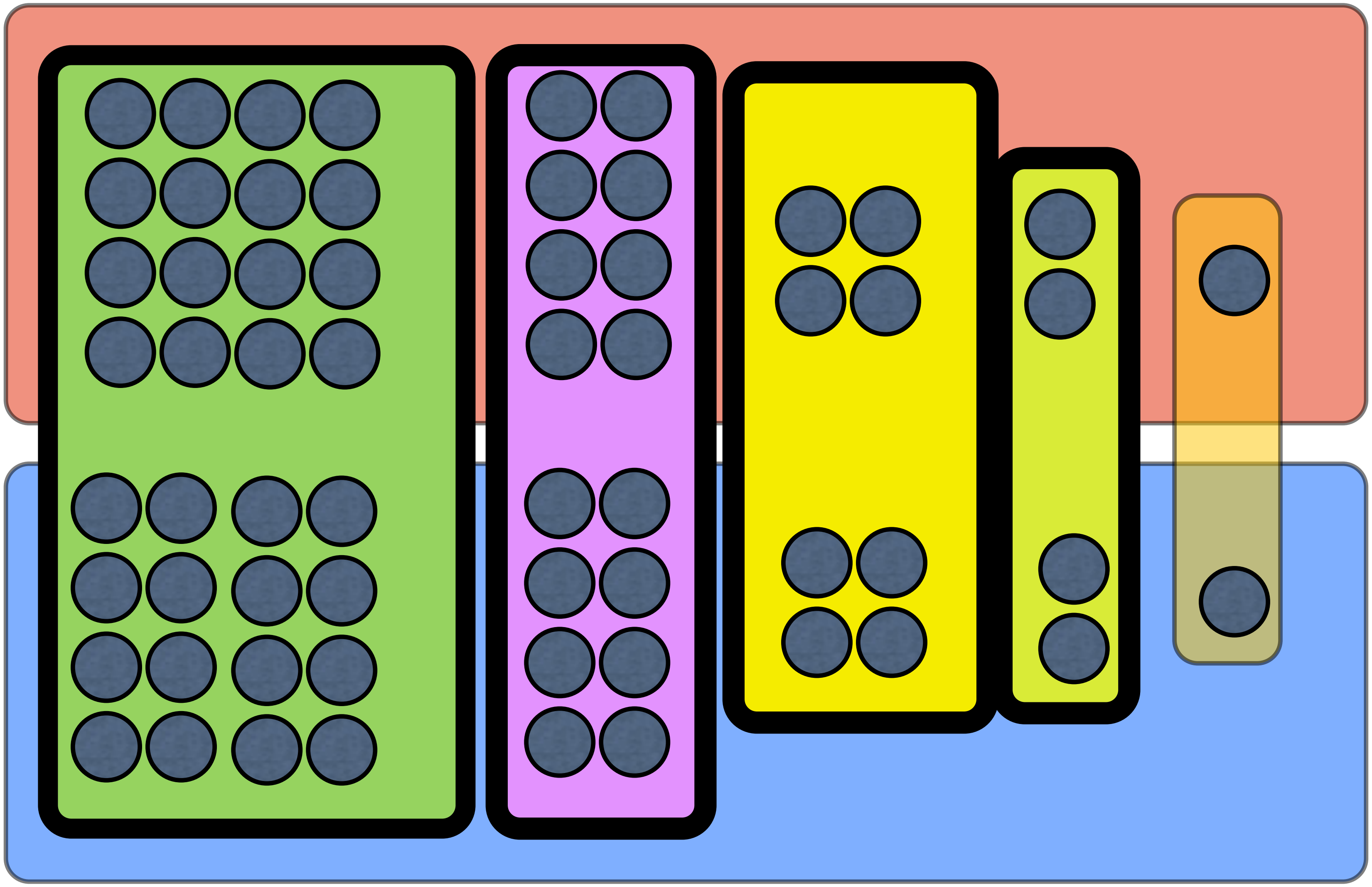
A Tight Example for Greedy



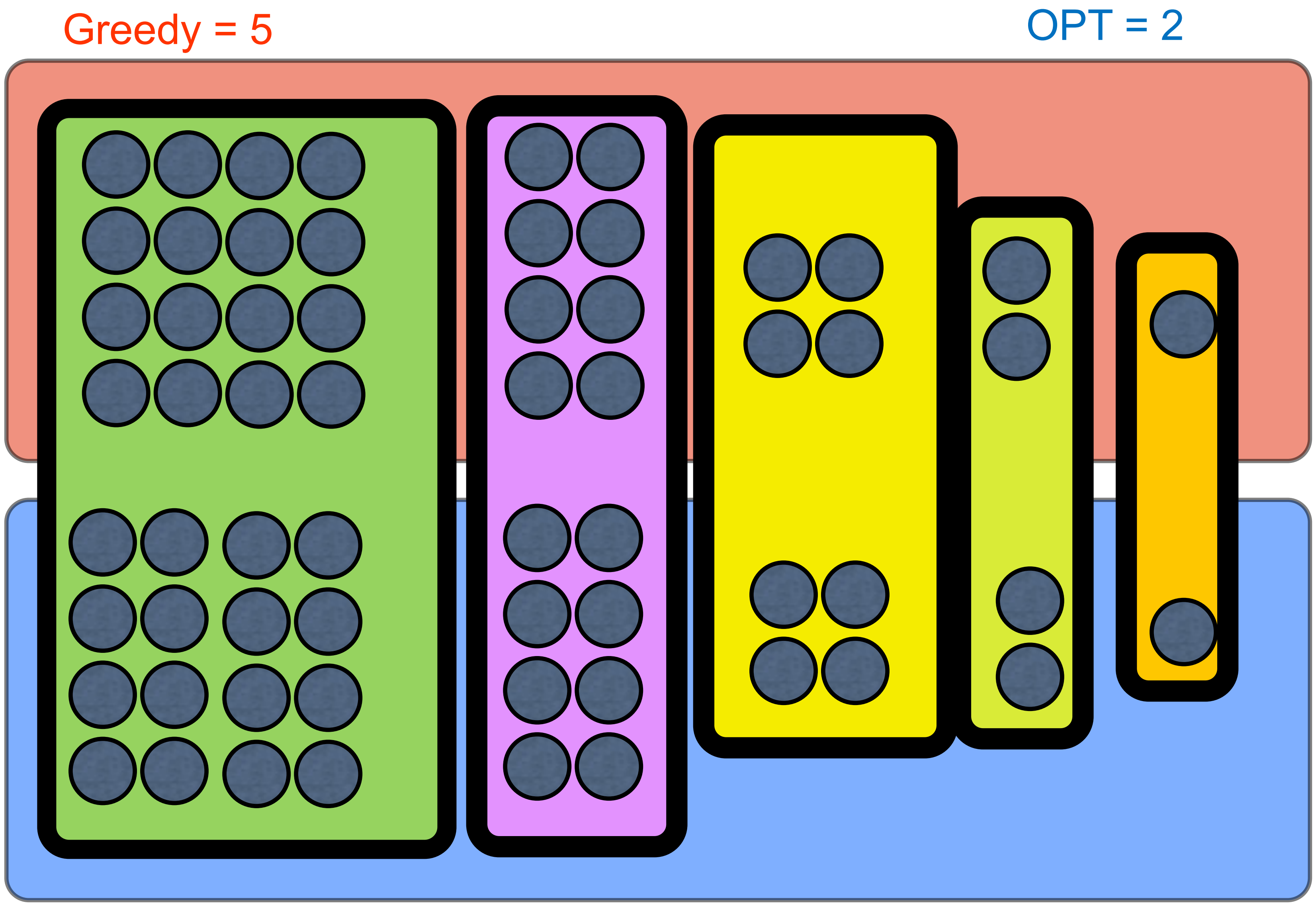
A Tight Example for Greedy



A Tight Example for Greedy



A Tight Example for Greedy



Greedy Gives O(log(n)) approximation

Thm: If the best solution has k sets, greedy finds at most k ln(n) sets.

Pf: Suppose OPT=k

There is set that covers 1/k fraction of remaining elements, since there are k sets that cover all remaining elements.

So in each step, algorithm will cover 1/k fraction of remaining elements.

#elements uncovered after t steps

$$\leq n\left(1-\frac{1}{k}\right)^{t} \leq ne^{-\frac{t}{k}}$$

So after $t = k \ln n$ steps, # uncovered elements < 1.

Approximation Alg Summary

- To design approximation Alg, always find a way to lower bound OPT
- The best known approximation Alg for vertex cover is the greedy.
  - It has been open for 50 years to obtain a polynomial time algorithm with approximation ratio better than 2
- The best known approximation Alg for set cover is the greedy.
  - It is NP-Complete to obtain better than ln n approximation ratio for set cover.

Strengthening Induction Hypothesis

We have seen examples on how to design algorithms by induction

Basic Idea: A solution to every instance can be constructed from solutions of smaller instances

In some cases it may help to strengthen the IH.
High-level plan: Prove $P(n) \wedge Q(n)$ inductively.

IH: Assume $P(n-1) \wedge Q(n-1)$.

IS: You may use $Q(n-1)$ to help you to prove $P(n)$
Remember you also have to prove $Q(n)$.

Maximum Consecutive Subsequence

Problem: Given a sequence $x_1, \ldots, x_n$ of integers (not necessarily positive),

Goal: Find a subsequence of consecutive elements s.t., the sum of its numbers is maximum.

$$1 \quad -3 \quad \boxed{7 \quad -2 \quad -3 \quad 8} \quad -10 \quad 1 \quad -7$$

Applications: Figuring out the highest interest rate period in stock market

Brute Force Approach

Try all consecutive subsequences of the input sequence.

There are $\binom{n}{2} = \Theta(n^2)$ such sequences.

We can compute the sum of numbers in each such sequence in $O(n)$ steps.

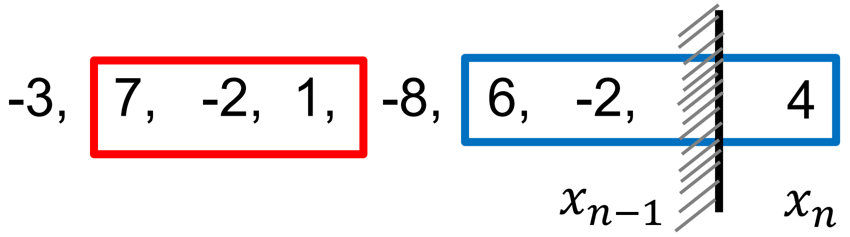
So, the ALG runs in $O(n^3)$.

With a clever loop we can do this in $O(n^2)$.
But, can we solve in linear time?

First Attempt (Induction)

Suppose we can find the maximum-sum subsequence of $x_1, \dots, x_{n-1}$. Say it is $x_i, \dots, x_j$

- If $x_n < 0$ then it does not belong to the largest subsequence. So, we can output $x_i, \dots, x_j$
- Suppose $x_n > 0$.
  - If $j = n - 1$ then $x_i, \dots, x_n$ is the maximum-sum subsequence.
  - If $j < n - 1$ there are two possibilities
    1) $x_i, \dots, x_j$ is still the maximum-sum subsequence
    2) A sequence $x_k, \dots, x_n$ is the maximum-sum subseqence



Second Attempt (Strengthing Ind Hyp)

Stronger Ind Hypothesis: Given $x_1, \dots, x_{n-1}$ we can compute the maximum-sum subsequence, and the maximum-sum suffix subsequence.

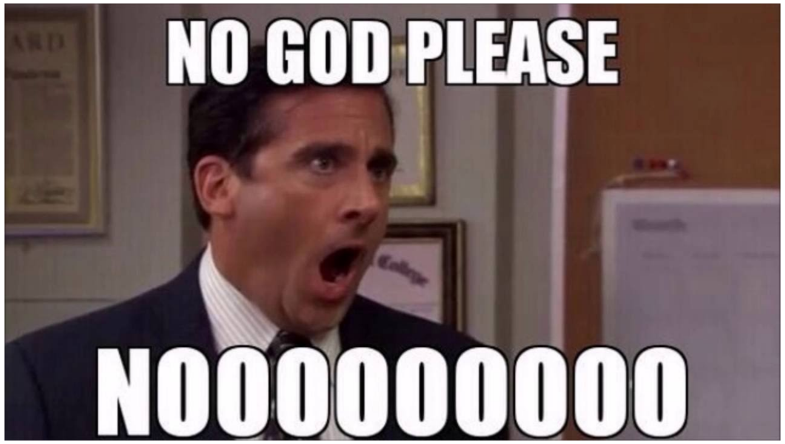
Can be empty

$$-3,\ \boxed{\underset{x_i}{7},\ \ -2,\ \ \underset{x_j}{1},}\ -8,\ \boxed{\underset{x_k}{6},\ \ \underset{x_{n-1}}{-2}}$$

Say $\boldsymbol{x_i, \dots, x_j}$ is the maximum-sum and $x_k, \dots, x_{n-1}$ is the maximum-sum suffix subsequences.

- If $x_k + \dots + x_{n-1} + x_n > x_i + \dots + x_j$ then $x_k, \dots, x_n$ will be the new maximum-sum subsequence

Are we done?



Updating Max Suffix Subsequence

$$-3,\quad 7,\quad -2,\quad 1,\quad -8,\quad \boxed{6,\quad -2,}\quad \Big|\quad \underset{x_n}{4}$$

Say $x_k, \dots, x_{n-1}$ is the maximum-sum suffix subsequences of $x_1, \dots, x_{n-1}$.

- If $x_k + \cdots + x_n \geq 0$ then,
  $x_k, \dots, x_n$ is the new maximum-sum suffix subsequence

- Otherwise,
  The new maximum-sum suffix is the empty string.

Maximum Sum Subsequence ALG

```
Initialize S=0 (Sum of numbers in Maximum Subseq)
Initialize U=0 (Sum of numbers in Maximum Suffix)
for  (i=1 to n) {
    if (x[i] + U > S)
        S = x[i] + U

    if (x[i] + U > 0)
        U = x[i] + U
    else
        U = 0
}
Output S.
```

-3 7 -2 1 -8 6 -2 4

Pf of Correct: Maximum Sum Subseq

Ind Hypo: Suppose
- $x_i, \dots, x_j$ is the max-sum-subseq of $x_1, \dots, x_{n-1}$
- $x_k, \dots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \dots, x_{n-1}$

Ind Step: Suppose $x_a, \dots, x_b$ is the max-sum-subseq of $x_1, \dots, x_n$

Case 1 ($b < n$): $x_a, \dots, x_b$ is also the max-sum-subseq of $x_1, \dots, x_{n-1}$
So, $a = i, b = j$ and the algorithm correctly outputs OPT

Case 2 ($b = n$): We must have $x_a, \dots, x_{b-1}$ is the max-suff-sum of $x_1, \dots, x_{n-1}$.
If not, then

$$x_k + \cdots x_{n-1} > x_a + \cdots + x_{n-1}$$

So, $x_k + \cdots + x_n > x_a + \cdots + x_b$ which is a contradiction.
Therefore, $a = k$ and the algorithm correctly outputs OPT

Special Cases (You don't need to mention if follows from above):
- The max-suffix-sum is empty string
- There are multiple maximum sum subsequences.

Pf of Correct: Max-Sum Suff Subseq

Ind Hypo: Suppose

- $x_i, \dots, x_j$ is the max-sum-subseq of $x_1, \dots, x_{n-1}$
- $x_k, \dots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \dots, x_{n-1}$

Ind Step: Suppose $x_a, \dots, x_n$ is the max-suffix-sum-subseq of $x_1, \dots, x_n$
Note that we may also have an empty sequence

Case 1 (OPT is empty): Then, we must have $x_k + \cdots + x_n < 0$. So the algorithm correctly finds max-suffix-sum subsequence.

Case 2 ($x_a, \dots, x_n$ is nonempty): We must have $x_a + \cdots + x_n \geq 0$.
Also, $x_a, \dots, x_{n-1}$ must be the max-suffix-sum of $x_1, \dots, x_{n-1}$. If not,

$$x_a + \cdots + x_{n-1} < x_k + \cdots + x_{n-1}$$

which implies $x_a + \cdots + x_n < x_k + \cdots + x_n$ which is a contradiction.

Therefore, $a = k$. So, the algorithm correctly finds max-suffix-sum subseqence.

Summary

- Try to reduce an instance of size n to smaller instances
  - Never solve a problem twice
- Before designing an algorithm study properties of optimum solution
- If ordinary induction fails, you may need to strengthen the induction hypothesis