

CSE 421

Divide and Conquer: Integer Multiplication

Shayan Oveis Gharan

Master Theorem

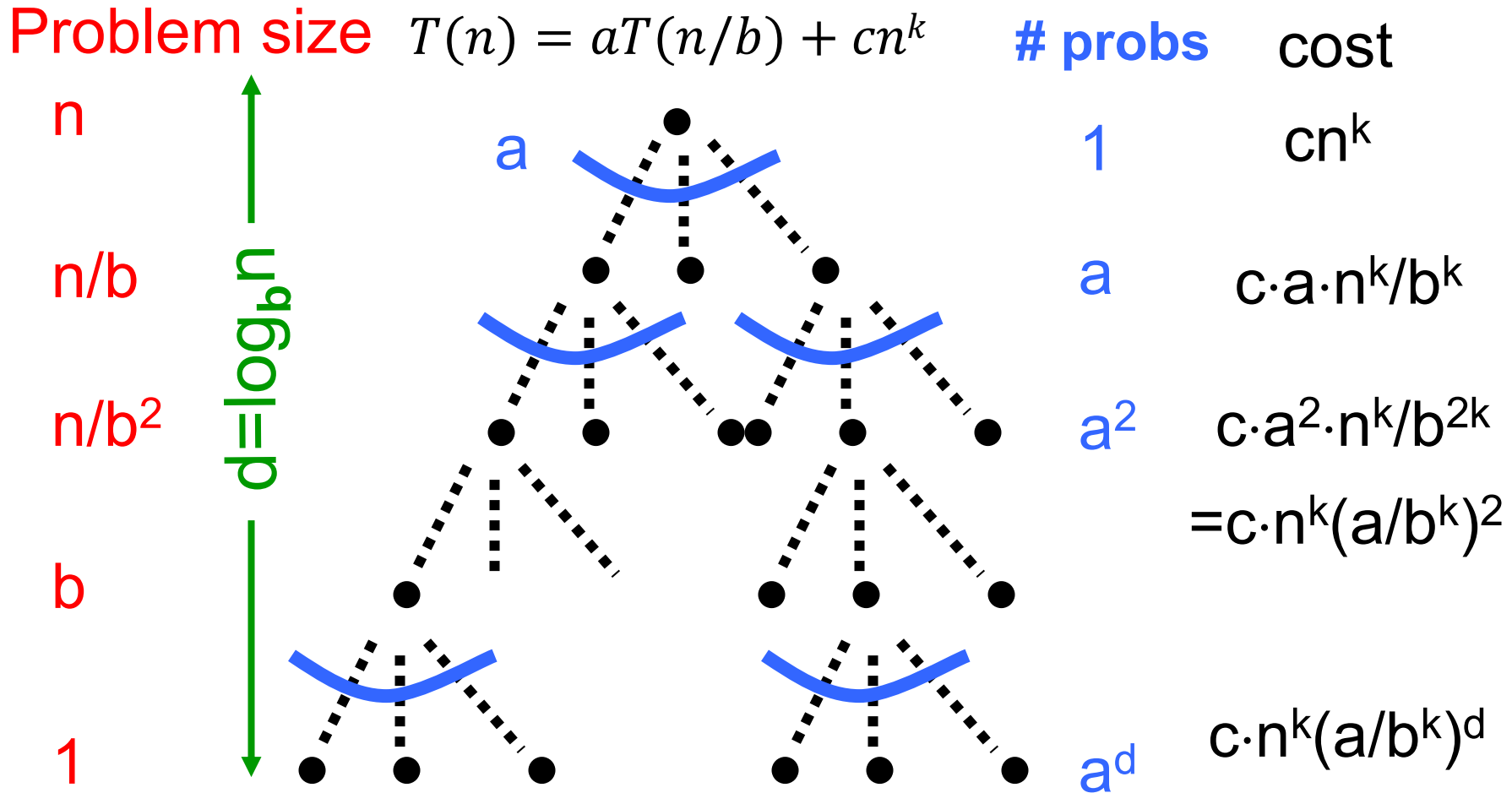
Suppose $T(n) = a T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$
- If $a < b^k$ then $T(n) = \Theta(n^k)$
- If $a = b^k$ then $T(n) = \Theta(n^k \log n)$

Works even if it is $\lceil \frac{n}{b} \rceil$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1, k \geq 0$ and $T(n) = O(1)$ for $n \leq b$.

Proving Master Theorem



$$T(n) = cn^k \sum_{i=0}^{d=\log_b n} \left(\frac{a}{b^k}\right)^i$$

A Useful Identity

Theorem: $1 + x + x^2 + \cdots + x^d = \frac{x^{d+1} - 1}{x - 1}$

Pf: Let $S = 1 + x + x^2 + \cdots + x^d$

Then, $xS = x + x^2 + \cdots + x^{d+1}$

So, $xS - S = x^{d+1} - 1$

i.e., $S(x - 1) = x^{d+1} - 1$

Therefore,

$$S = \frac{x^{d+1} - 1}{x - 1}$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k, a > b^k$

$$T(n) = cn^k \sum_{i=0}^{\log_b n} \left(\frac{a}{b^k}\right)^i$$

$$\frac{x^{d+1}-1}{x-1} \text{ for } x = \frac{a}{b^k}$$

$$d = \log_b n$$

using $x \neq 1$

$$= cn^k \frac{\left(\frac{a}{b^k}\right)^{\log_b n + 1} - 1}{\left(\frac{a}{b^k}\right) - 1}$$

$$b^k \log_b n$$

$$= (b^{\log_b n})^k$$

$$= n^k$$

$$= c \left(\frac{n^k}{b^k \log_b n} \right) \frac{\left(\frac{a}{b^k}\right)^{\log_b n + 1} - 1}{\left(\frac{a}{b^k}\right) - 1} a^{\log_b n}$$

$$a^{\log_b n}$$

$$= (b^{\log_b a})^{\log_b n}$$

$$= (b^{\log_b n})^{\log_b a}$$

$$= n^{\log_b a}$$

$$\leq c' a^{\log_b n} = O(n^{\log_b a})$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k$, $a = b^k$

$$\begin{aligned} T(n) &= cn^k \sum_{i=0}^{\log_b n} \left(\frac{a}{b^k}\right)^i \\ &= cn^k \log_b n \end{aligned}$$

Master Theorem

Suppose $T(n) = a T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$
- If $a < b^k$ then $T(n) = \Theta(n^k)$
- If $a = b^k$ then $T(n) = \Theta(n^k \log n)$

Works even if it is $\lceil \frac{n}{b} \rceil$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1, k \geq 0$ and $T(n) = O(1)$ for $n \leq b$.

Median

Selecting k-th smallest

Problem: Given numbers x_1, \dots, x_n and an integer $1 \leq k \leq n$ output the k -th smallest number

$$\text{Sel}(\{x_1, \dots, x_n\}, k)$$

A simple algorithm: Sort the numbers in time $O(n \log n)$ then return the k -th smallest in the array.

Can we do better?

Yes, in time $O(n)$ if $k = 1$ or $k = n$.

Can we do $O(n)$ for all possible values of k ?

Assume all numbers are distinct for simplicity.

An Idea

Choose a number w from x_1, \dots, x_n

Define

- $S_{<}(w) = \{x_i : x_i < w\}$
- $S_{=}(w) = \{x_i : x_i = w\}$
- $S_{>}(w) = \{x_i : x_i > w\}$

Can be computed in
linear time

Solve the problem recursively as follows:

- If $k \leq |S_{<}(w)|$, output $Sel(S_{<}(w), k)$
- Else if $k \leq |S_{<}(w)| + |S_{=}(w)|$, output w
- Else output $Sel(S_{>}(w), k - |S_{<}(w)| - |S_{=}(w)|)$

Ideally want $|S_{<}(w)|, |S_{>}(w)| \leq n/2$. In this case ALG runs in $O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) + \dots + O(1) = O(n)$.

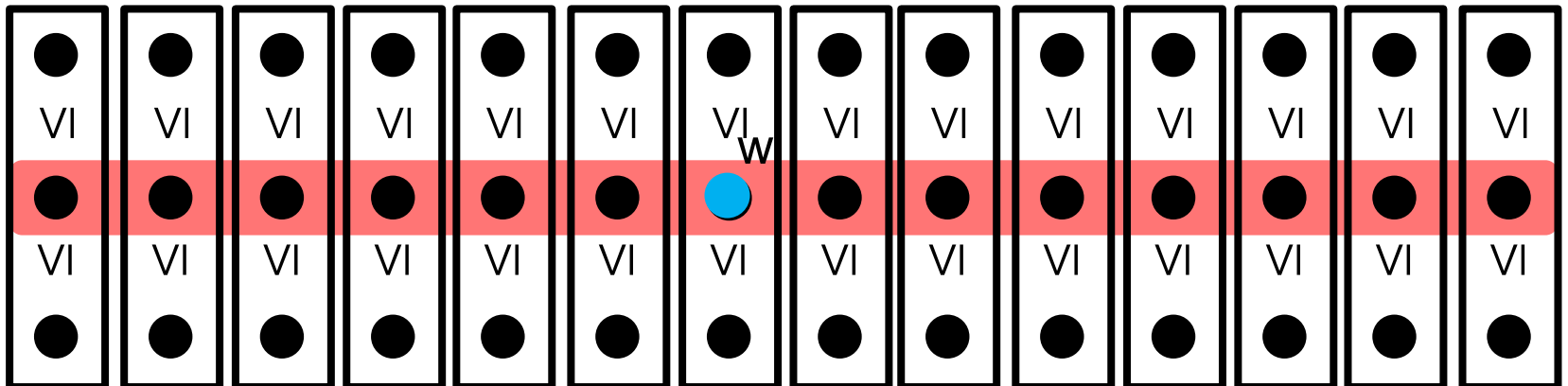
How to choose w ?

Suppose we choose w uniformly at random
similar to the pivot in quicksort.

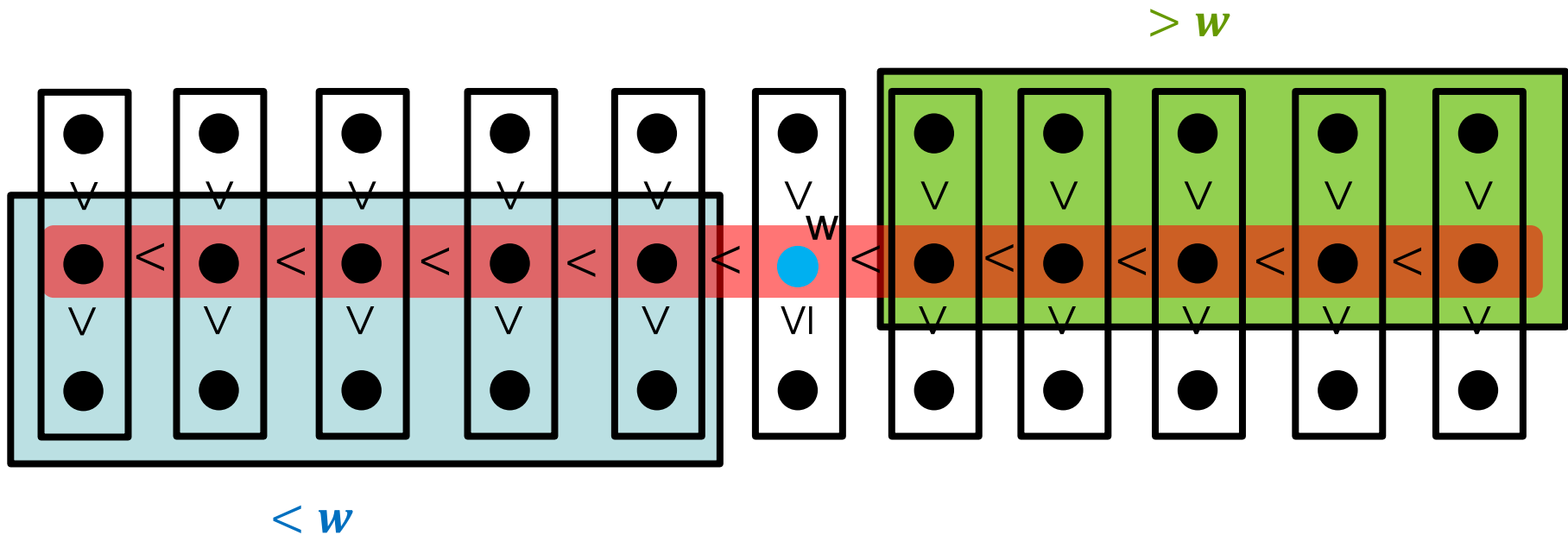
Then, $\mathbb{E}[|S_{<}(w)|] = \mathbb{E}[|S_{>}(w)|] = n/2$. Algorithm runs in $O(n)$ in expectation.

Can we get $O(n)$ running time deterministically?

- Partition numbers into sets of size 3.
- Sort each set (takes $O(n)$)
- $w = \text{Sel}(\text{midpoints}, n/6)$



How to lower bound $|S_{<}(w)|$, $|S_{>}(w)|$?



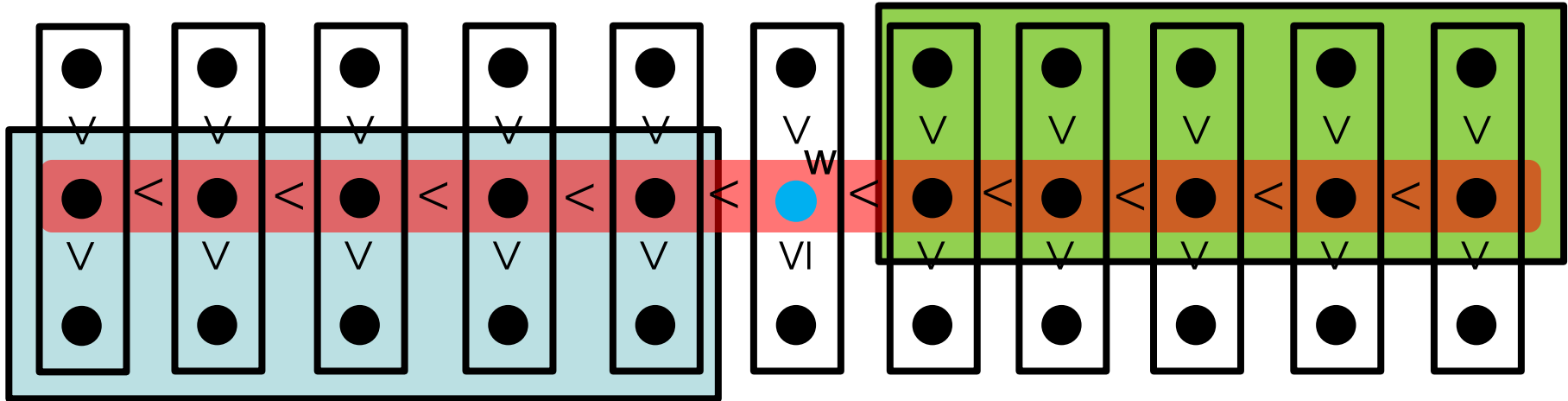
- $|S_{<}(w)| \geq 2 \binom{n}{6} = \frac{n}{3}$
- $|S_{>}(w)| \geq 2 \binom{n}{6} = \frac{n}{3}$.



$$\frac{n}{3} \leq |S_{<}(w)|, |S_{>}(w)| \leq \frac{2n}{3}$$

So, what is the running time?

Asymptotic Running Time?



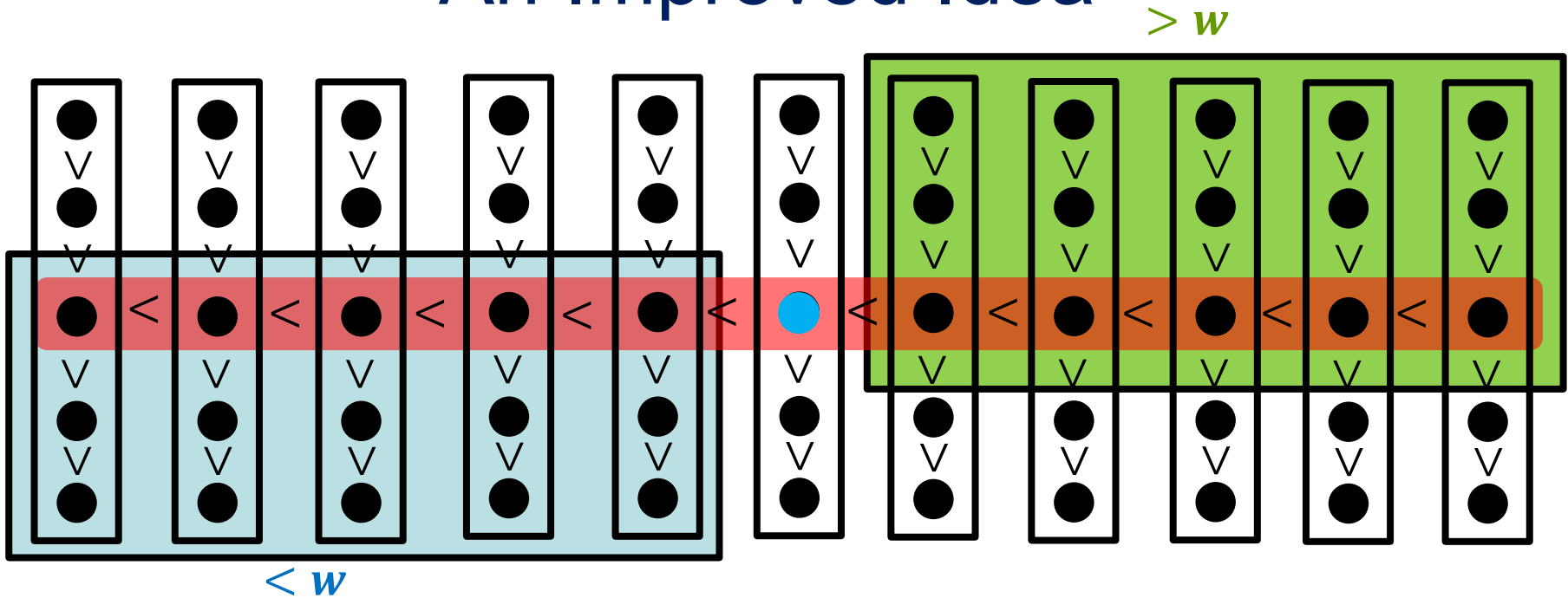
- If $k \leq |S_{<}(w)|$, output $Sel(S_{<}(w), k)$
- Else if $k \leq |S_{<}(w)| + |S_{=}(w)|$, output w
- Else output $Sel(S_{>}(w), k - |S_{<}(w)| - |S_{=}(w)|)$

$O(n \log n)$ again?
So, what is the point?

Where $\frac{n}{3} \leq |S_{<}(w)|, |S_{>}(w)| \leq \frac{2n}{3}$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

An Improved Idea



Partition into $n/5$ sets. Sort each set and set $w = \text{Sel}(\text{midpoints}, n/10)$

- $|S_{<}(w)| \geq 3 \left(\frac{n}{10}\right) = \frac{3n}{10}$
 - $|S_{>}(w)| \geq 3 \left(\frac{n}{10}\right) = \frac{3n}{10}$
- $\frac{3n}{10} \leq |S_{<}(w)|, |S_{>}(w)| \leq \frac{7n}{10}$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) \Rightarrow T(n) = O(n)$$


An Improved Idea

```

Sel(S, k) {
  n ← |S|
  If (n < ??) return ??
  Partition S into n/5 sets of size 5
  Sort each set of size 5 and let M be the set of medians, so
  |M|=n/5
  Let w=Sel(M,n/10)
  For i=1 to n{
    If  $x_i < w$  add x to  $S_{<}(w)$ 
    If  $x_i > w$  add x to  $S_{>}(w)$ 
    If  $x_i = w$  add x to  $S_{=}(w)$ 
  }
  If ( $k \leq |S_{<}(w)|$ )
    return Sel( $S_{<}(w)$ , k)
  else if ( $k \leq |S_{<}(w)| + |S_{=}(w)|$ )
    return w;
  else
    return Sel( $S_{>}(w)$ ,  $k - |S_{<}(w)| - |S_{=}(w)|$ )
}

```

We can maintain each set in an array



D&C Summary

Idea:

“Two halves are better than a whole”

- if the base algorithm has super-linear complexity.

“If a little's good, then more's better”

- repeat above, recursively
- Applications: Many.
 - Binary Search, Merge Sort, (Quicksort),
 - Root of a Function
 - Closest points,
 - Integer multiplication
 - Median
 - Matrix Multiplication

In-class Exercise

Prove that every amount of postage of 12 cents or more can be formed using just 4-cents and 5-cents stamps.

For example $12=4+4+4$.

Approximation Algorithms

How to deal with NP-complete Problem

Many of the important problems in real world are NP-complete.

SAT, Set Cover, Graph Coloring, TSP, Max IND Set, Vertex Cover, ...

So, we cannot find optimum solutions in polynomial time.

What to do instead?

- Find optimum solution of special cases (e.g., random inputs)
- Find near optimum solution in the worst case

Approximation Algorithm

Polynomial-time Algorithms with a guaranteed approximation ratio.

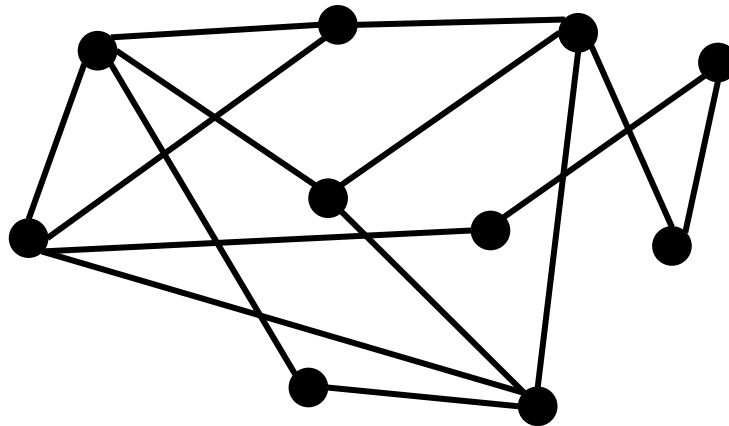
$$\alpha = \frac{\text{Cost of computed solution}}{\text{Cost of the optimum}}$$

worst case over all instances.

Goal: For each NP-hard problem find an approximation algorithm with the best possible approximation ratio.

Vertex Cover

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



Greedy Algorithm?

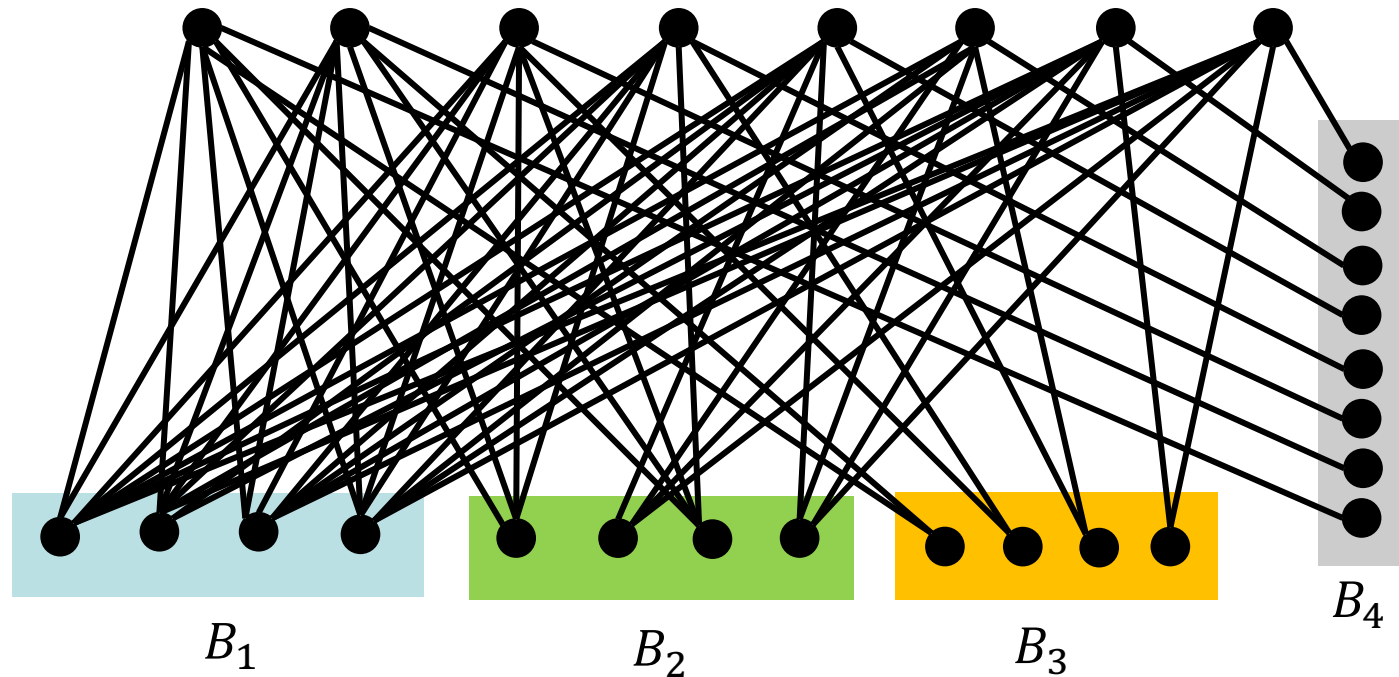
Greedy algorithms are typically used in practice to find a (good) solution to NP-hard problems

Strategy (1): Iteratively, include a vertex that covers most new edges

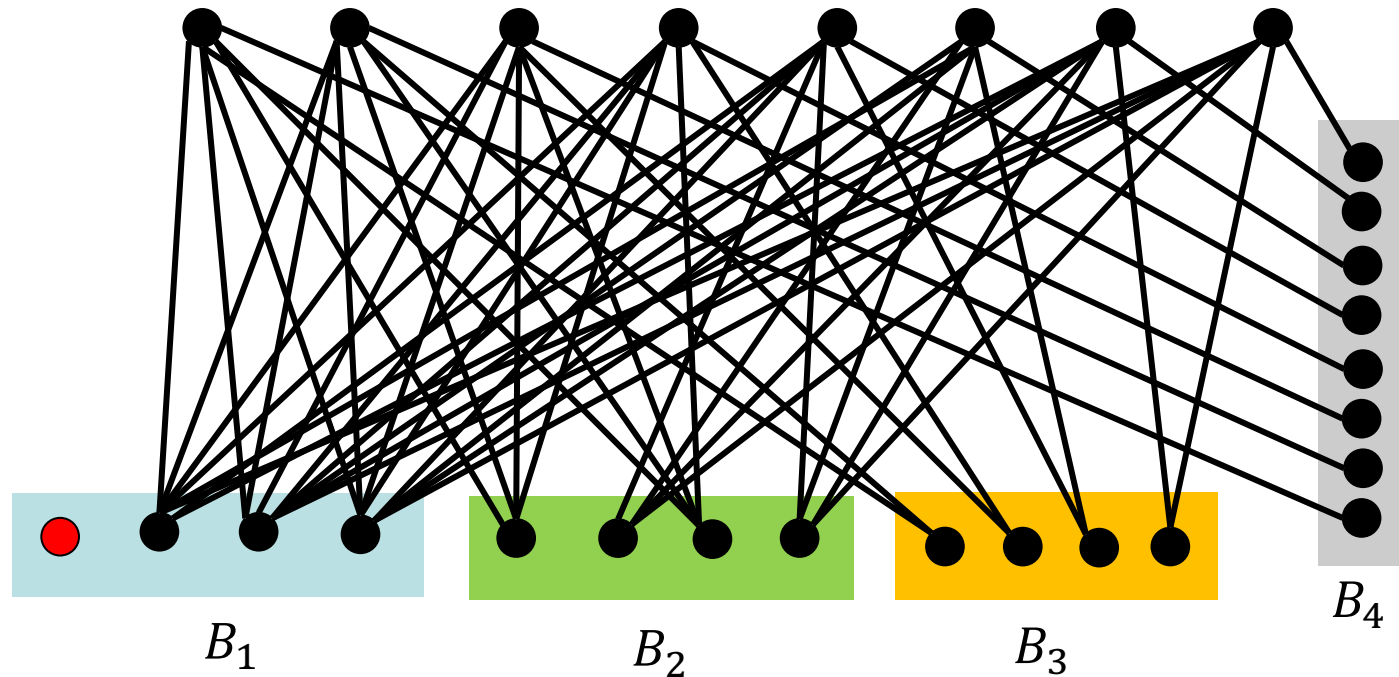
Q: Does this give an optimum solution?

A: No,

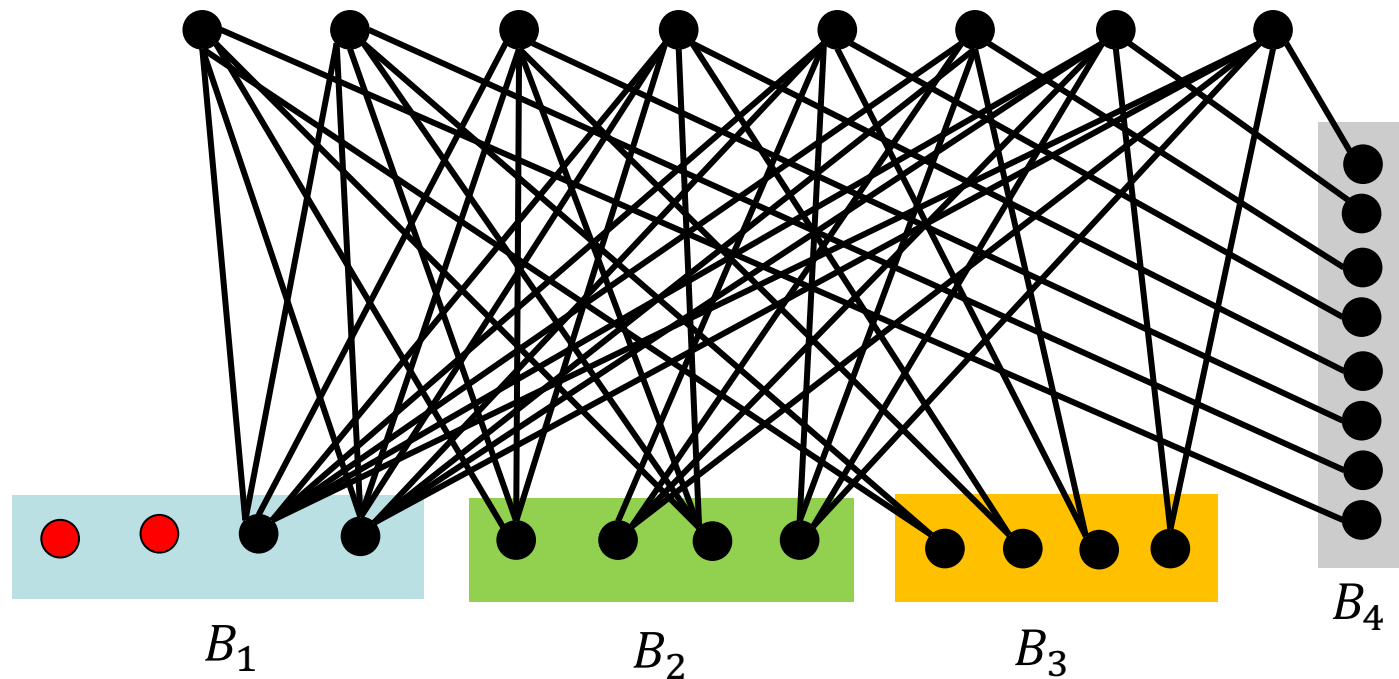
Greedy (1): Pick vertex that covers the most



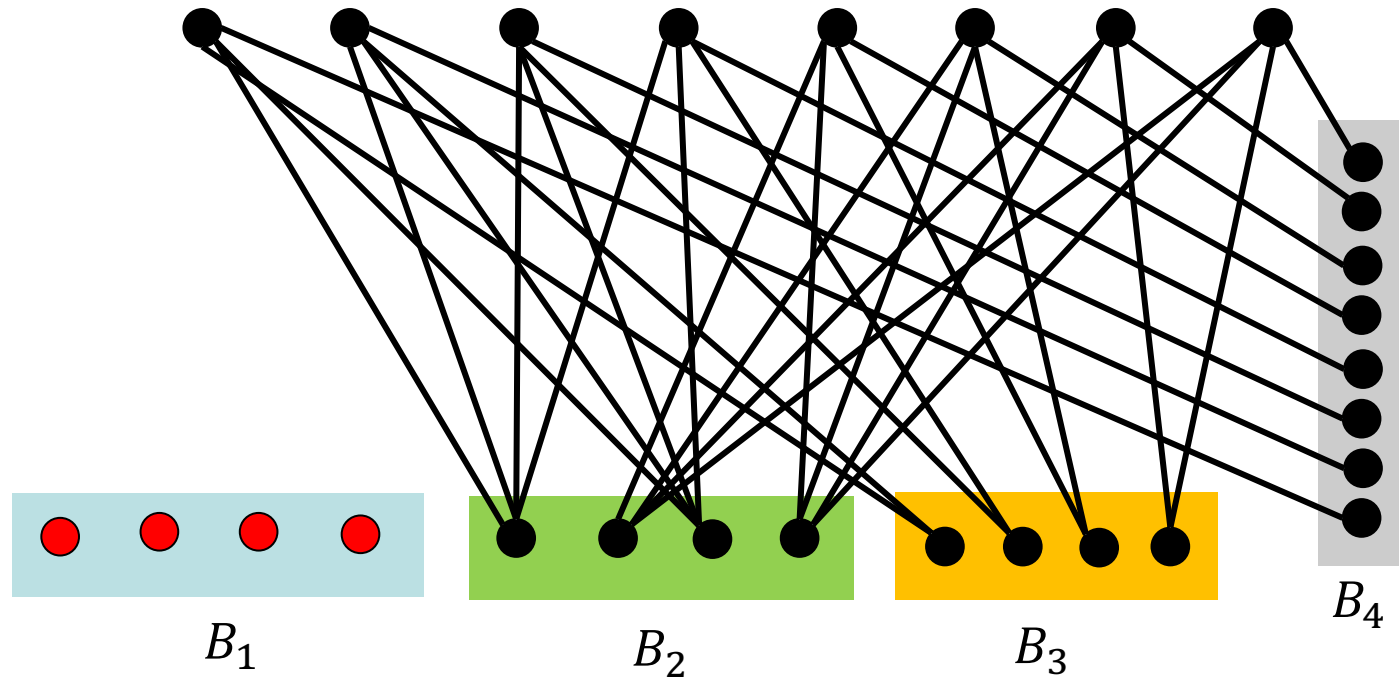
Greedy (1): Pick vertex that covers the most



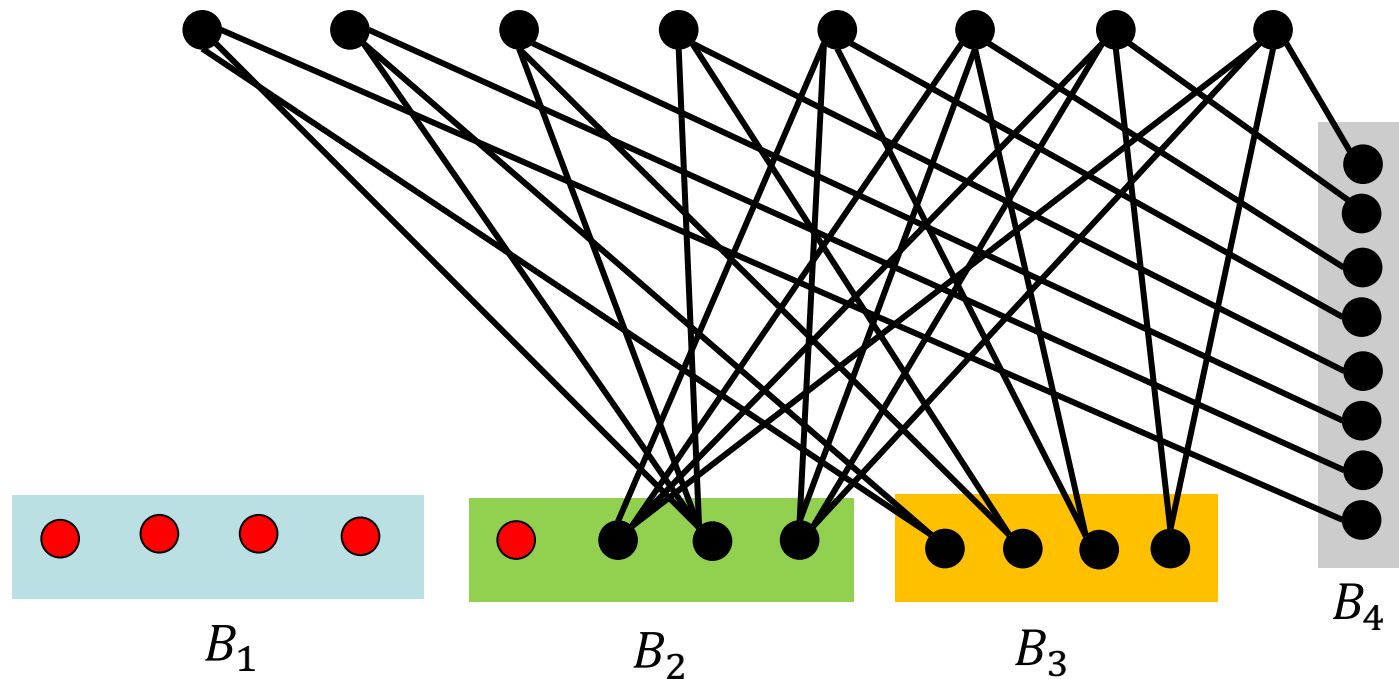
Greedy (1): Pick vertex that covers the most



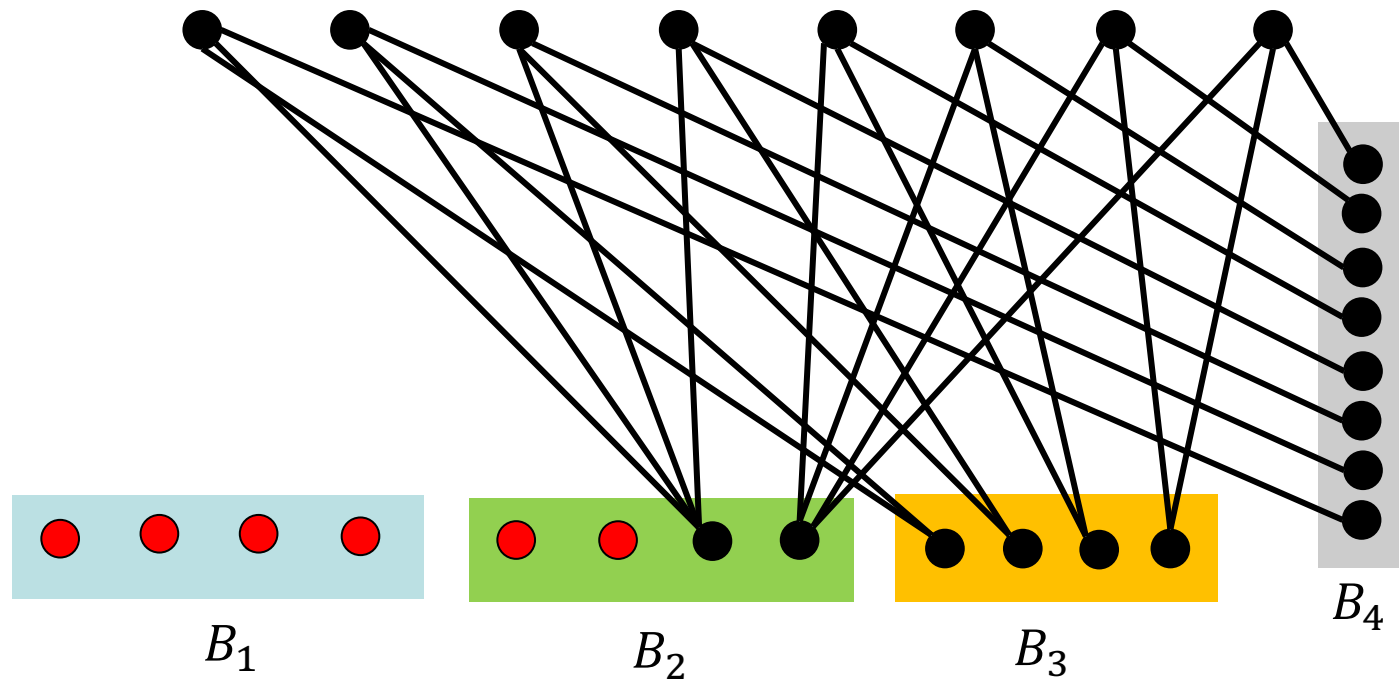
Greedy (1): Pick vertex that covers the most



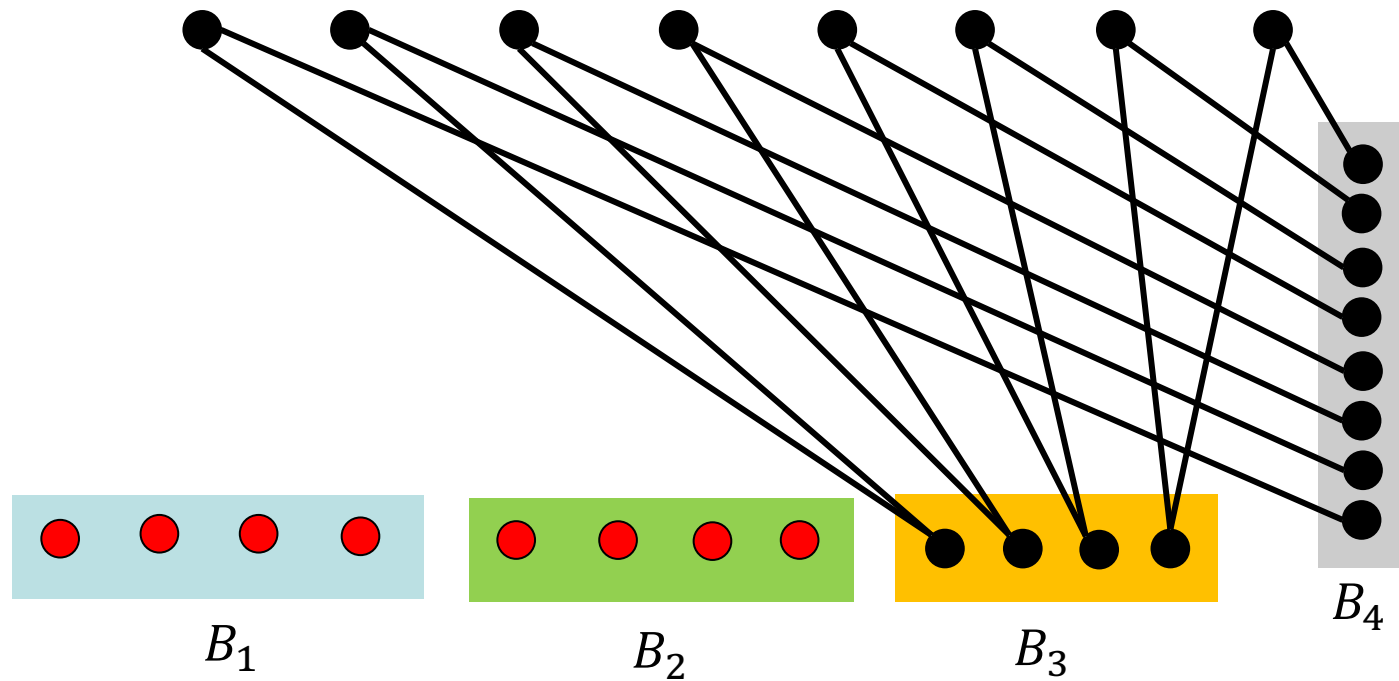
Greedy (1): Pick vertex that covers the most



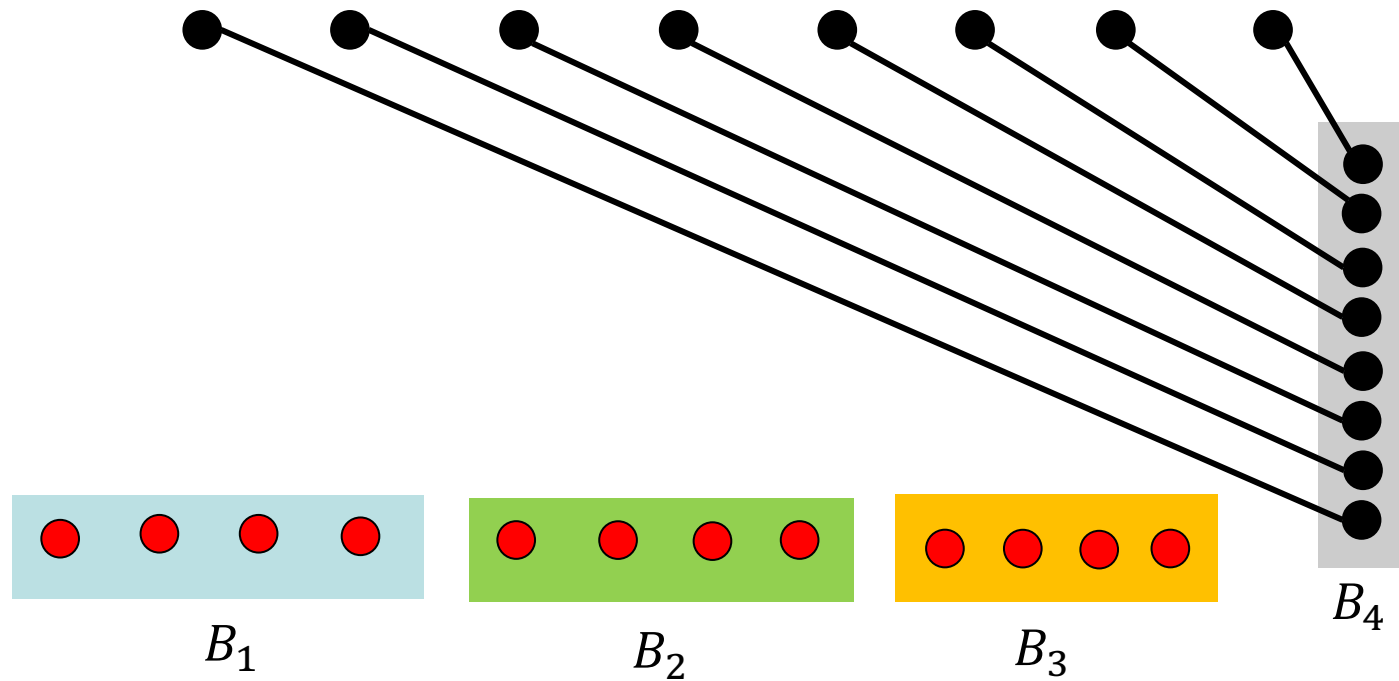
Greedy (1): Pick vertex that covers the most



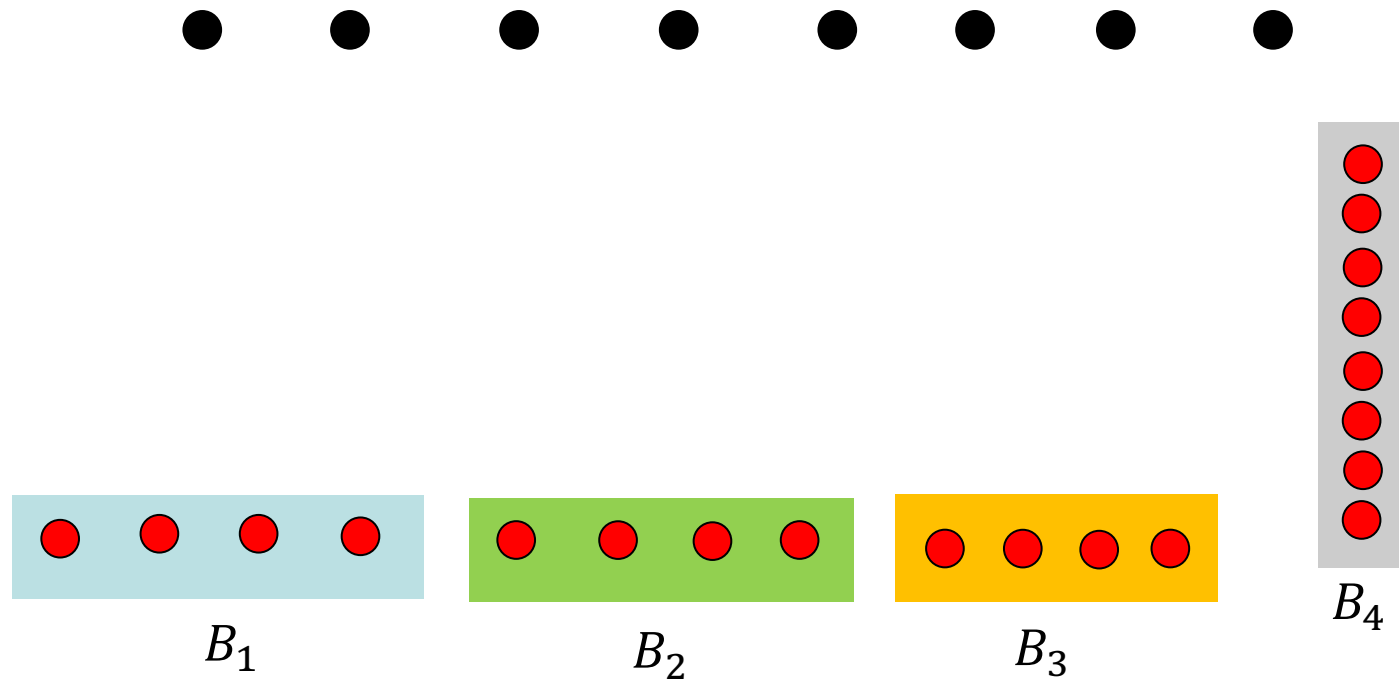
Greedy (1): Pick vertex that covers the most



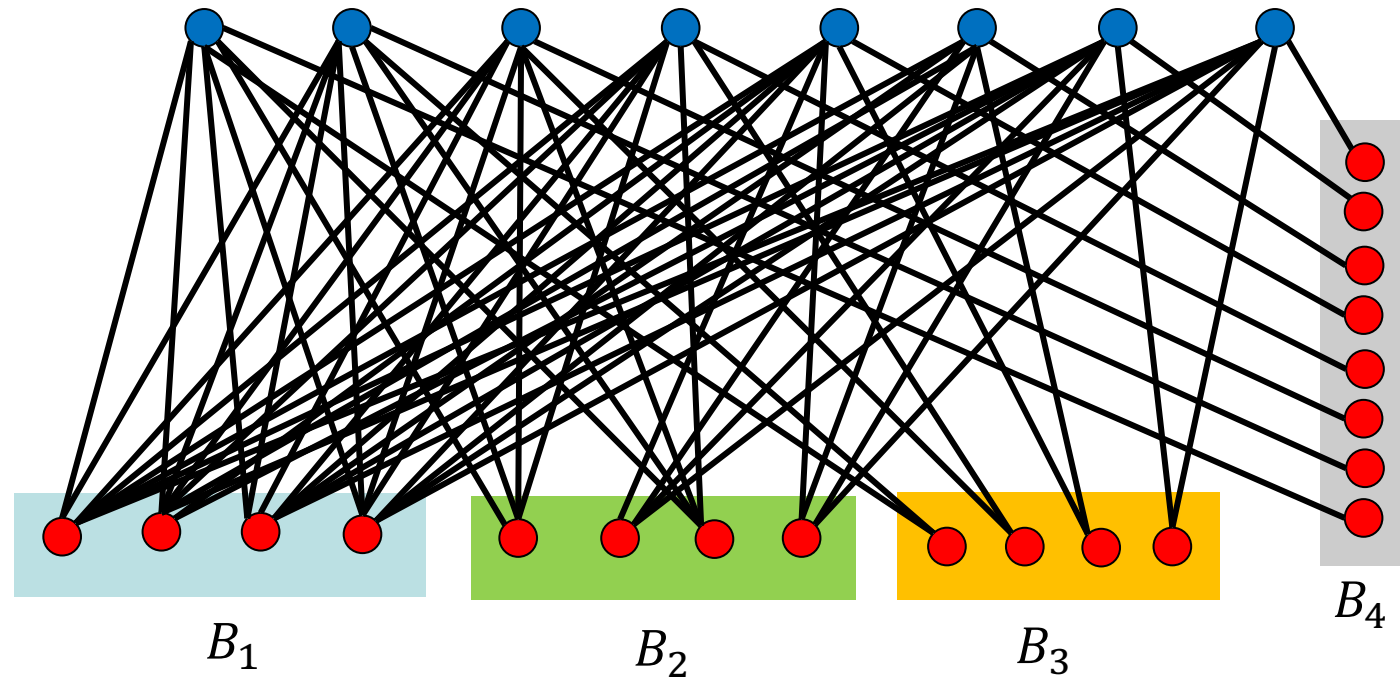
Greedy (1): Pick vertex that covers the most



Greedy (1): Pick vertex that covers the most



Greedy (1): Pick vertex that covers the most

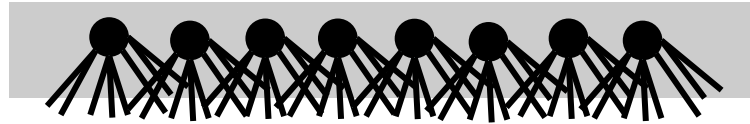


Greedy Vertex cover = 20

OPT Vertex cover = 8

Greedy (1): Pick vertex that covers the most

n vertices. Each vertex has one edge into each B_i



B_n



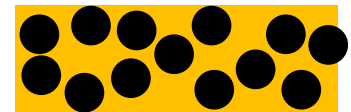
B_{n-1}

.....



$|B_i| = n/i$

.....



B_1

Each vertex in B_i has i edges to top

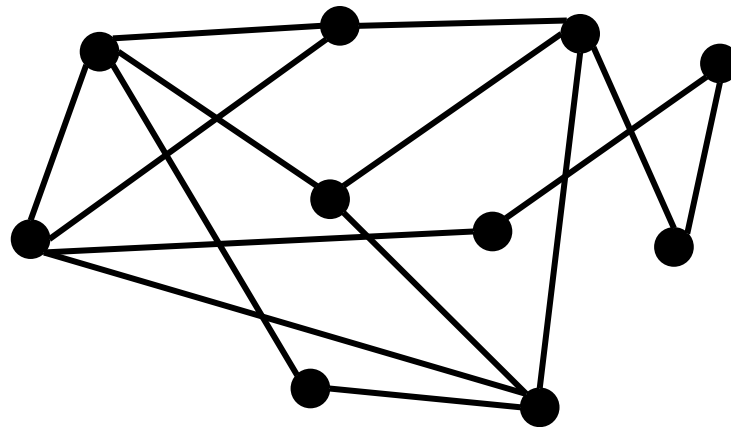
Greedy pick bottom vertices = $n + \frac{n}{2} + \frac{n}{3} + \dots + 1 \approx n \ln n$

OPT pick top vertices = n

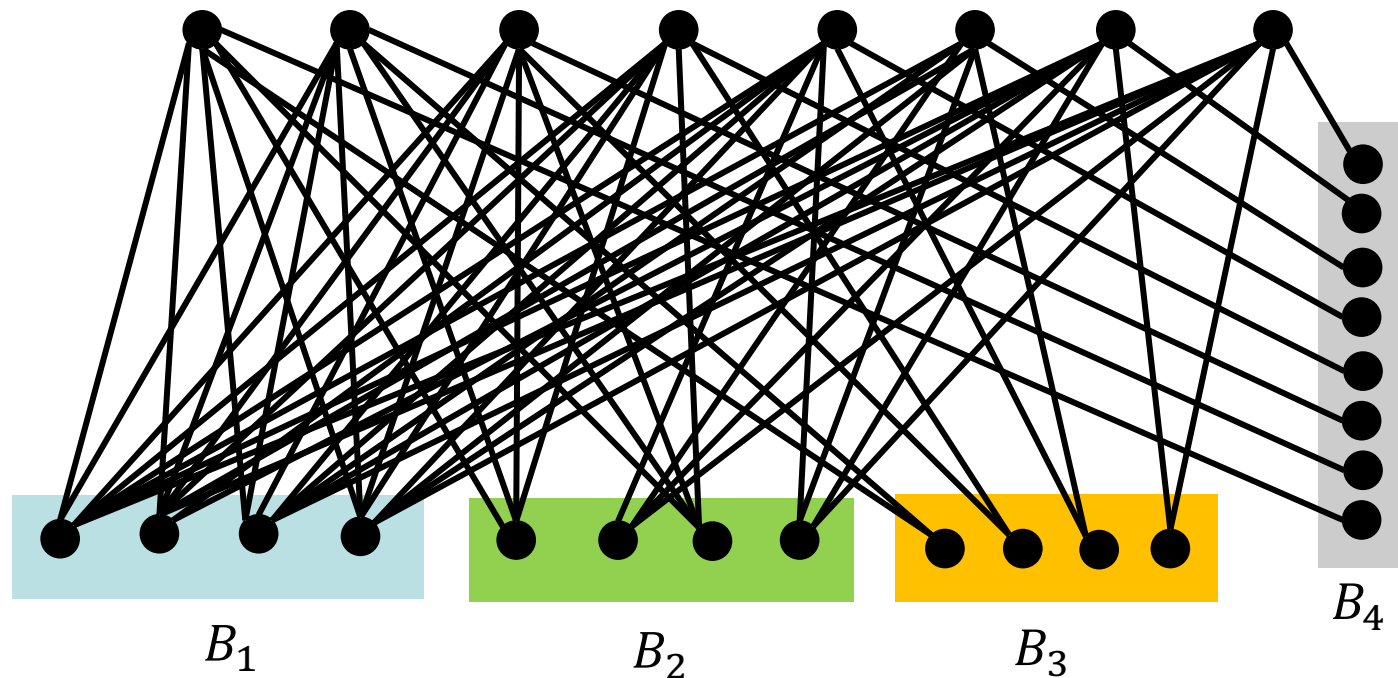
A Different Greedy Rule

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

Vertex cover = 6



Greedy 2: Pick Both endpoints of an uncovered edge



Greedy vertex cover = 16

OPT vertex cover = 8

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.