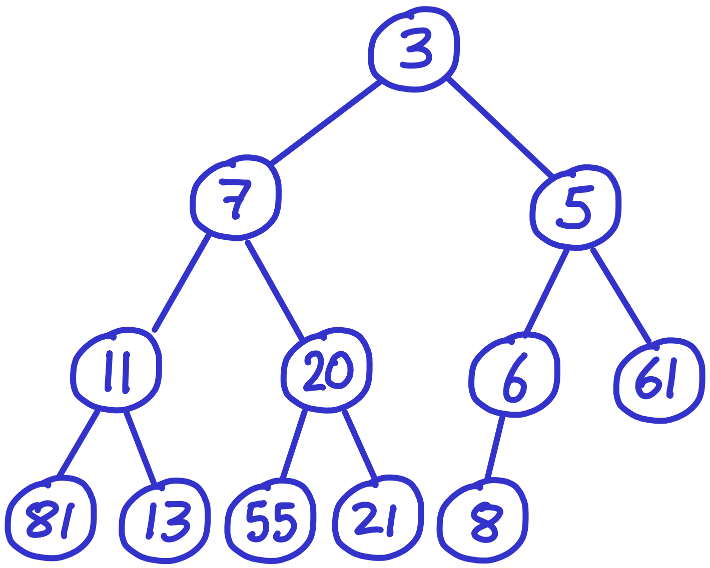
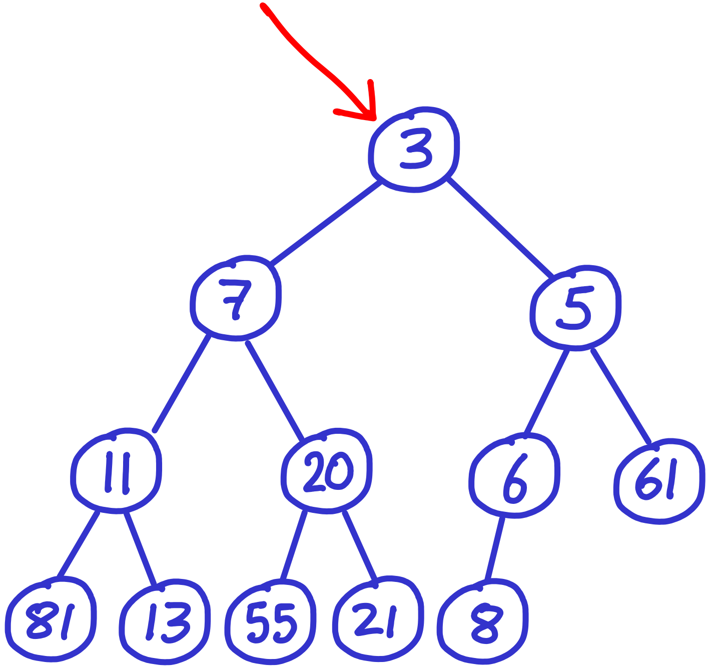
HEAPS

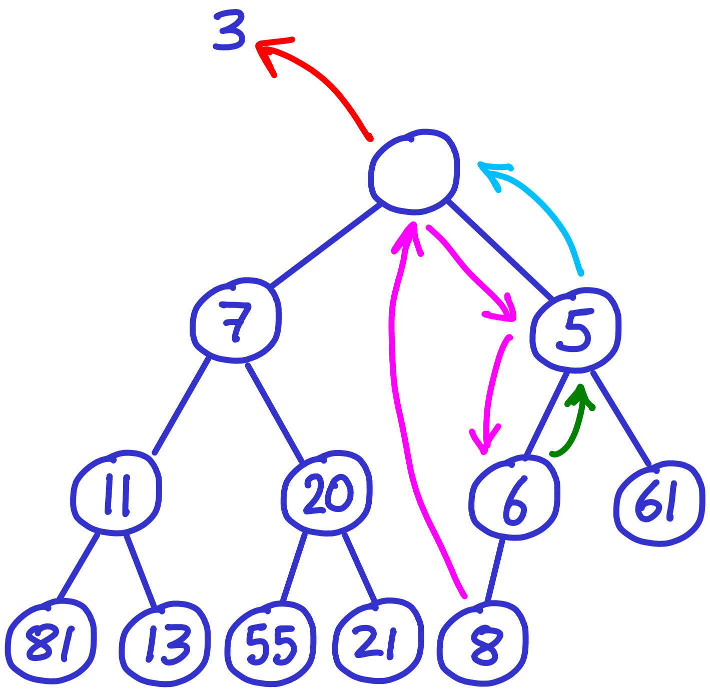


HEAPS



REPORT MIN: $O(1)$

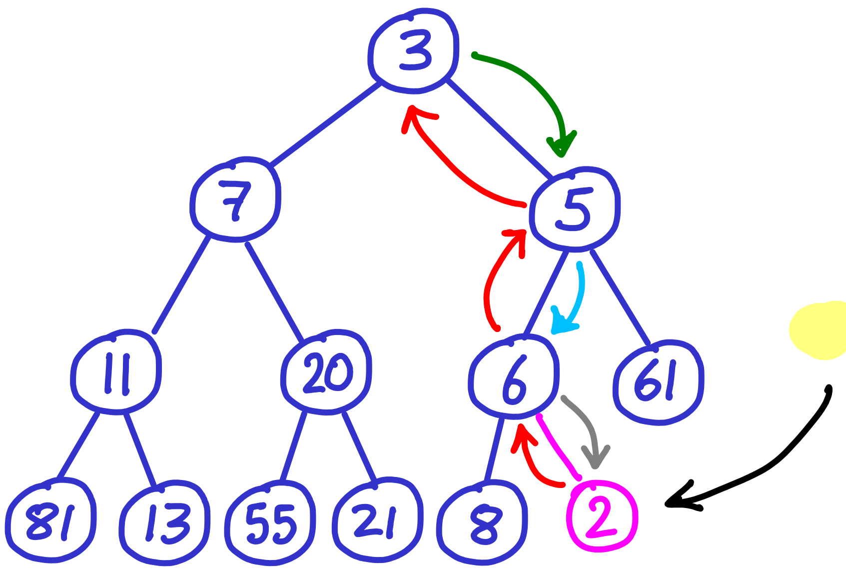
HEAPS



REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

HEAPS

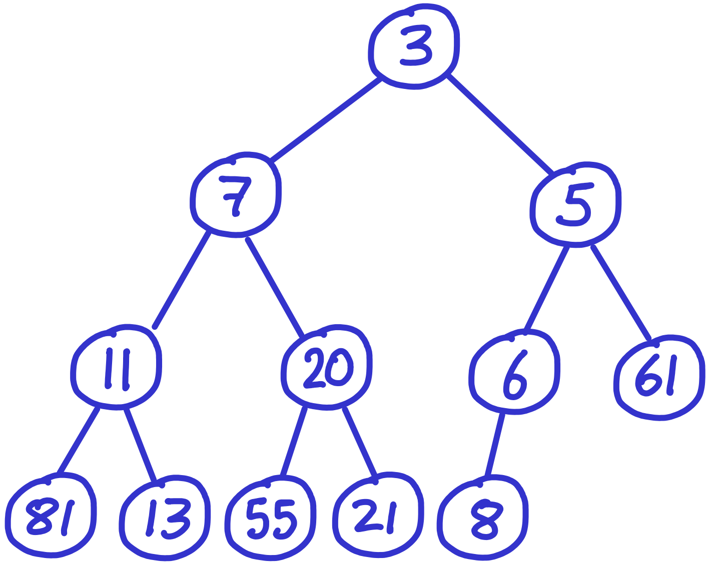


REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$

HEAPS



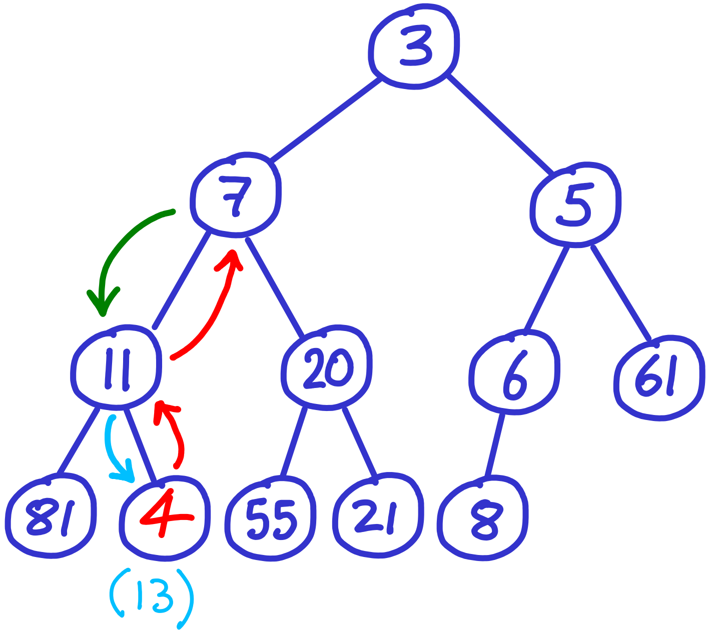
REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$ - but $O(n)$ for n inserts

↳ amortized

HEAPS



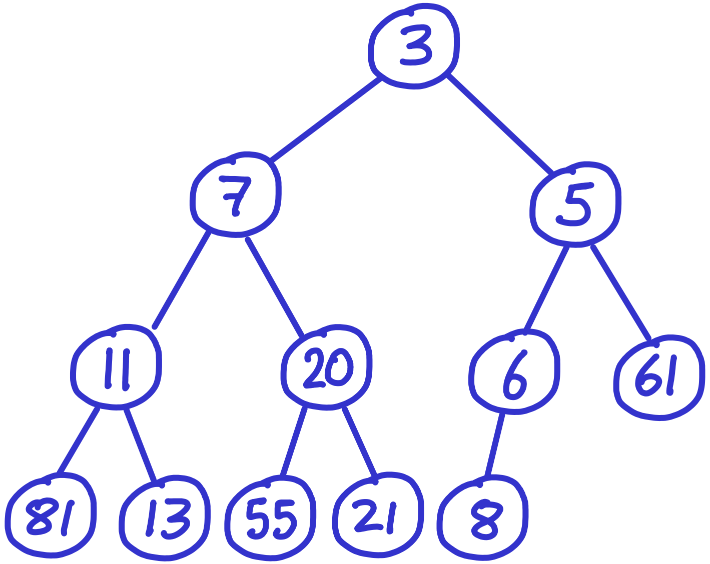
REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$ - but $O(n)$ for n inserts
↳ amortized

DECREASE KEY: $O(\log n)$

HEAPS



REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

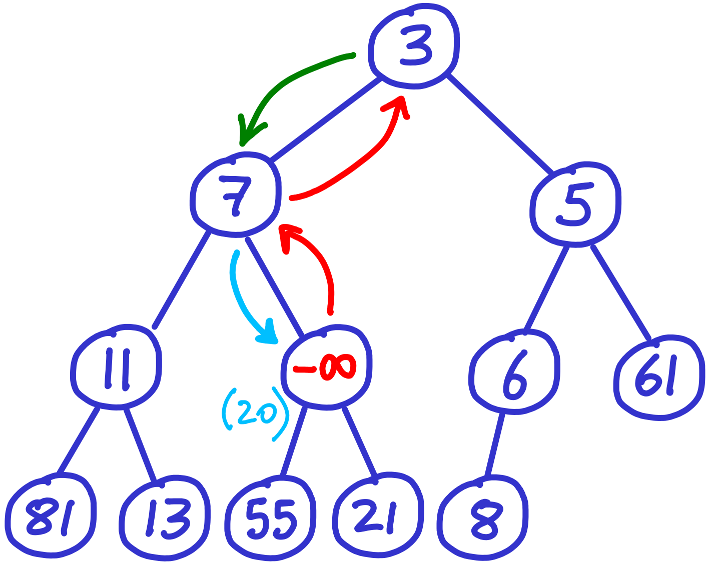
INSERT: $O(\log n)$ - but $O(n)$ for n inserts

↳ amortized

DECREASE KEY: $O(\log n)$

DELETE: $O(\log n)$ Why?

HEAPS



REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

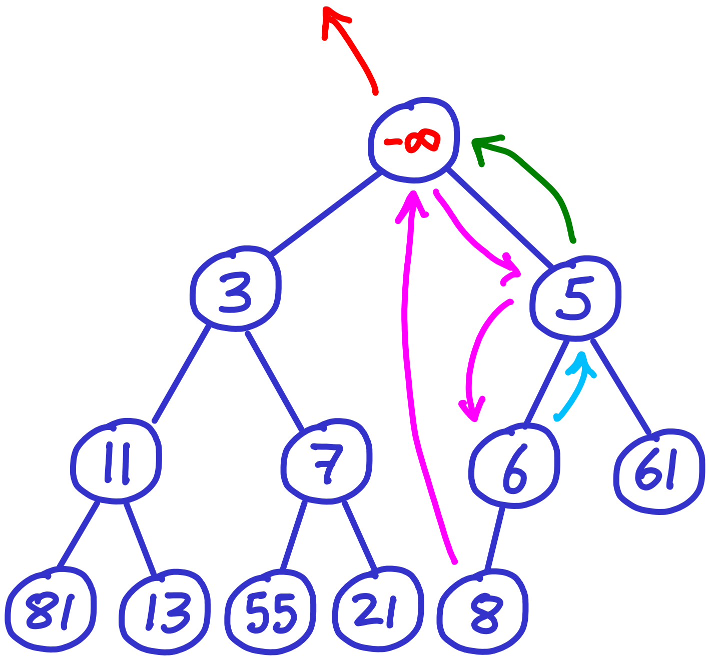
INSERT: $O(\log n)$ – but $O(n)$ for n inserts

↪ amortized

DECREASE KEY: $O(\log n)$

DELETE: $O(\log n)$ – decrease to $-\infty$

HEAPS



REPORT MIN: $O(1)$

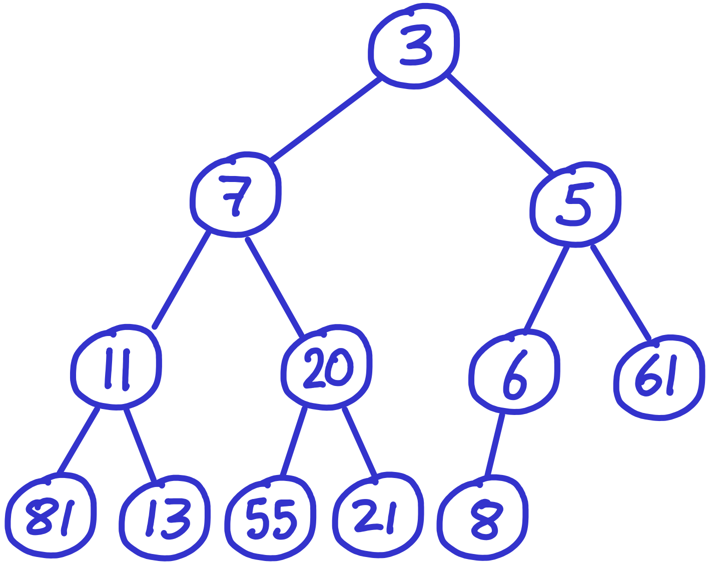
EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$ – but $O(n)$ for n inserts
↳ amortized

DECREASE KEY: $O(\log n)$

DELETE: $O(\log n)$ – decrease to $-\infty$
& extract

HEAPS



REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$ – but $O(n)$ for n inserts

amortized

DECREASE KEY: $O(\log n)$

DELETE: $O(\log n)$ – decrease to $-\infty$ & extract

UNION: $O(n)$ – rebuild

(assuming both are large)

BINOMIAL HEAPS

REGULAR HEAPS

REPORT MIN: $O(1)$

EXTRACT MIN: $O(\log n)$

INSERT: $O(\log n)$ - but $O(n)$ for n inserts
↳ amortized

DECREASE KEY: $O(\log n)$

DELETE: $O(\log n)$ - decrease to $-\infty$ & extract

$O(\log n)$ ← UNION: $O(n)$ - rebuild
(assuming both are large)

BINOMIAL HEAPS	REGULAR HEAPS	
$O(\log n)$ *but really $O(1)$	REPORT MIN: $O(1)$	
	EXTRACT MIN: $O(\log n)$	
	INSERT: $O(\log n)$	– but $O(n)$ for n inserts ↳ amortized
	DECREASE KEY: $O(\log n)$	
	DELETE: $O(\log n)$	– decrease to $-\infty$ & extract
$O(\log n)$	UNION: $O(n)$	– rebuild (assuming both are large)

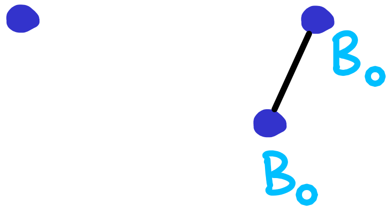
BINOMIAL HEAP = a collection of BINOMIAL TREES

BINOMIAL TREES

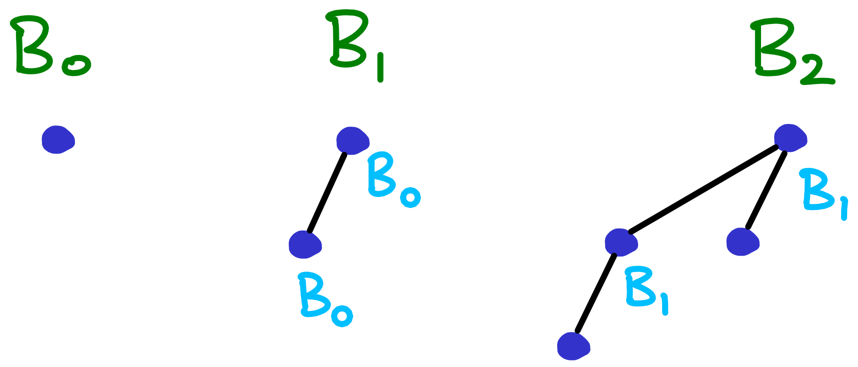
B_0

BINOMIAL TREES

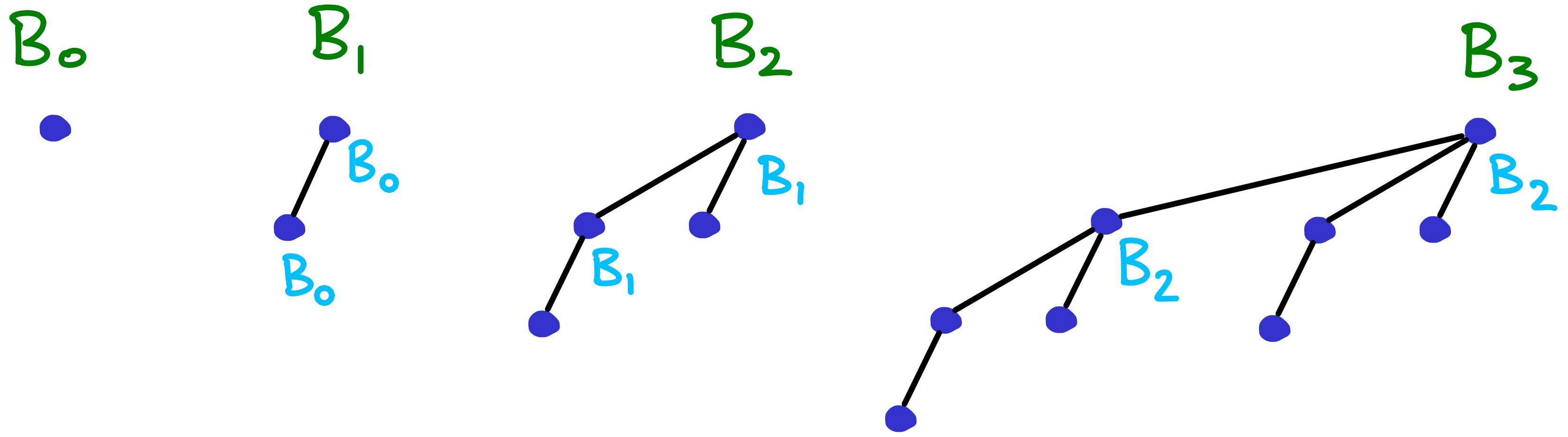
B_0 B_1



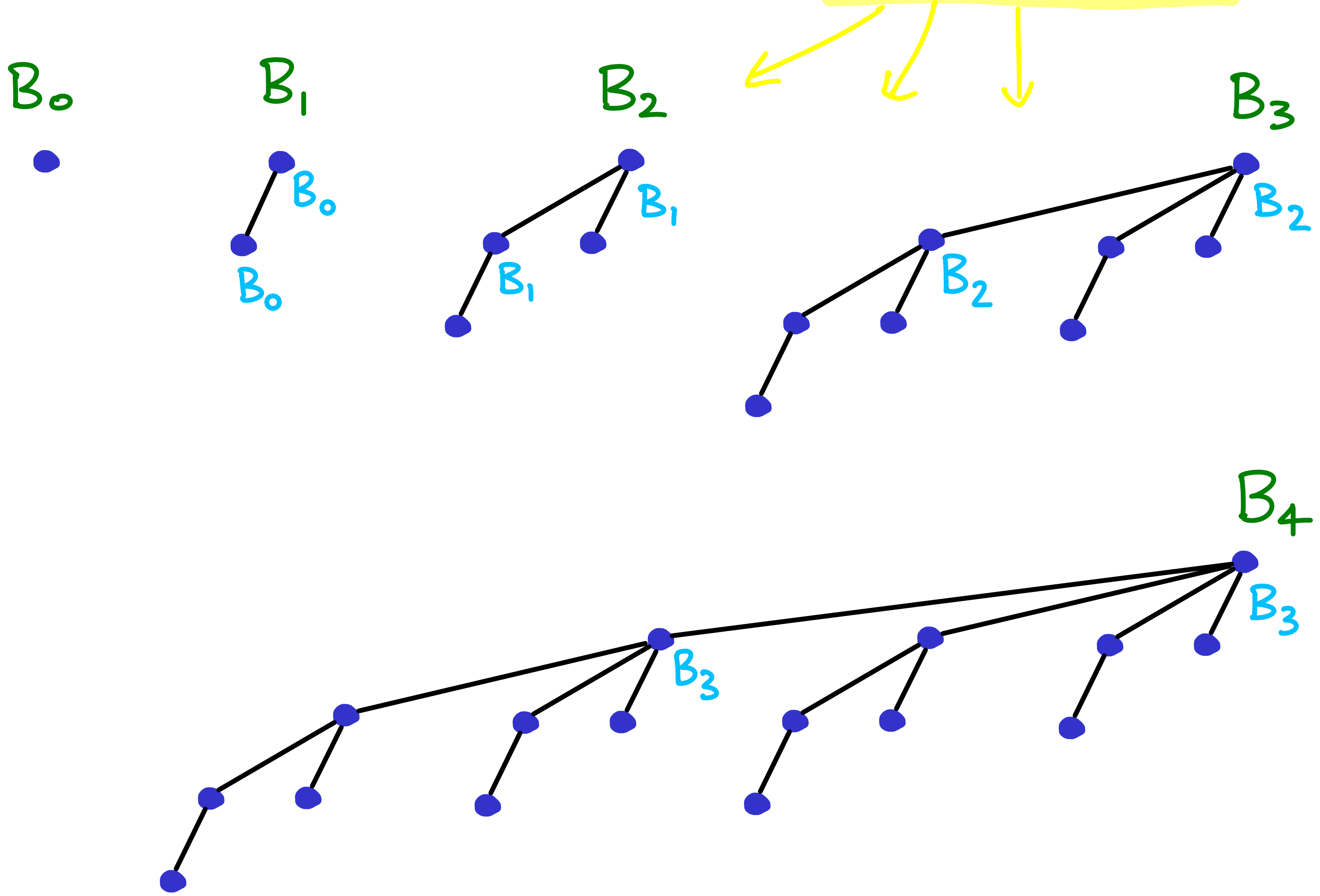
BINOMIAL TREES



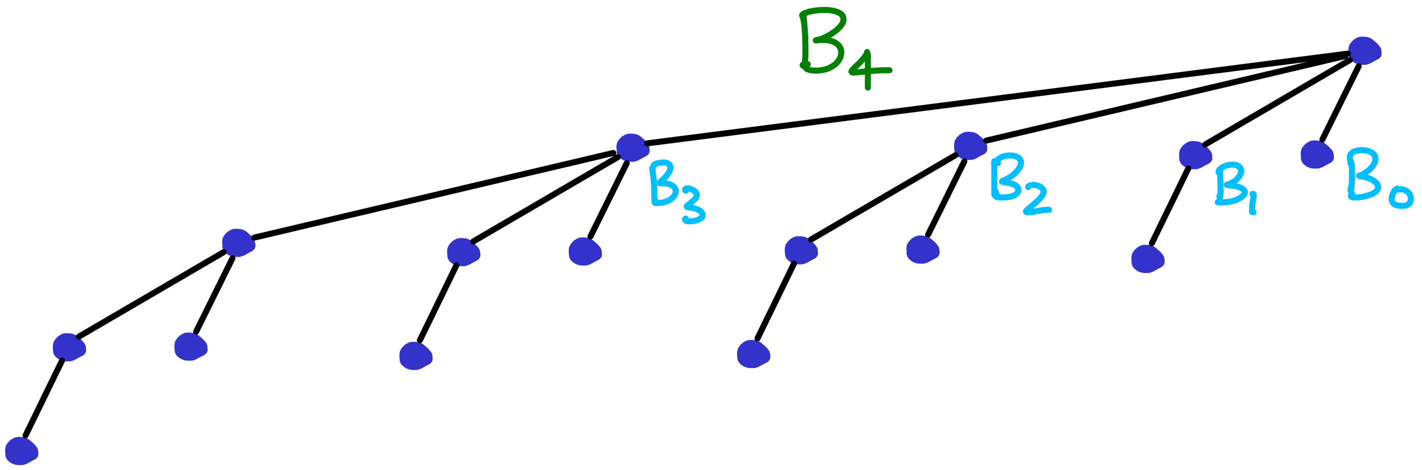
BINOMIAL TREES

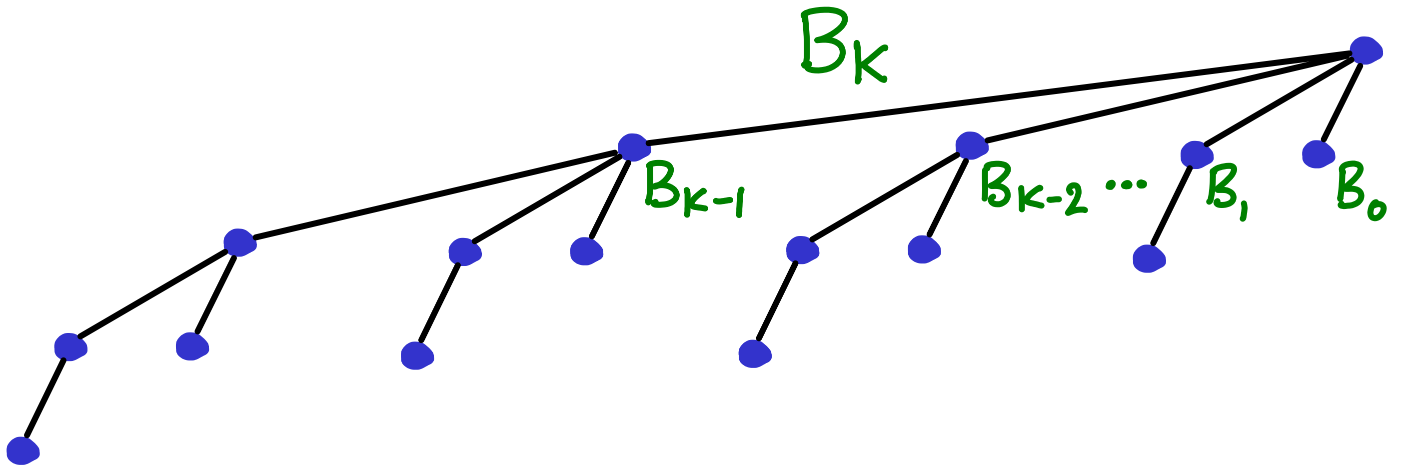


BINOMIAL TREES

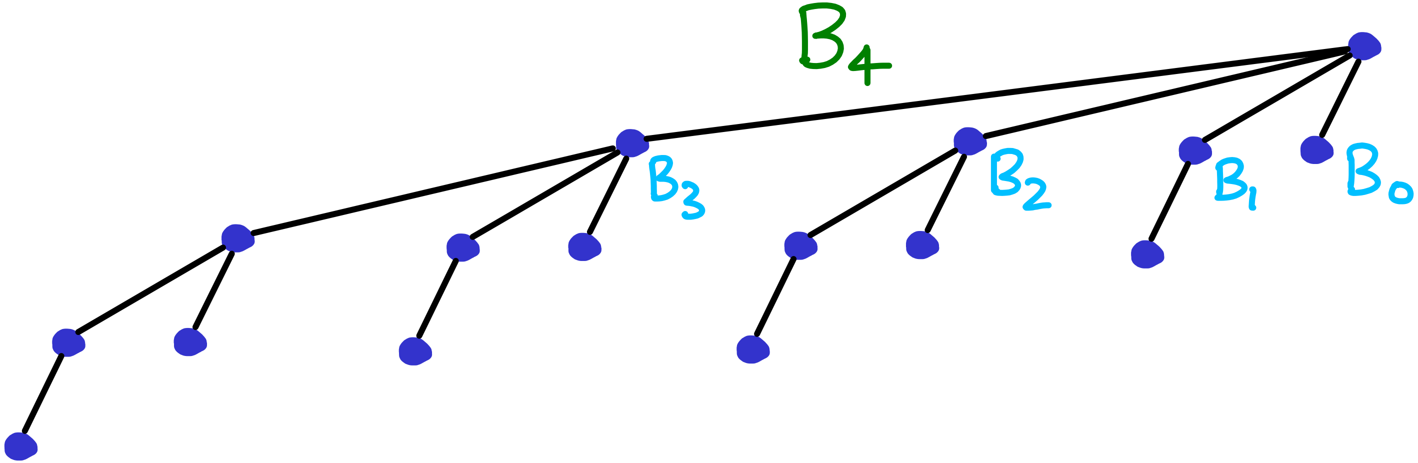


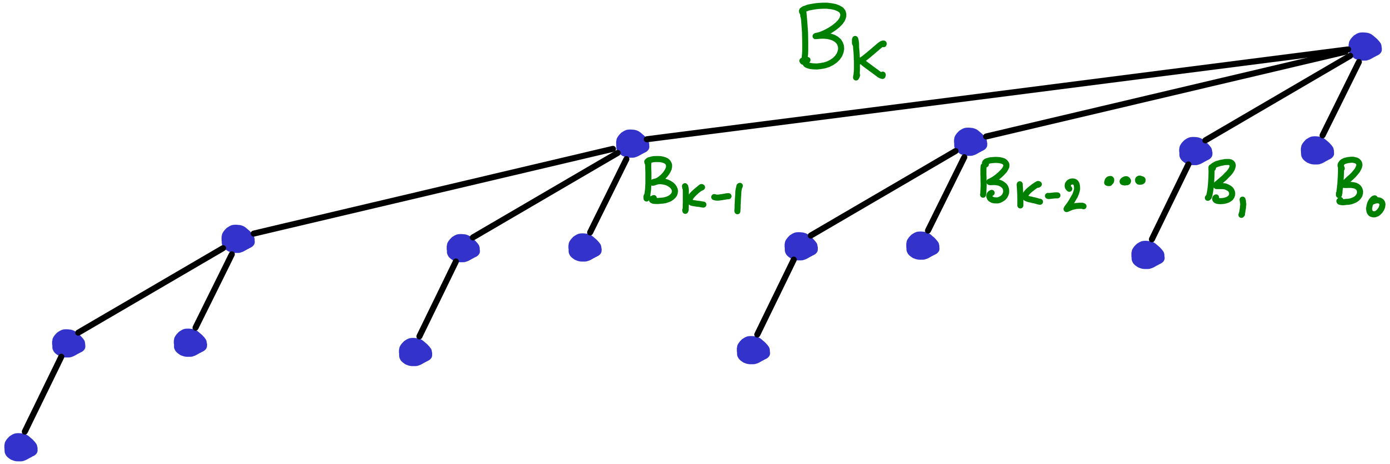
BINOMIAL TREES

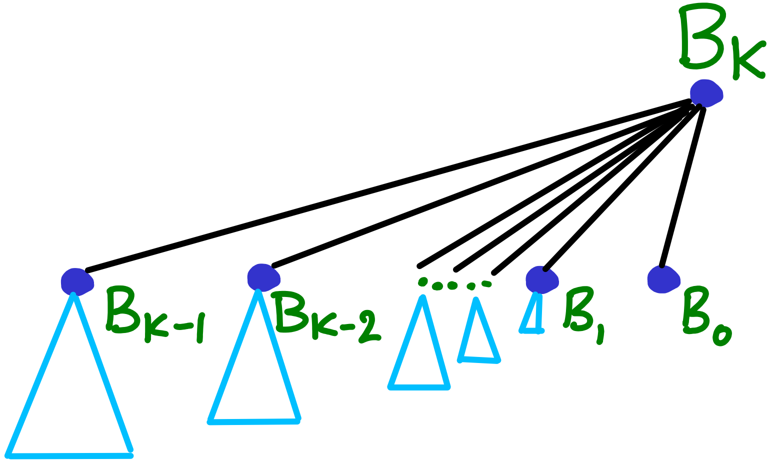




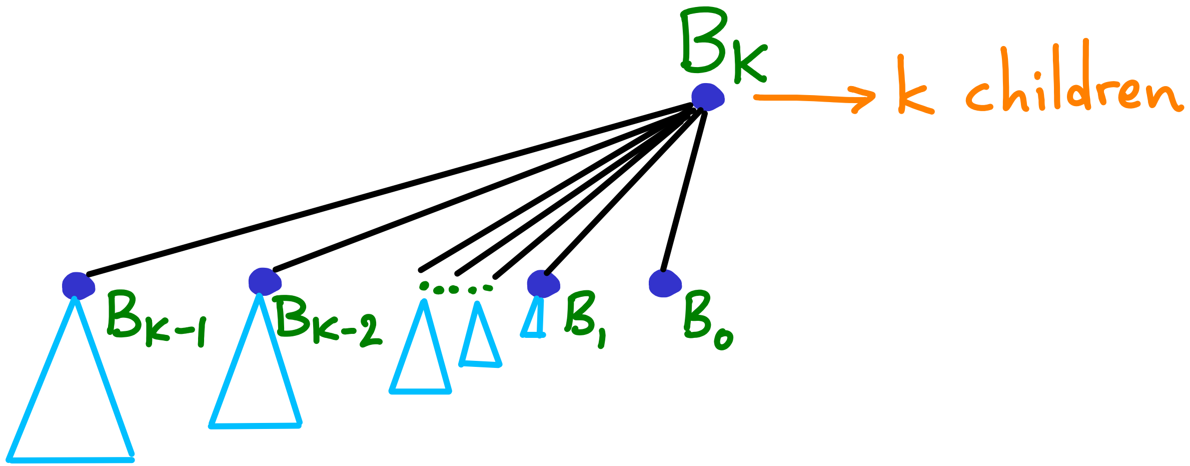
BINOMIAL TREES



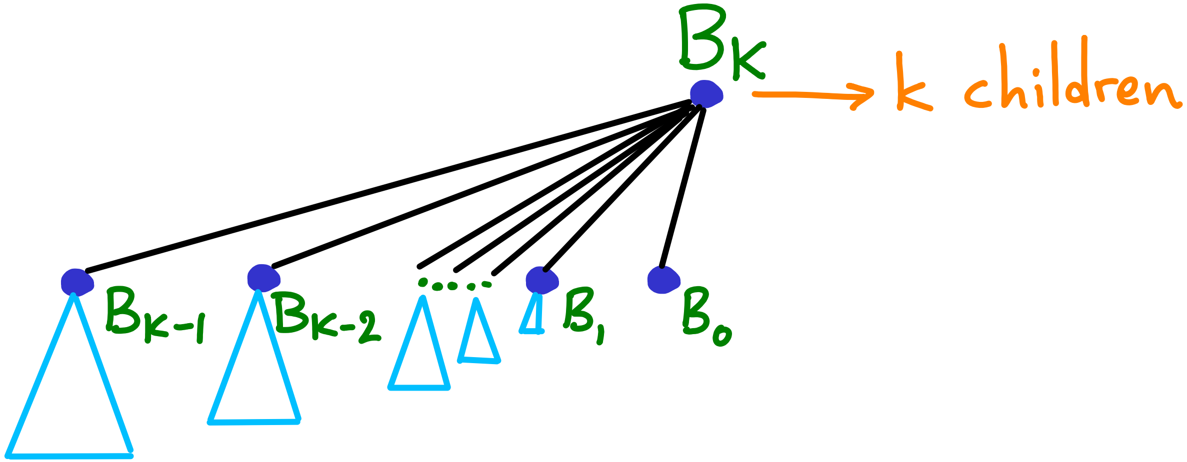




BINOMIAL TREE of degree k

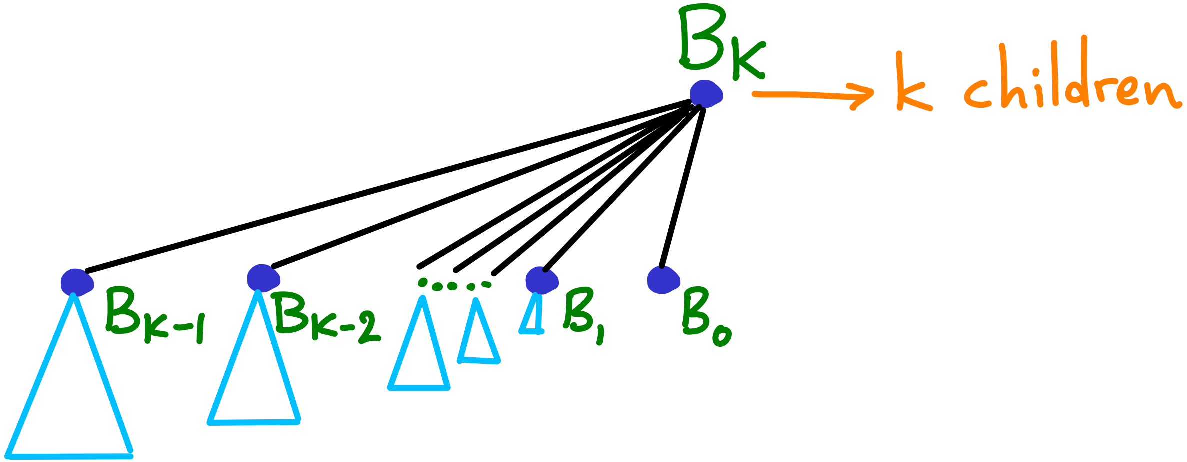


BINOMIAL TREE of degree k

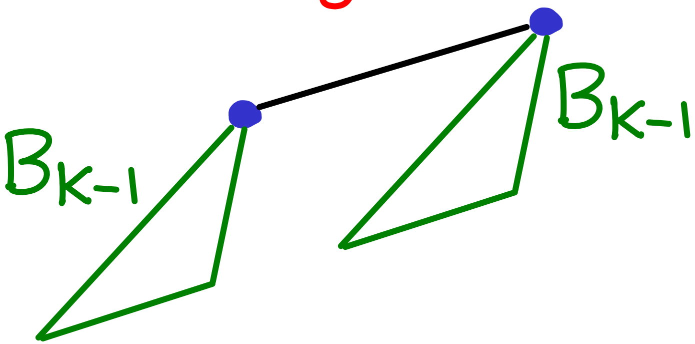


How many nodes?

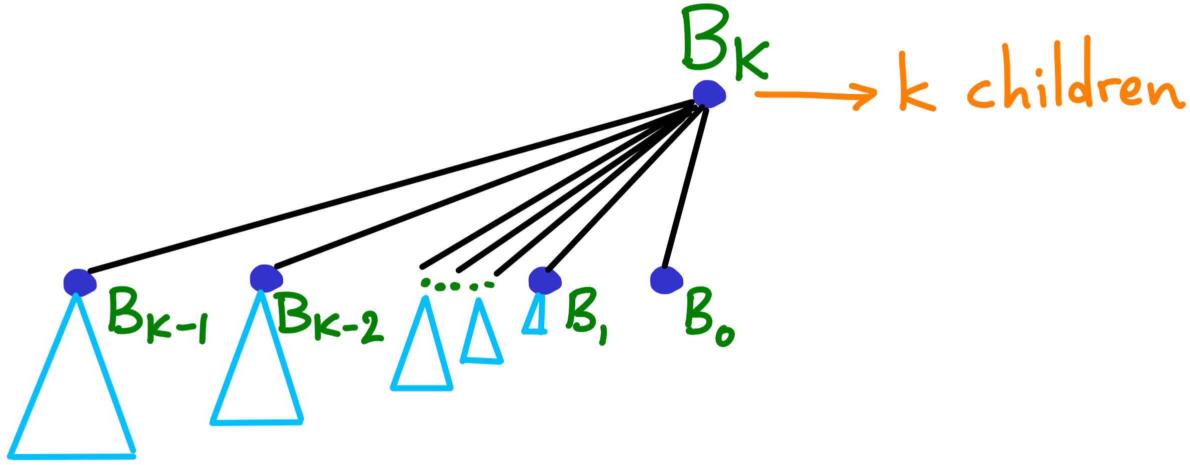
BINOMIAL TREE of degree k



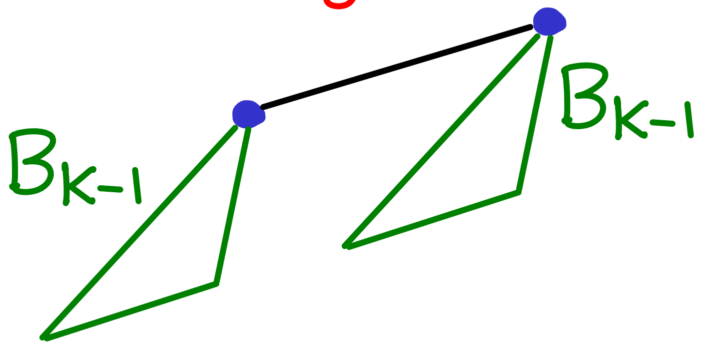
How many nodes?



BINOMIAL TREE of degree k

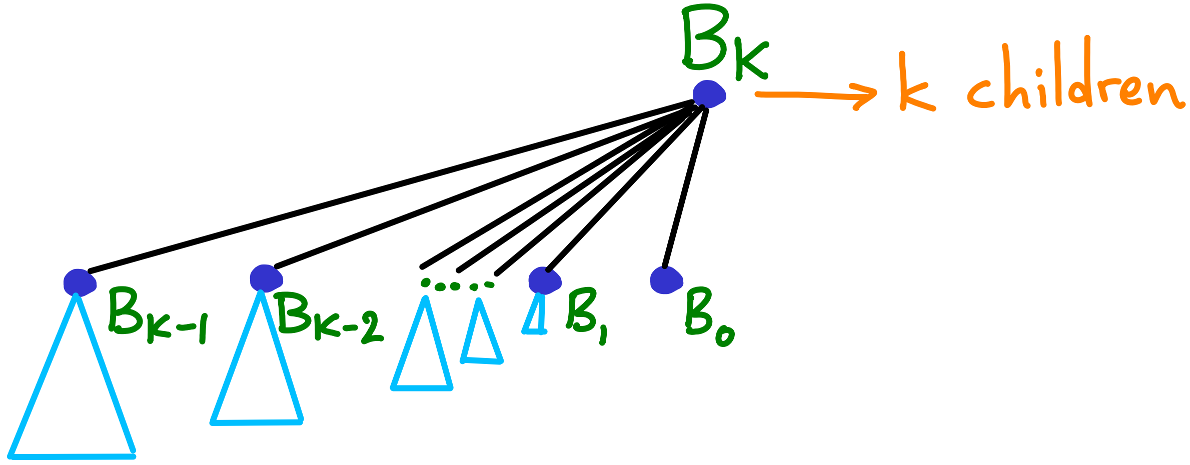


How many nodes?

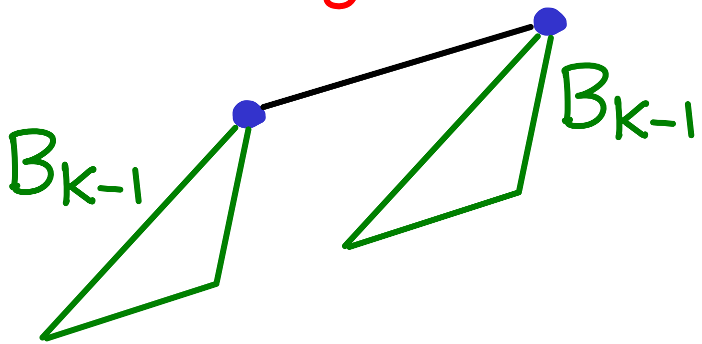


$$n = Size(k) = 2\,Size(k-1)$$

BINOMIAL TREE of degree k



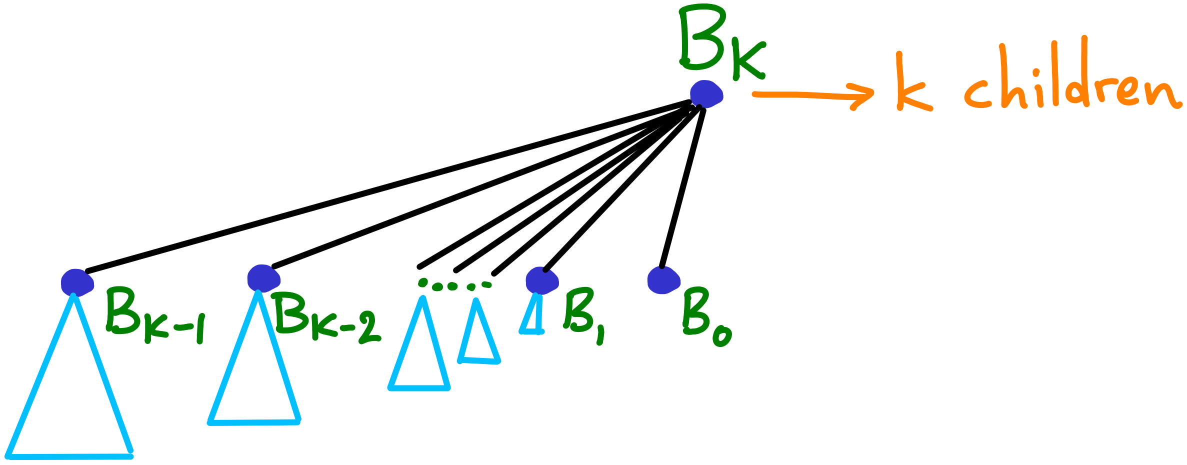
How many nodes?



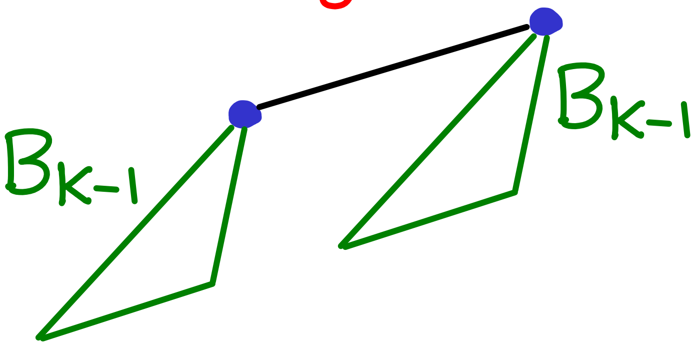
$$n = Size(k) = 2\,Size(k-1)$$

$$\longrightarrow 2^k$$

BINOMIAL TREE of degree k



How many nodes?

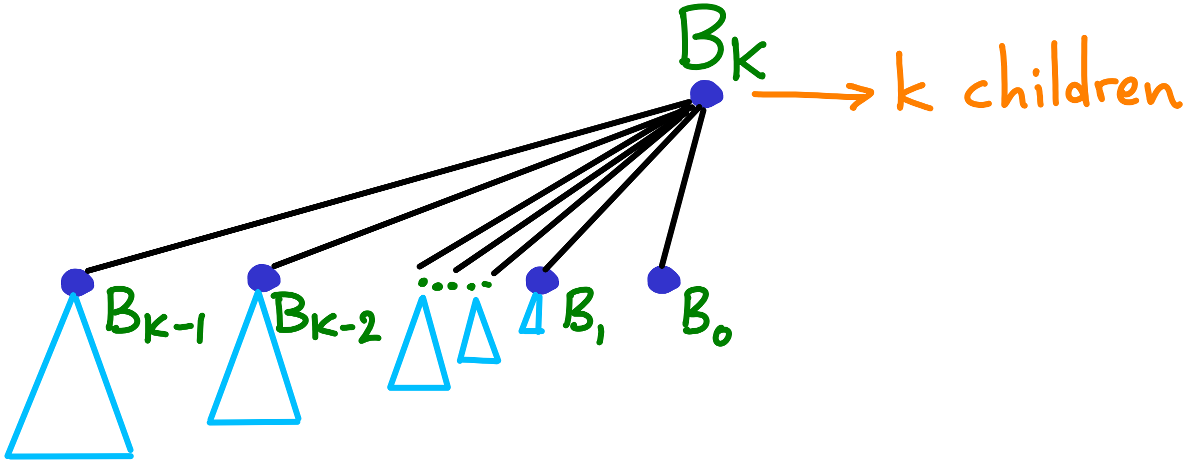


$$n = Size(k) = 2\,Size(k-1)$$

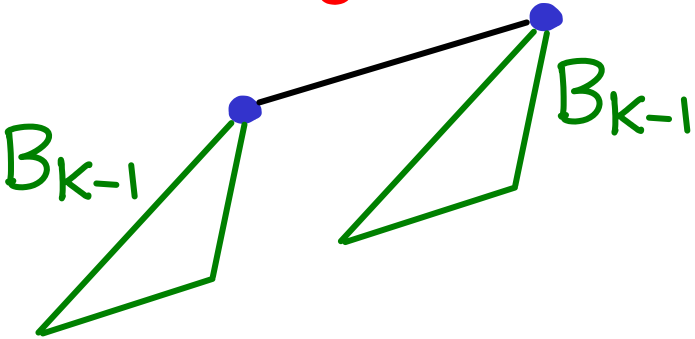
$$\longrightarrow 2^k$$

Height of B_k?

Binomial tree of degree k



How many nodes?



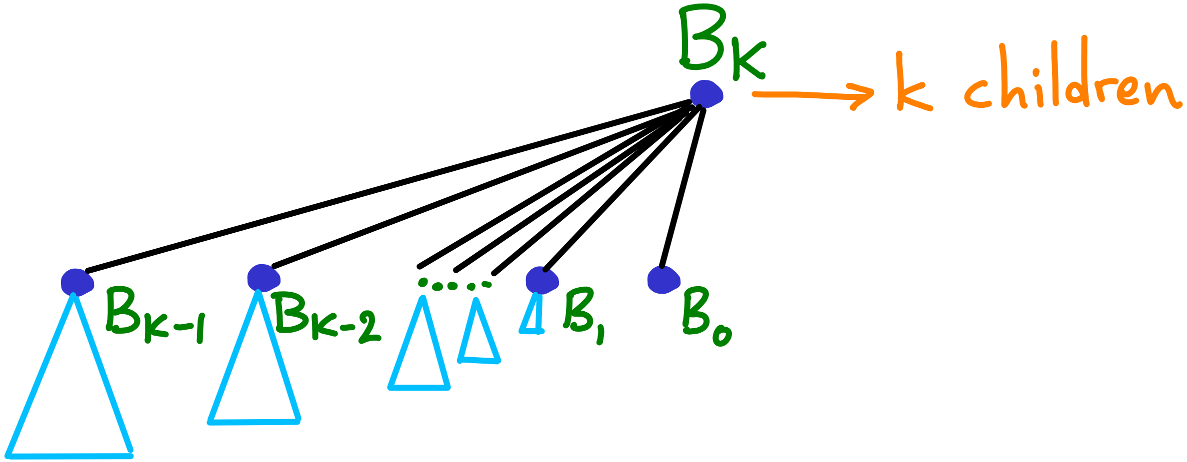
$H(k) =$ Height of B_k ?

$\hookrightarrow 1 + H(k-1) = k$ ... so?

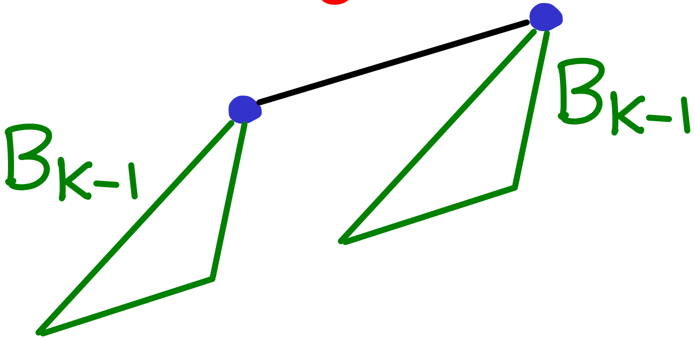
$n = \text{Size}(k) = 2\,\text{Size}(k-1)$

$\longrightarrow 2^k$

BINOMIAL TREE of degree k



How many nodes?



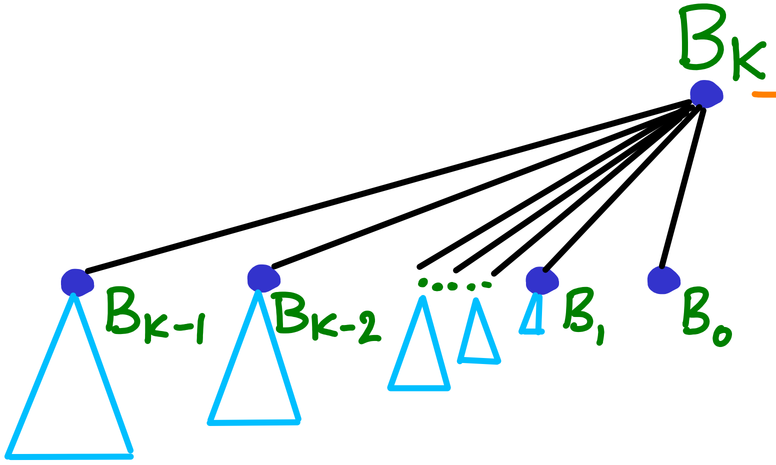
$H(k) =$ Height of B_k?

$\hookrightarrow 1 + H(k-1) = k = \log n$

$n = \text{Size}(k) = 2\,\text{Size}(k-1)$

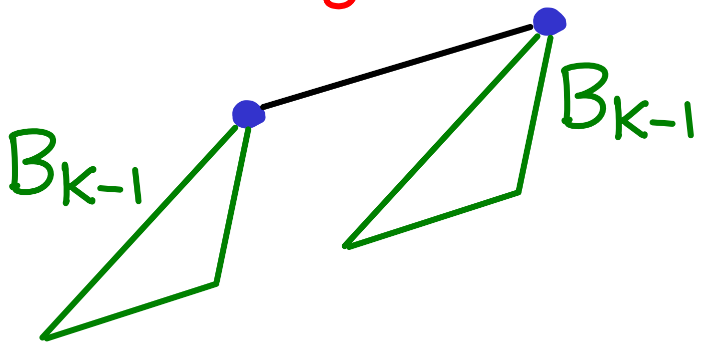
$\longrightarrow 2^k$

BINOMIAL TREE of degree k



$\rightarrow$ k children

How many nodes?



$H(k) =$ Height of B_k?

$\hookrightarrow 1 + H(k-1) = k = \log n$

$n = Size(k) = 2\, Size(k-1)$

$\rightarrow 2^k$

FYI: level i has $\binom{k}{i}$ nodes: a binomial coefficient

back to the BINOMIAL HEAP

- like regular heap, child > parent
- collection of binomial trees

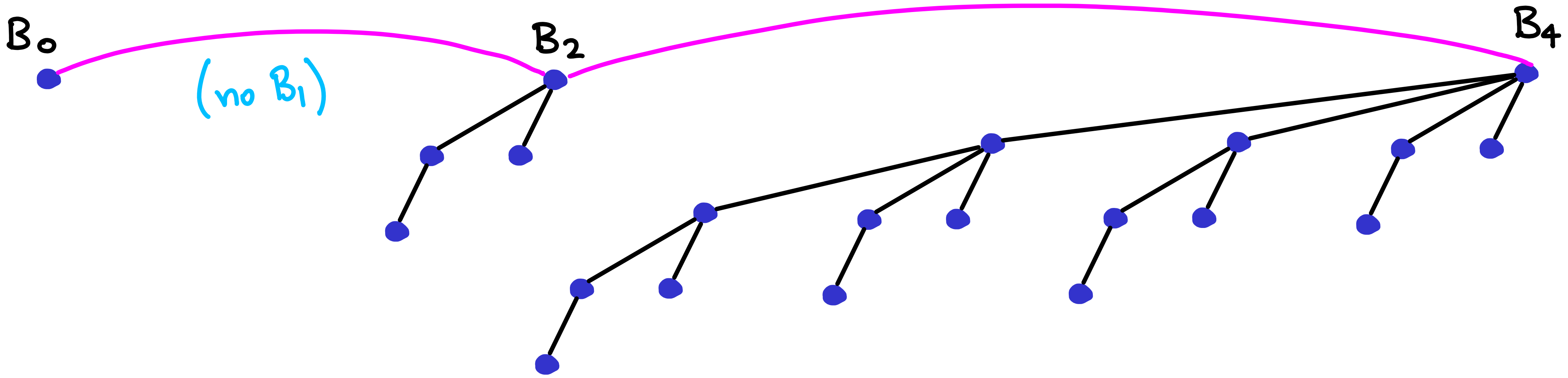
back to the BINOMIAL HEAP

- like regular heap, child > parent
- collection of binomial trees, but at most one for each degree

(max necessary degree is $\log n$)

back to the BINOMIAL HEAP

- like regular heap, child > parent
- collection of binomial trees, but at most one for each degree
  (max necessary degree is $\log n$)
- store roots in linked list, sorted by degree

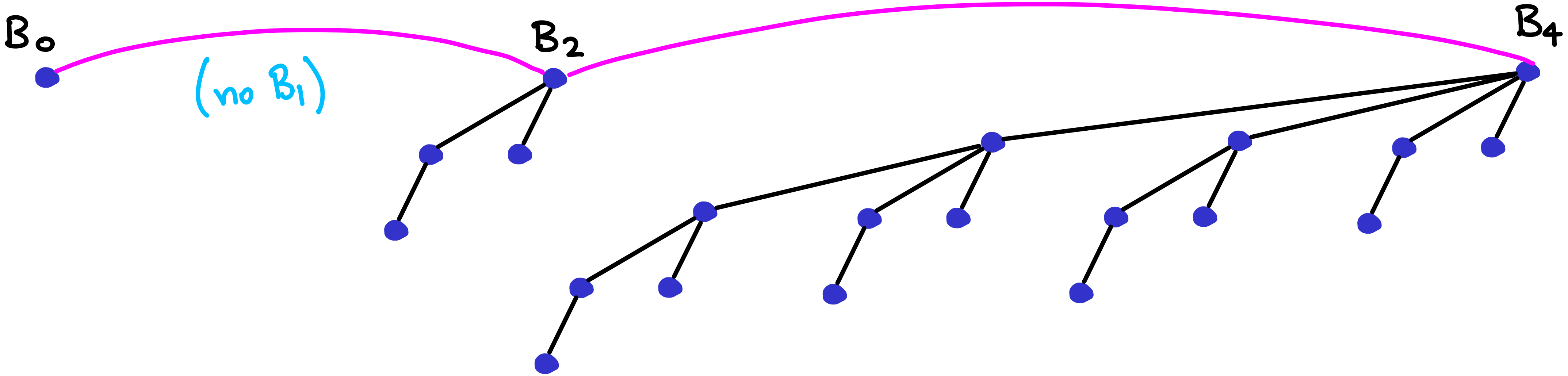


back to the BINOMIAL HEAP

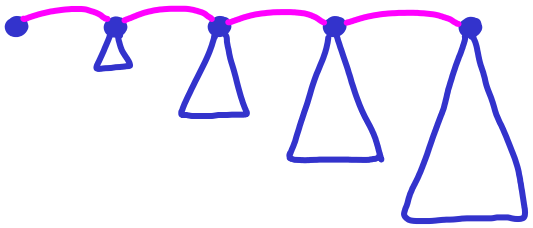
- like regular heap, child > parent
- collection of binomial trees, but at most one for each degree

  (max necessary degree is $\log n$)
- store roots in linked list, sorted by degree

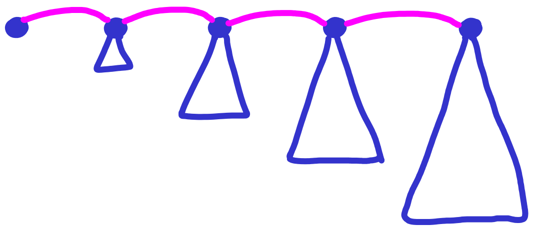
  $\exists$ root node for every 1 in binary representation of n



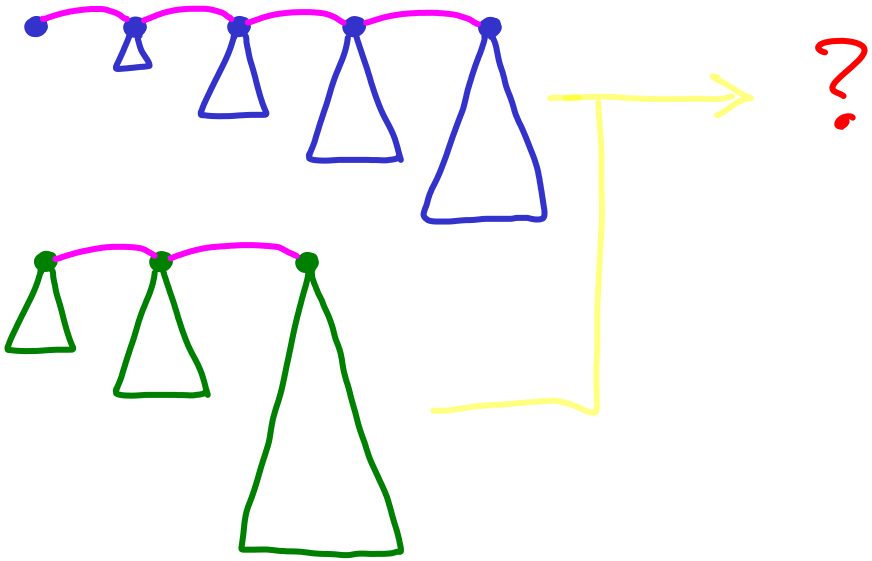
- REPORT MIN: ?



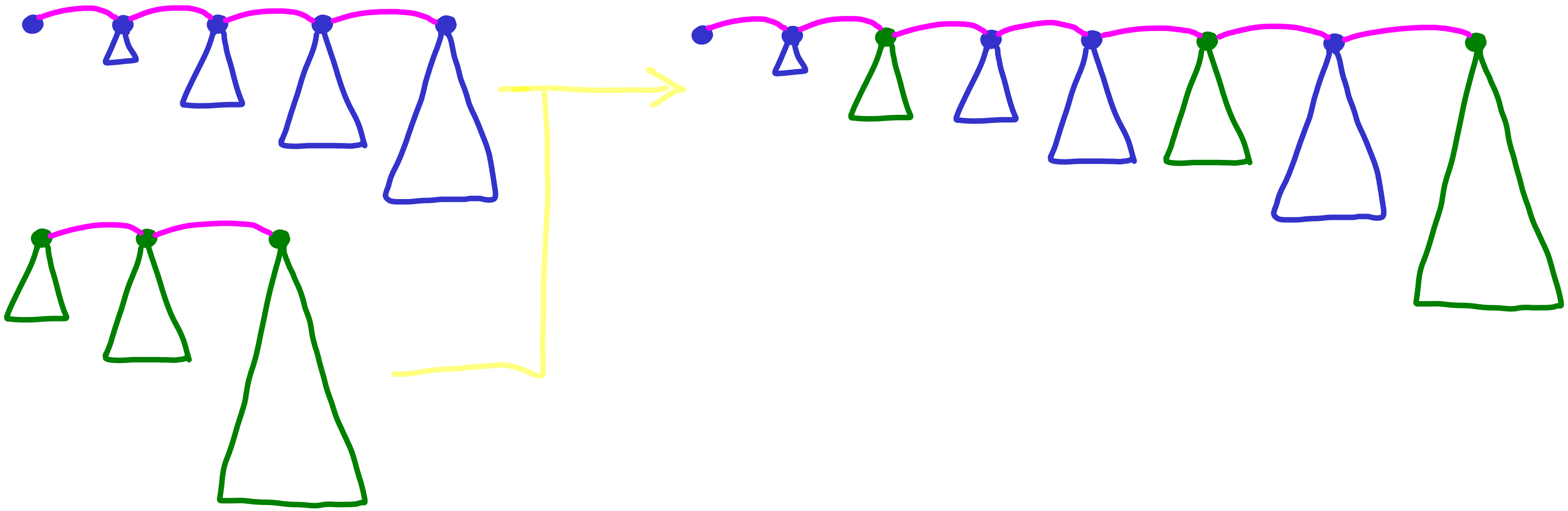
- REPORT MIN: look at roots, O(log n) but can also keep a pointer to min root



- REPORT MIN: look at roots, O(logn) but can also keep a pointer to min root
- UNION: ?

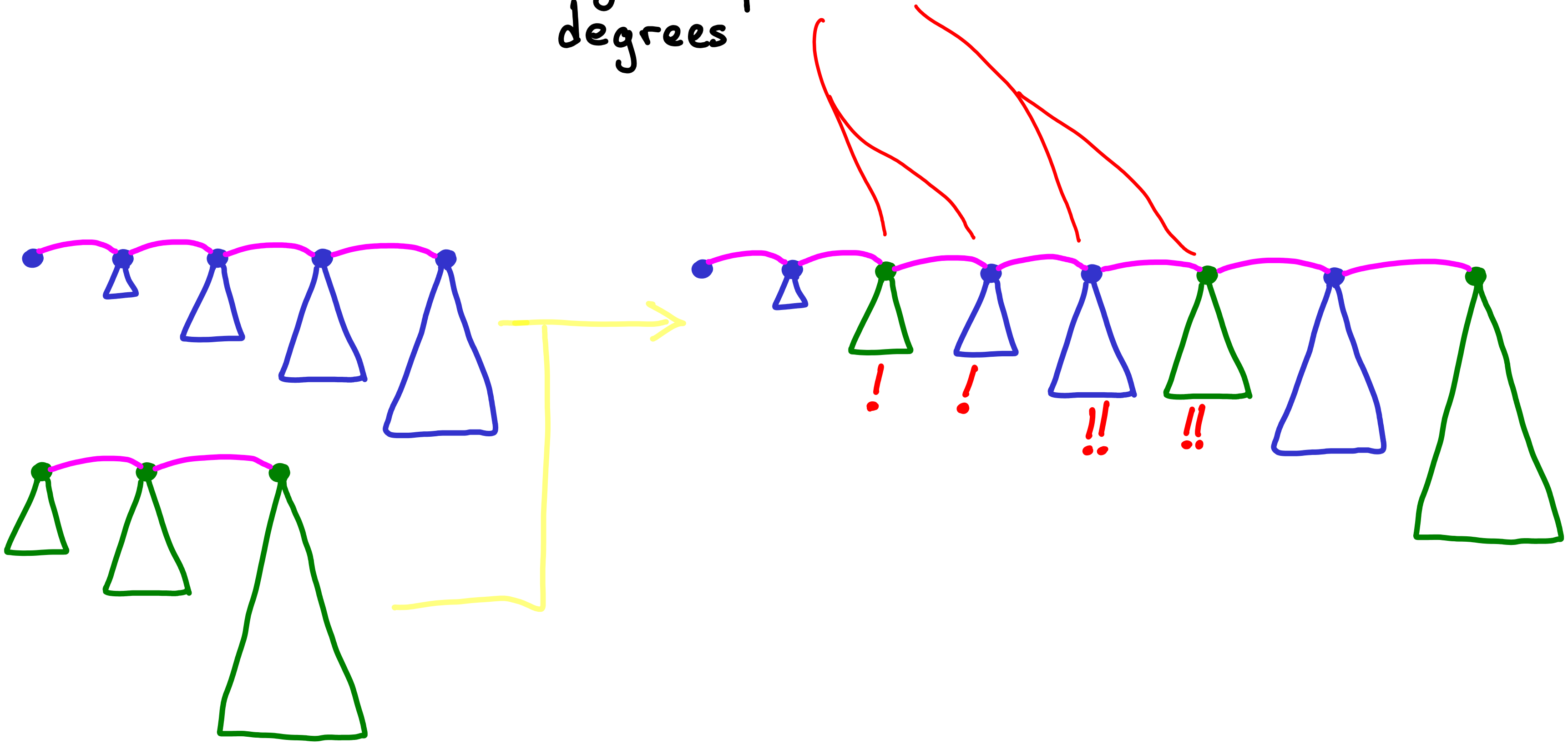


- REPORT MIN: look at roots, O(logn) but can also keep a pointer to min root
- UNION: (i) merge lists → time?



- REPORT MIN: look at roots, O(logn) but can also keep a pointer to min root
- UNION: (i) merge lists O(logn)

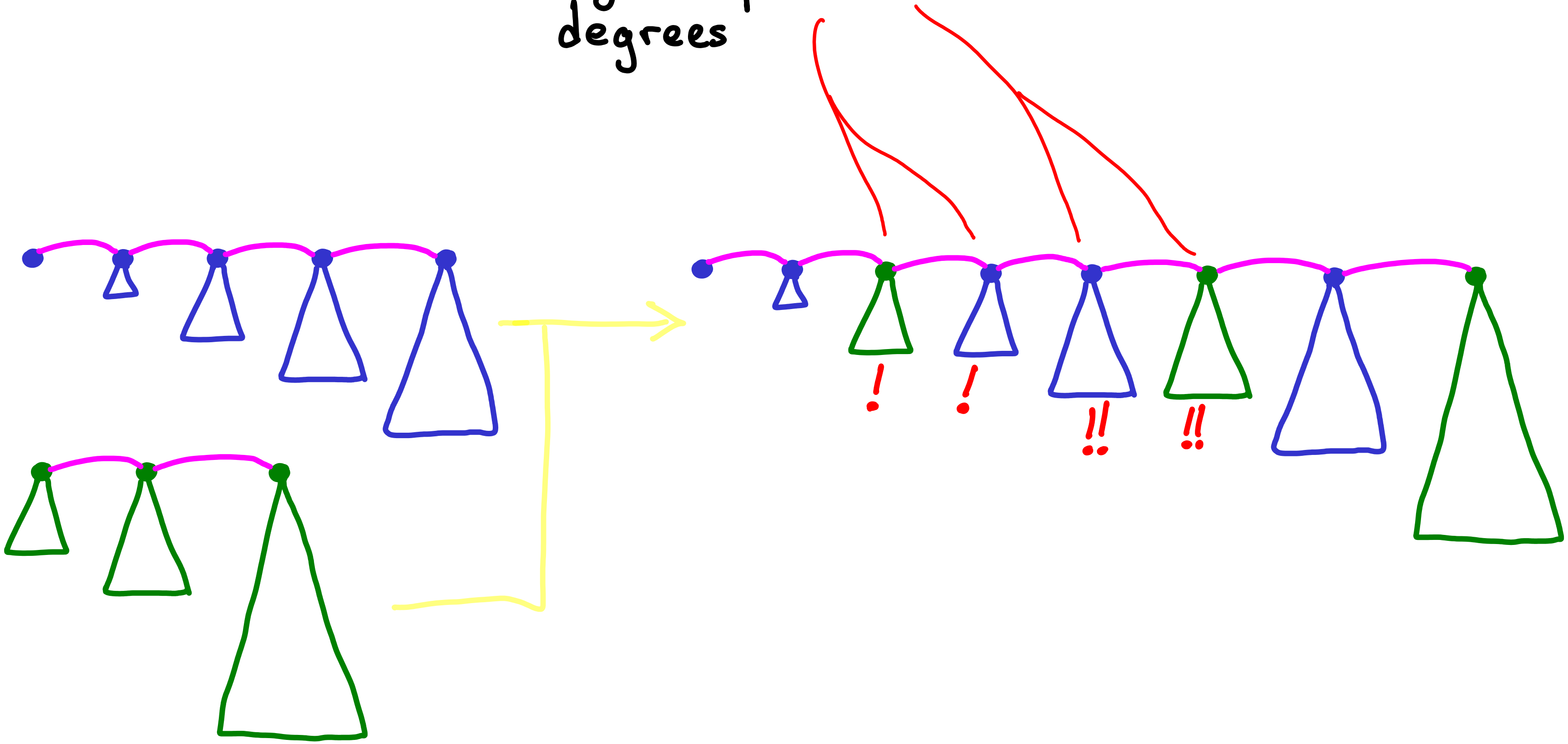
↳ can get duplicate degrees



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ - (ii) restore distinct degrees

↳ can get duplicate degrees

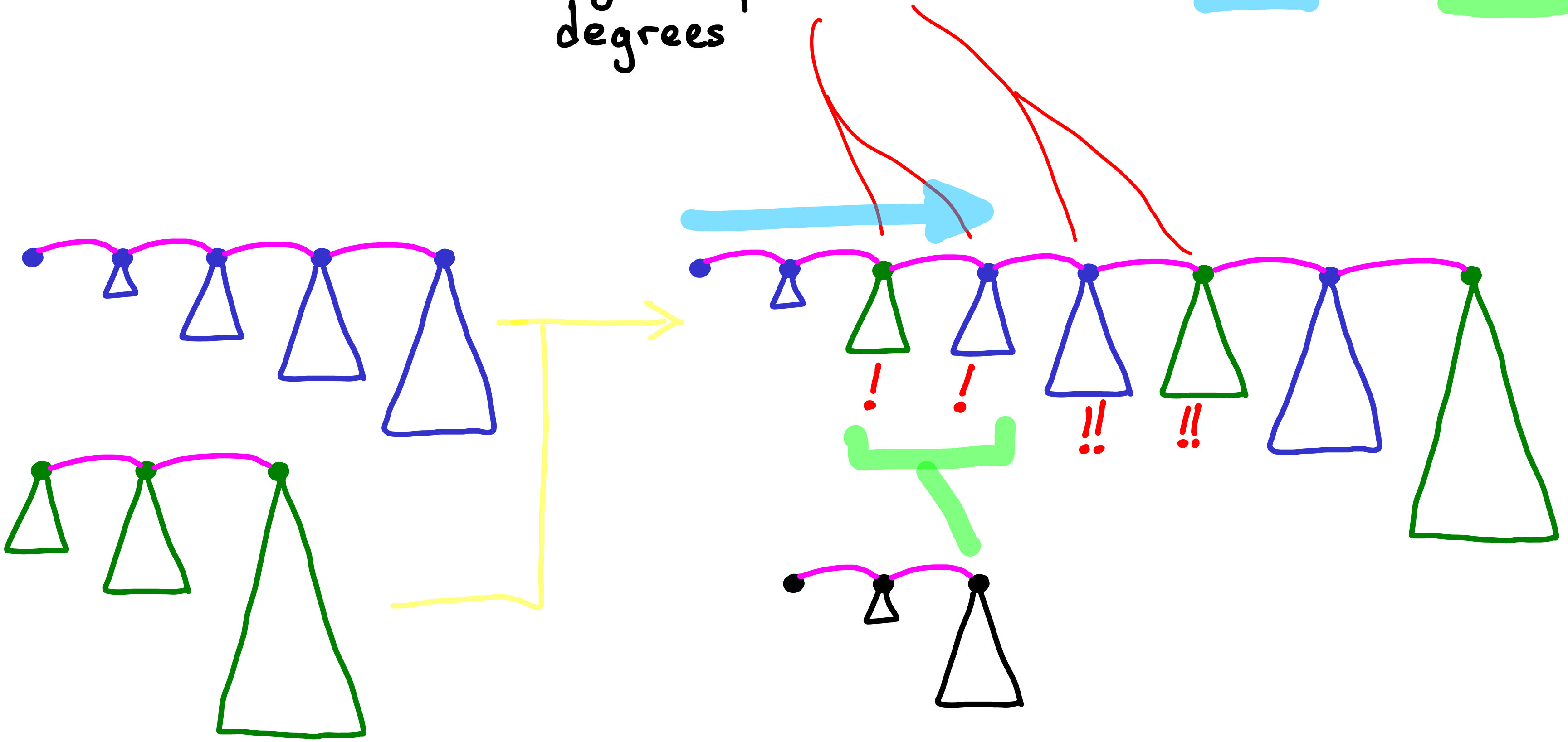
how?



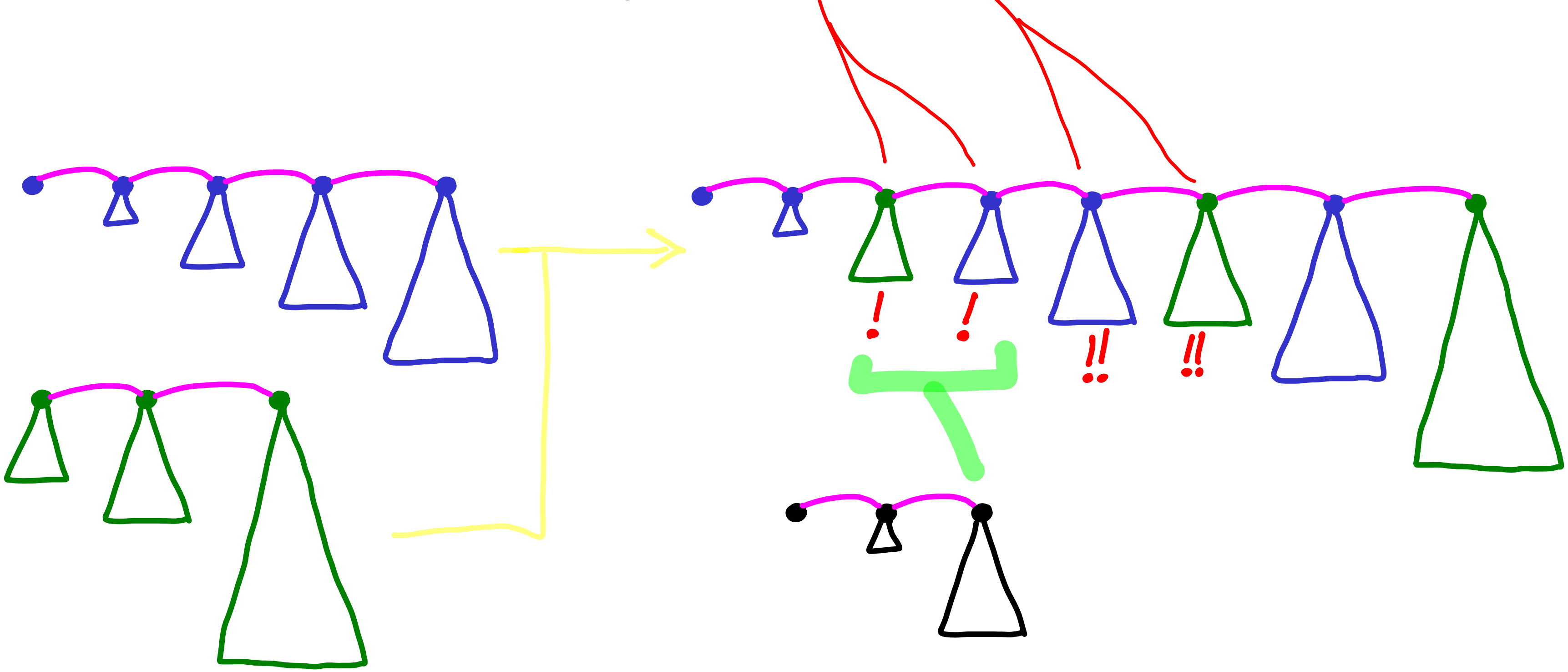
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ - (ii) restore distinct degrees

↳ can get duplicate degrees

↳ scan and merge equal sizes



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees
  - ↳ can get duplicate degrees
  - ↳ scan and merge equal sizes
    - ↳ each merge costs ?



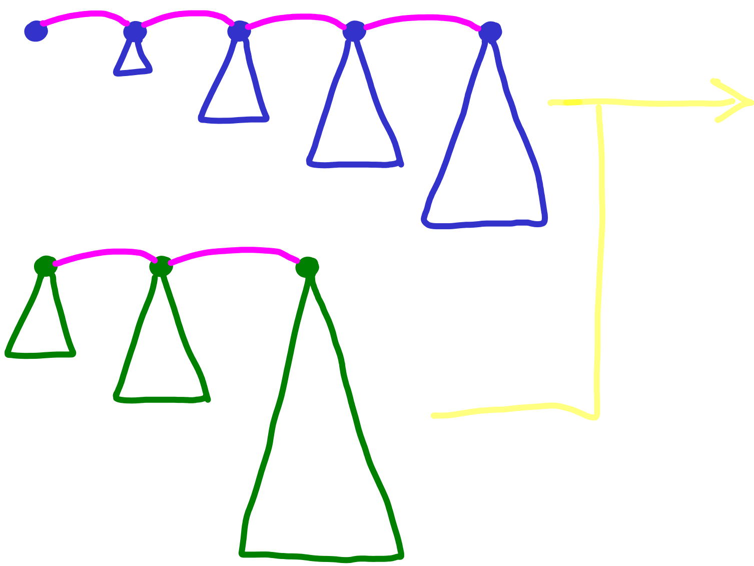
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root

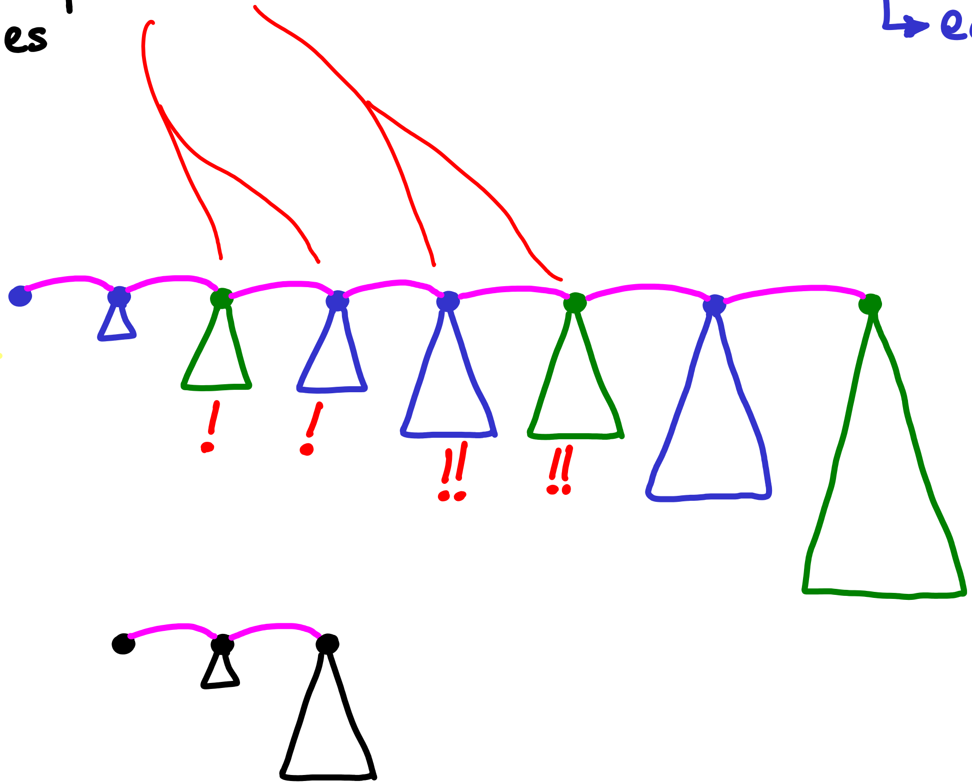
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees

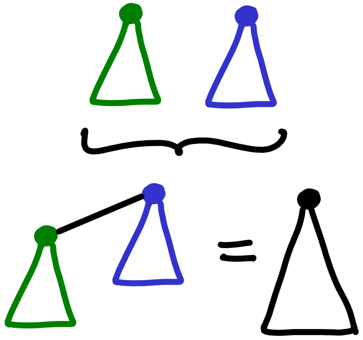
  ↳ can get duplicate degrees

  ↳ scan and merge equal sizes

  ↳ each merge $O(1)$







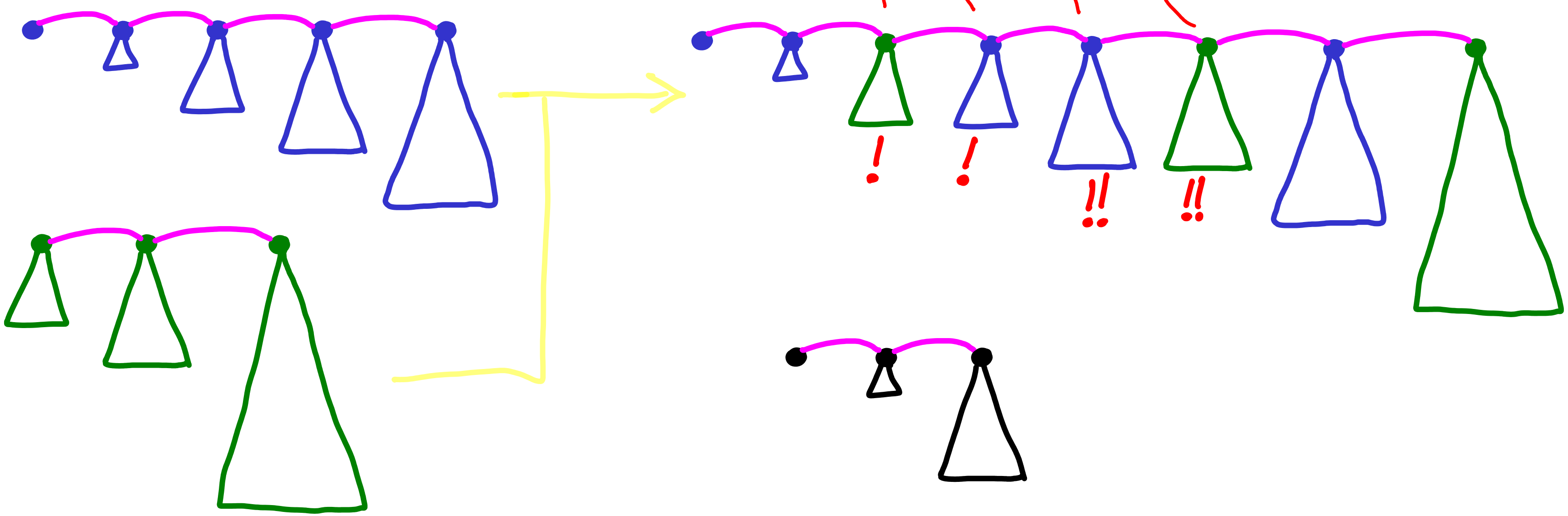
- REPORT MIN: look at roots, O(log n) but can also keep a pointer to min root
- UNION: (i) merge lists O(log n) - (ii) restore distinct degrees

↳ can get duplicate degrees

↳ scan and merge equal sizes

↳ each merge O(1)

↳ new size ...?



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root

- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees

↳ can get duplicate degrees

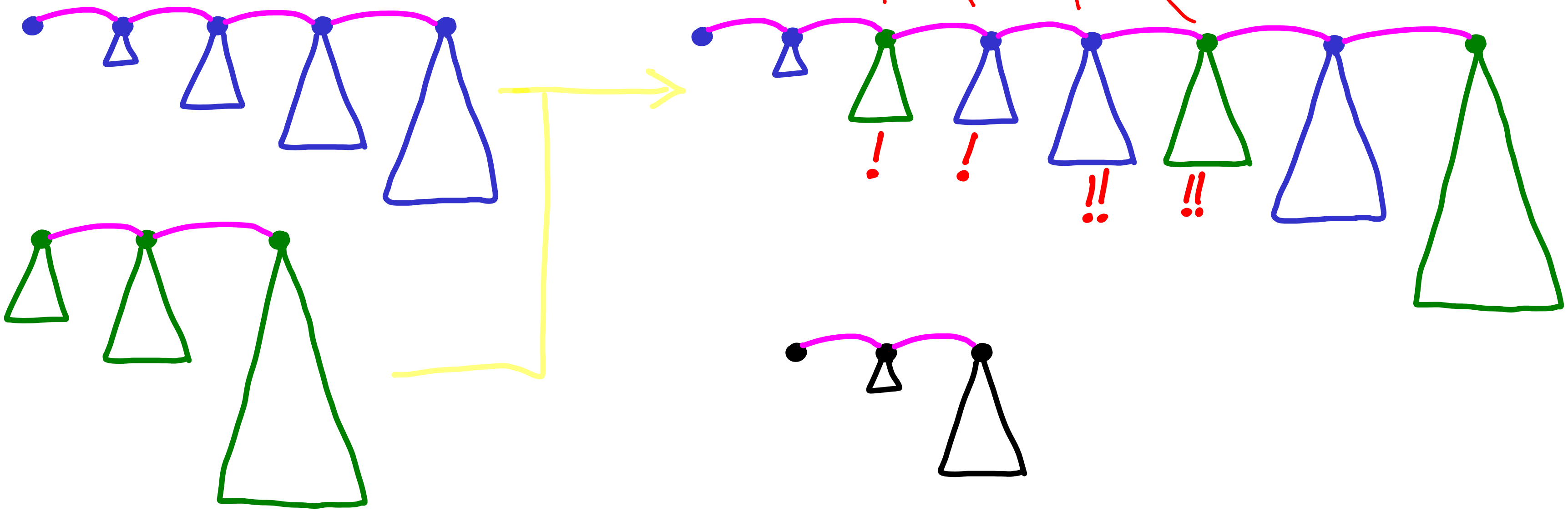
↳ scan and merge equal sizes

↳ each merge $O(1)$

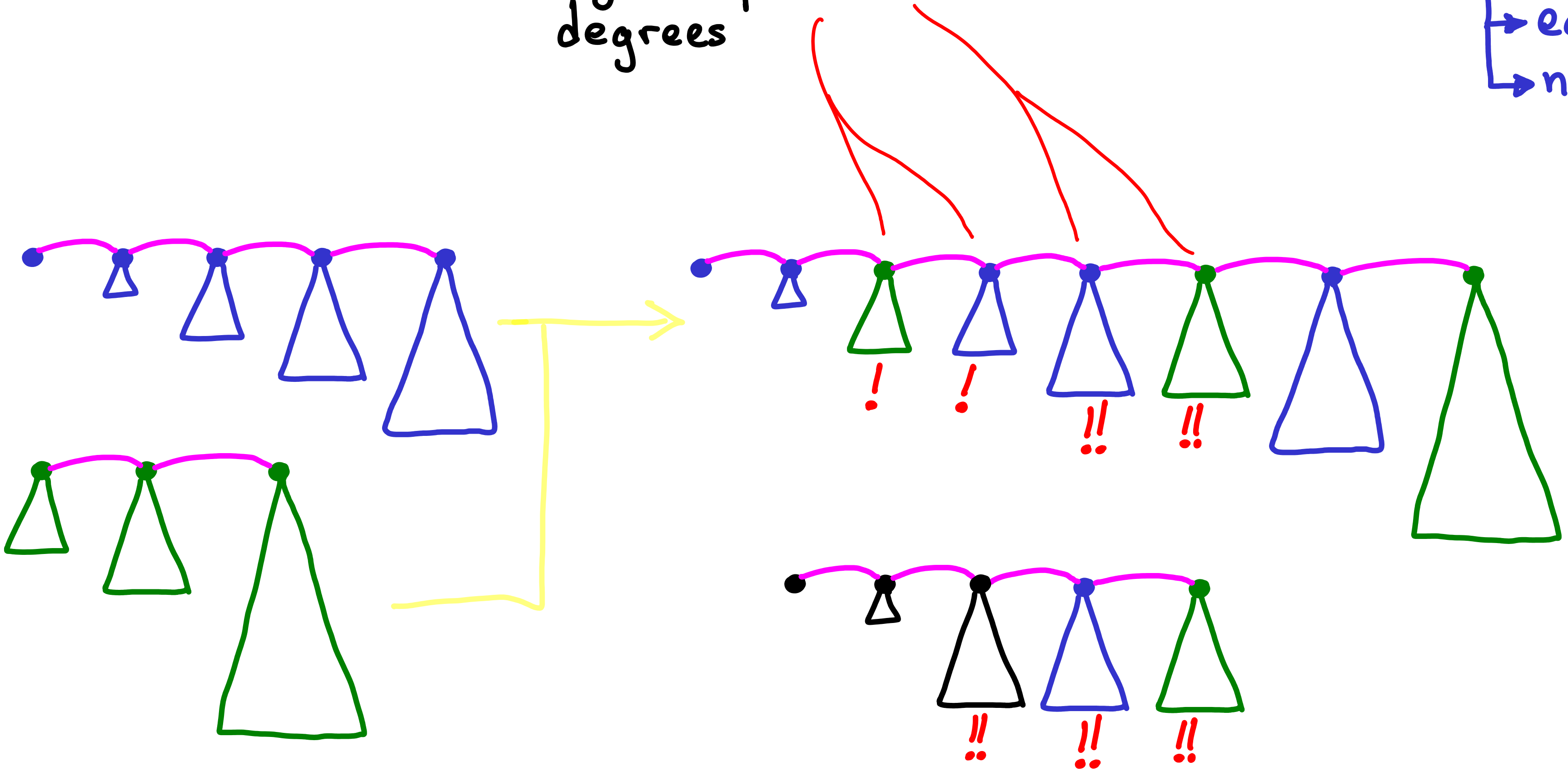
↳ new size $\leq$ next

if $<$ then advance

if $=$ then ?



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees
  - ↳ can get duplicate degrees
  - ↳ scan and merge equal sizes
    - ↳ each merge $O(1)$
    - ↳ new size $\leq$ next
      - if $<$ then advance
      - if $=$ then we get duplicate or triplicate (but no more)



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees

↳ can get duplicate degrees

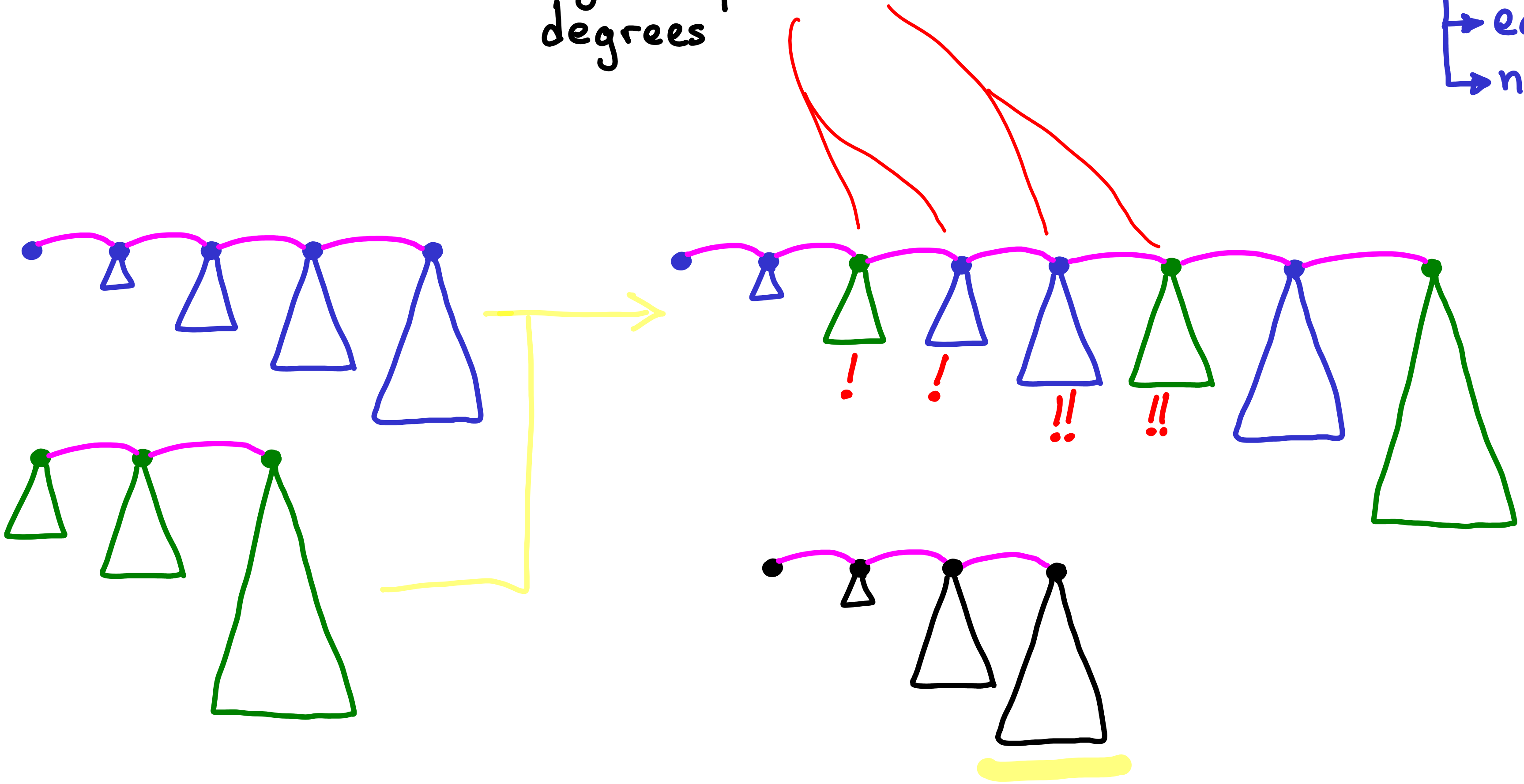
↳ scan and merge equal sizes

↳ each merge $O(1)$

↳ new size ≤ next

if < then advance.

if = then we get duplicate or triplicate, so merge two



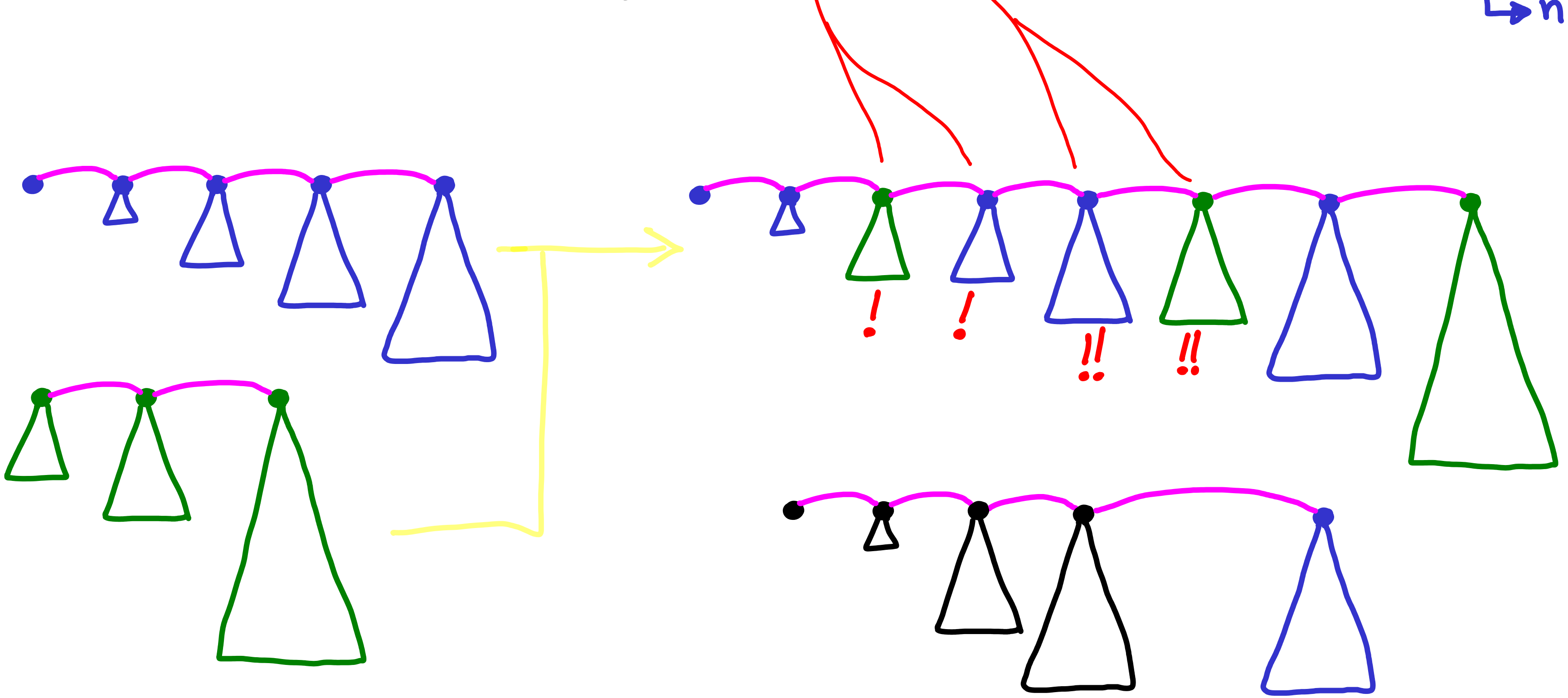
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees

↳ can get duplicate degrees

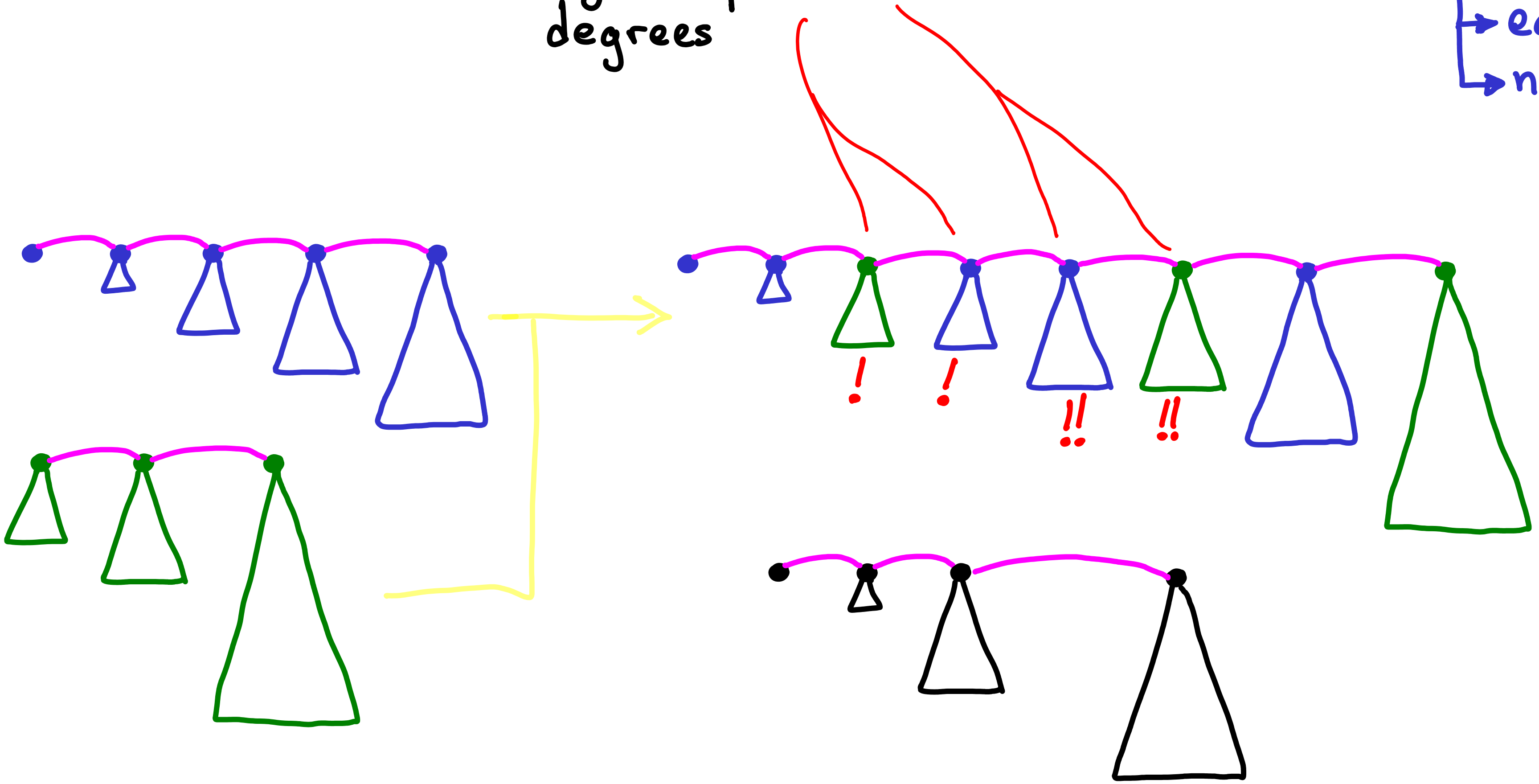
↳ scan and merge equal sizes

- ↳ each merge $O(1)$
- ↳ new size $\leq$ next

if < then advance.
if = then we get duplicate or triplicate, so merge two and advance

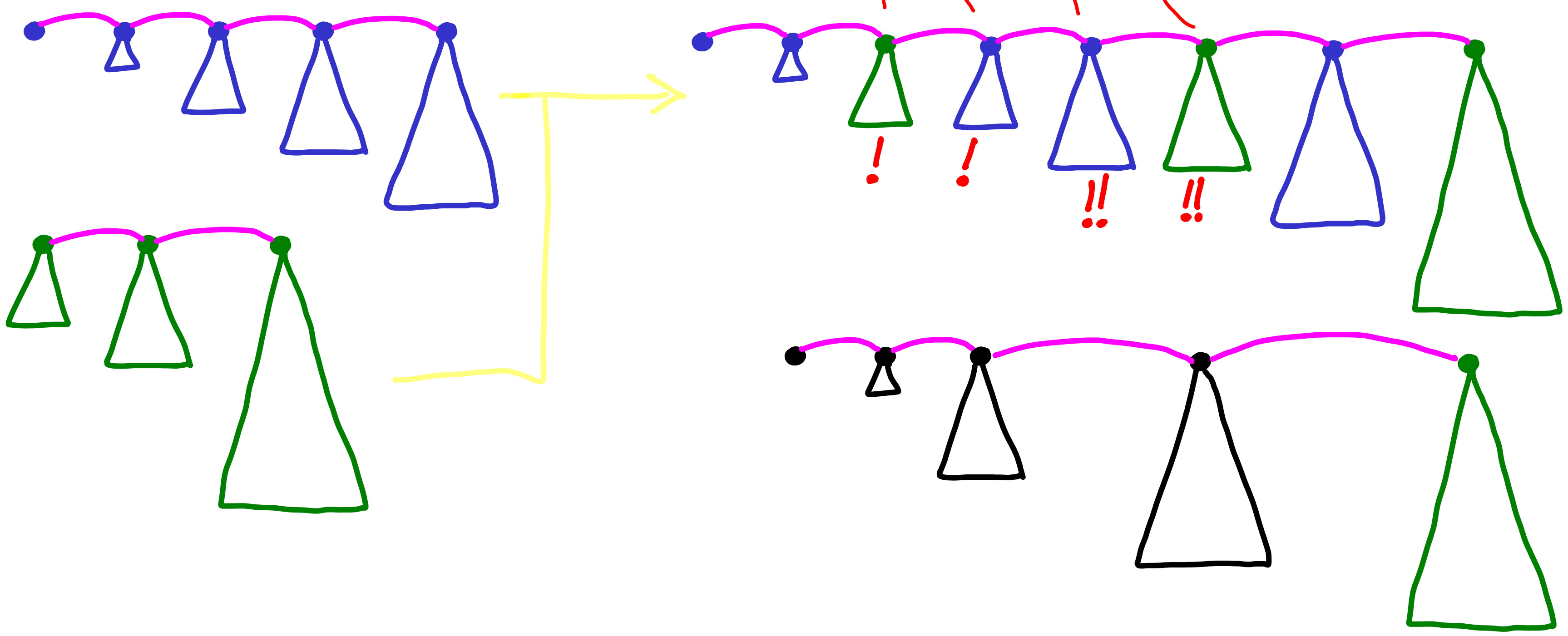


- **REPORT MIN:** look at roots, $O(\log n)$ but can also keep a pointer to min root
- **UNION:** (i) merge lists $O(\log n)$ – (ii) restore distinct degrees
  - ↳ can get duplicate degrees
  - ↳ scan and merge equal sizes
    - ↳ each merge $O(1)$
    - ↳ new size $\leq$ next
      - if $<$ then advance.
      - if $=$ then we get duplicate or triplicate, so merge two and advance



new merge

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists $O(\log n)$ – (ii) restore distinct degrees
  - ↳ can get duplicate degrees
  - ↳ scan and merge equal sizes $O(\log n)$
    - ↳ each merge $O(1)$
    - ↳ new size $\leq$ next
      - if $<$ then advance.
      - if $=$ then we get duplicate or triplicate, so merge two and advance



Conclusion:

UNION $O(\log n)$

($\sim$ binary addition)

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists -(ii) restore distinct degrees $O(\log n)$
- INSERT: ?

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts

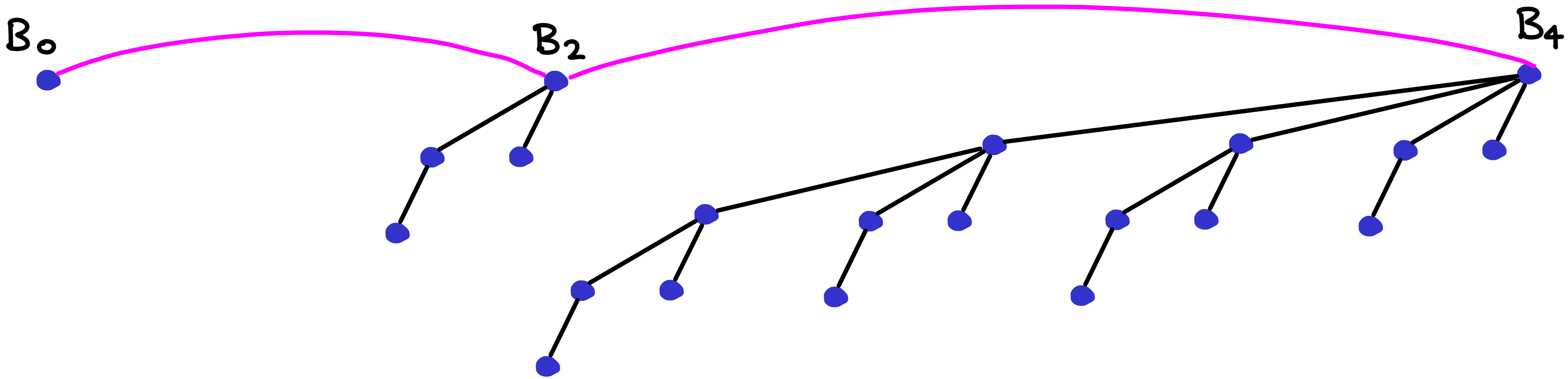
WHY?

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts

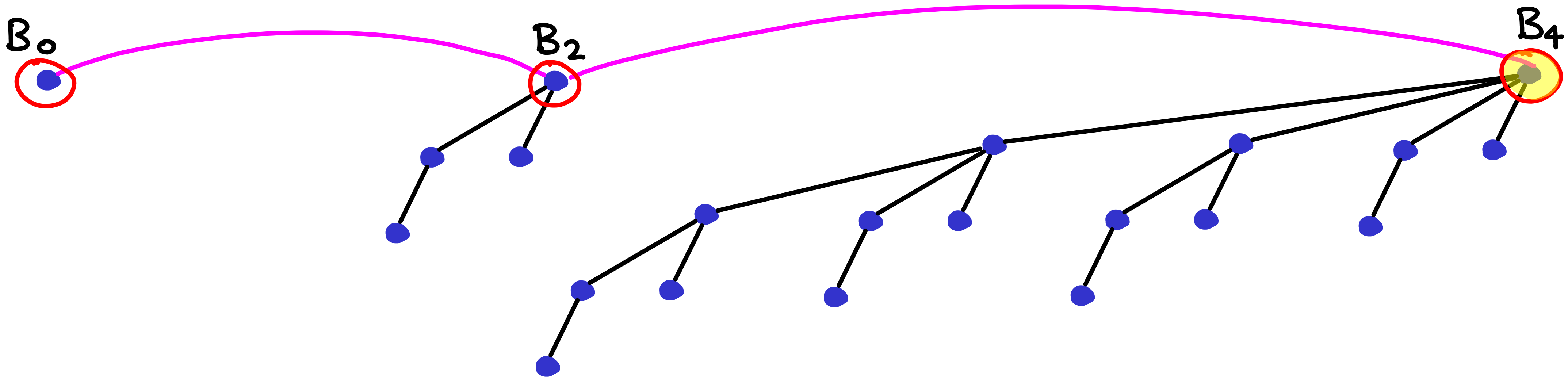
WHY?

It's the binary counter problem!
(see CLRS)

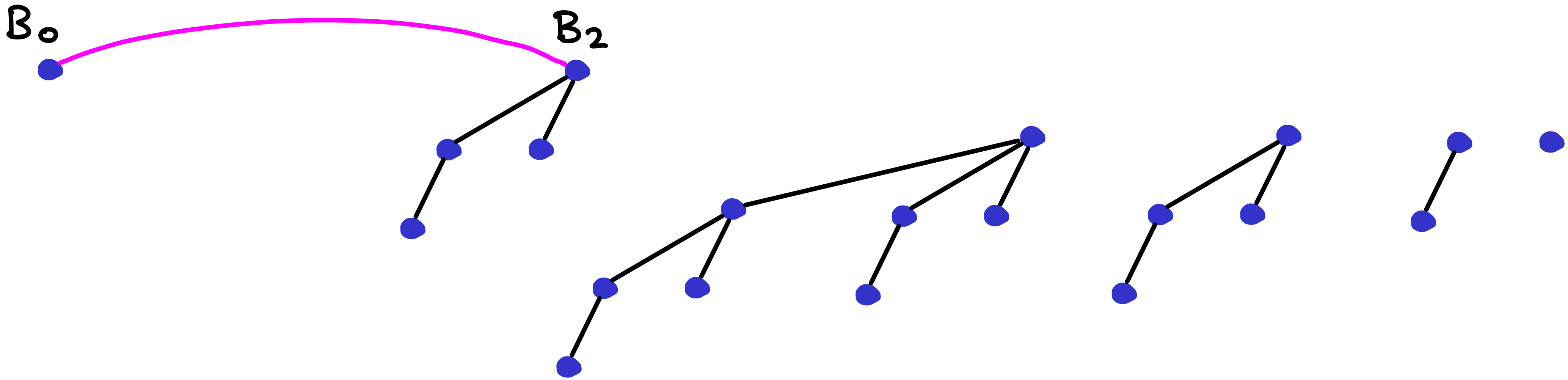
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: ?



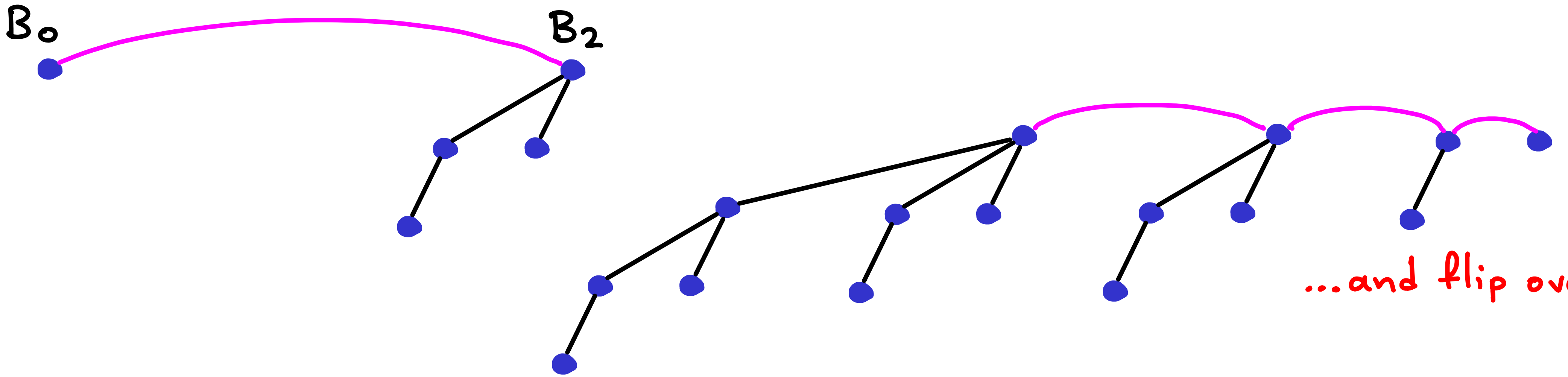
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min $O(\log n)$ or $O(1)$



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min $O(\log n)$

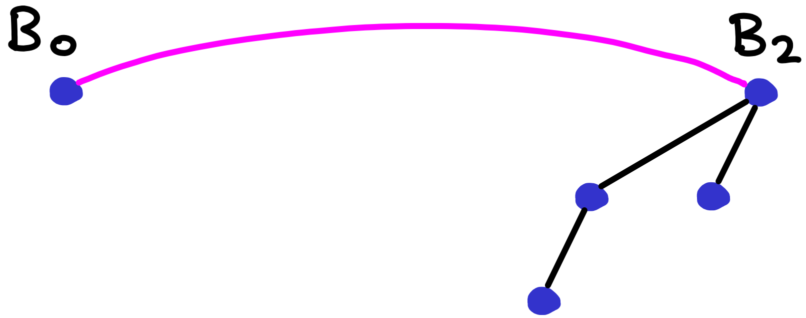


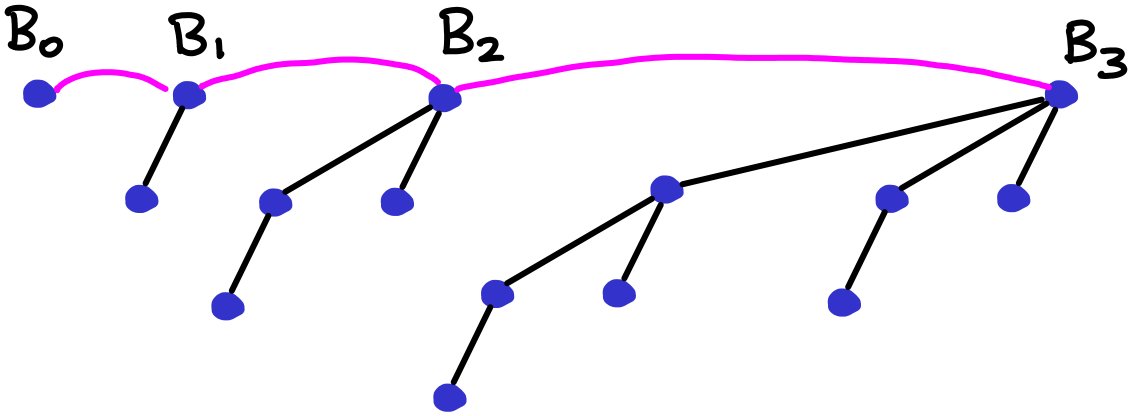
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children $O(\log n)$



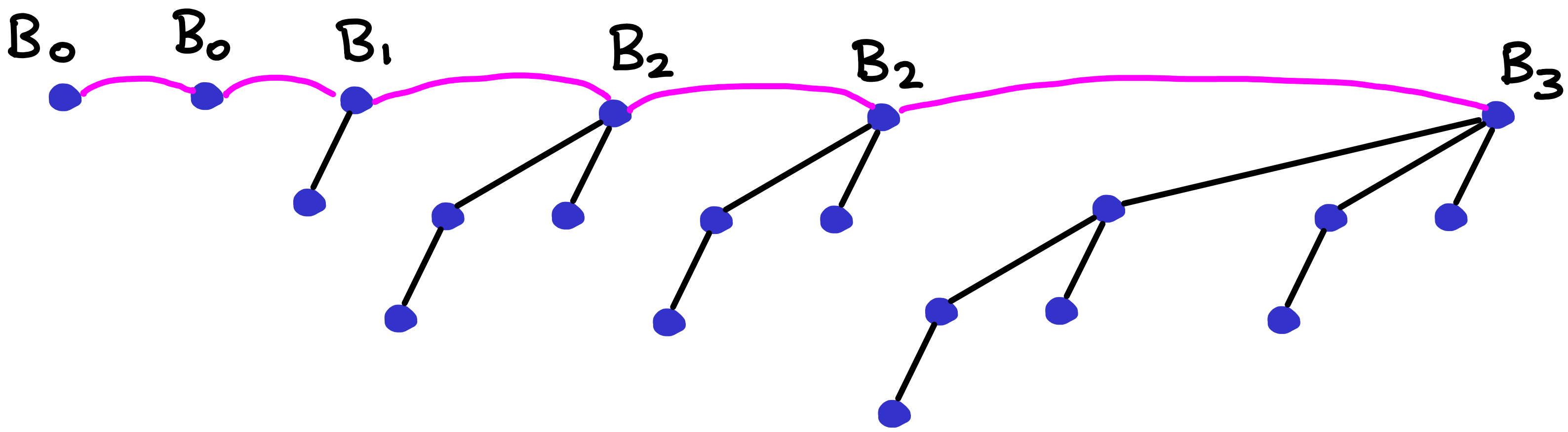
...and flip over

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children $O(\log n)$

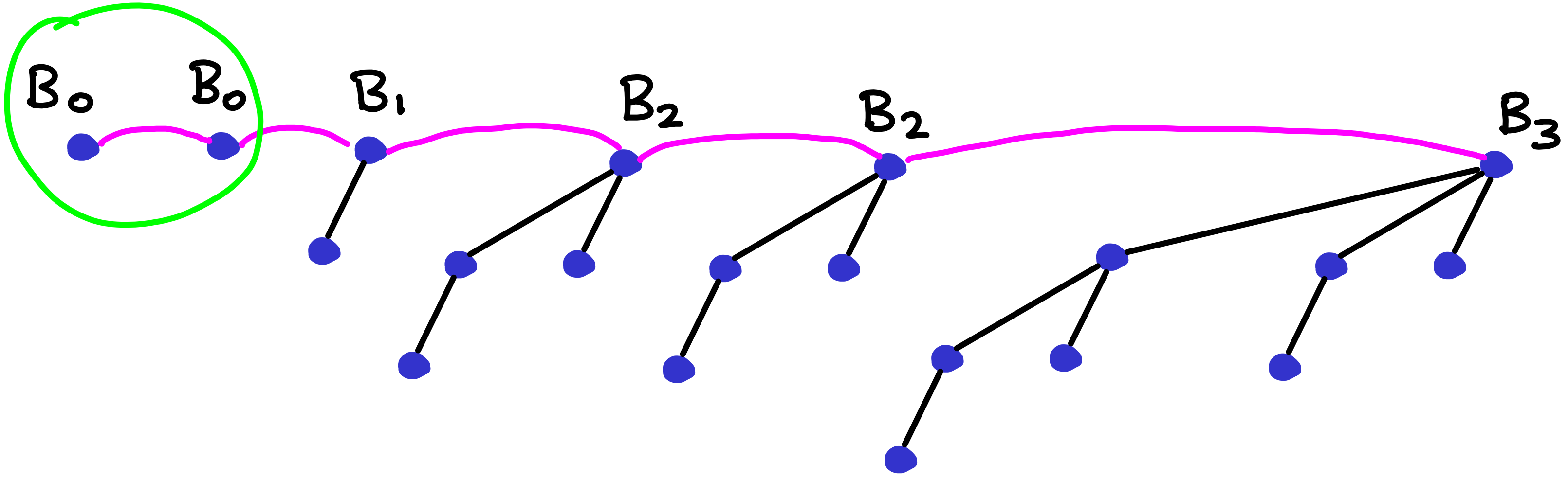




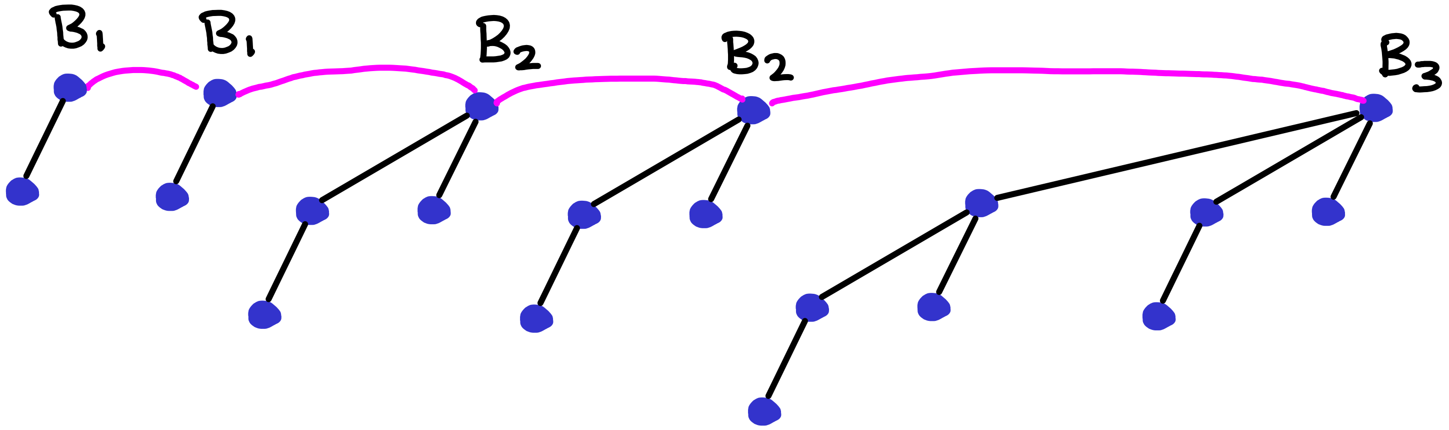
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\Big\}$ $O(\log n)$



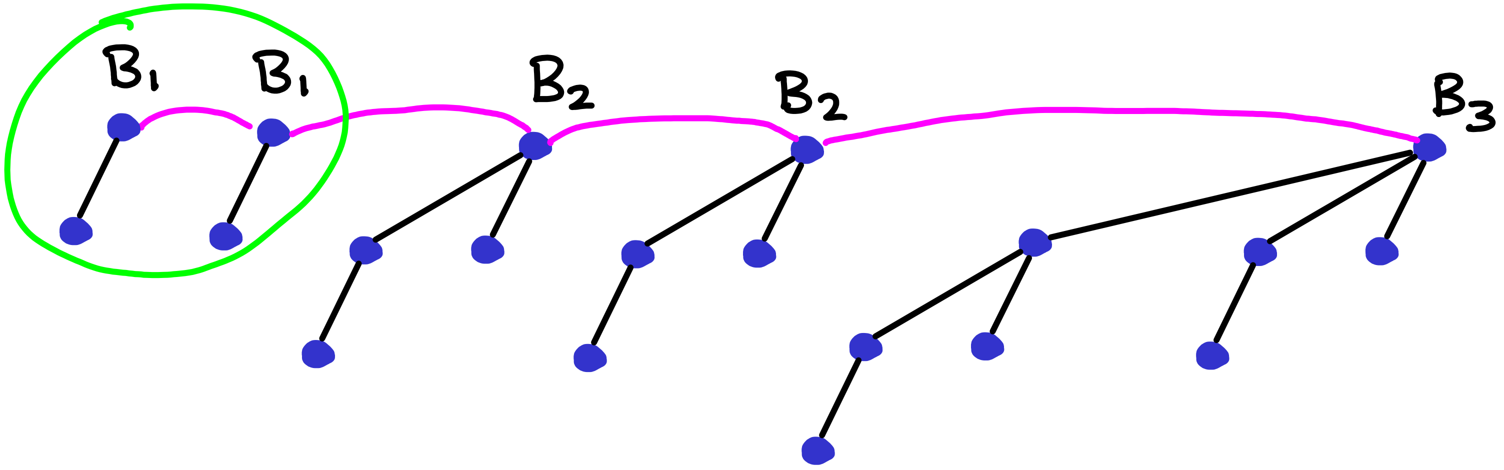
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\}$ $O(\log n)$



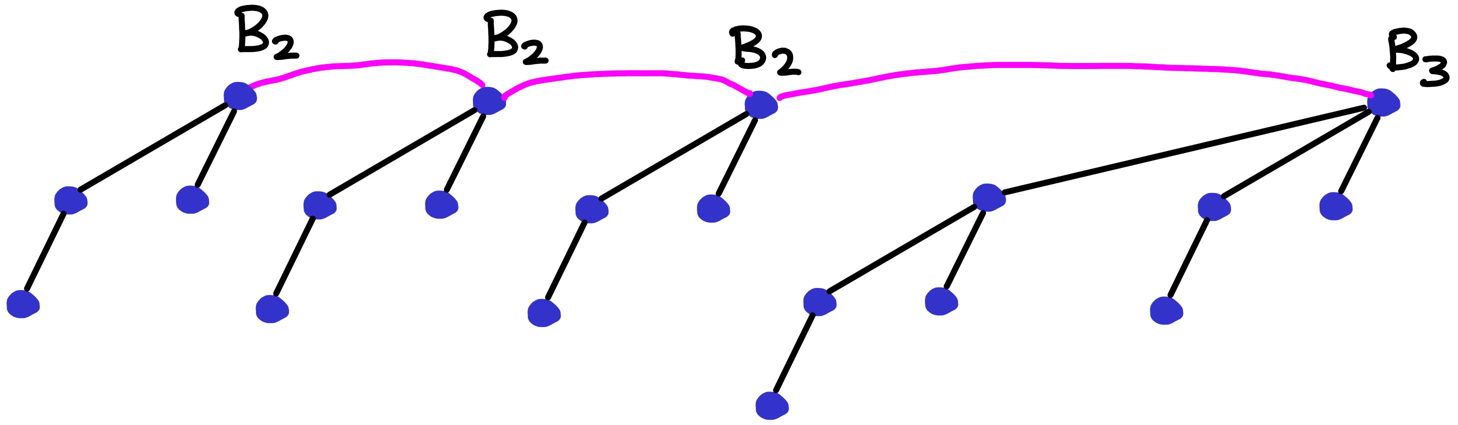
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\Big\}$ $O(\log n)$



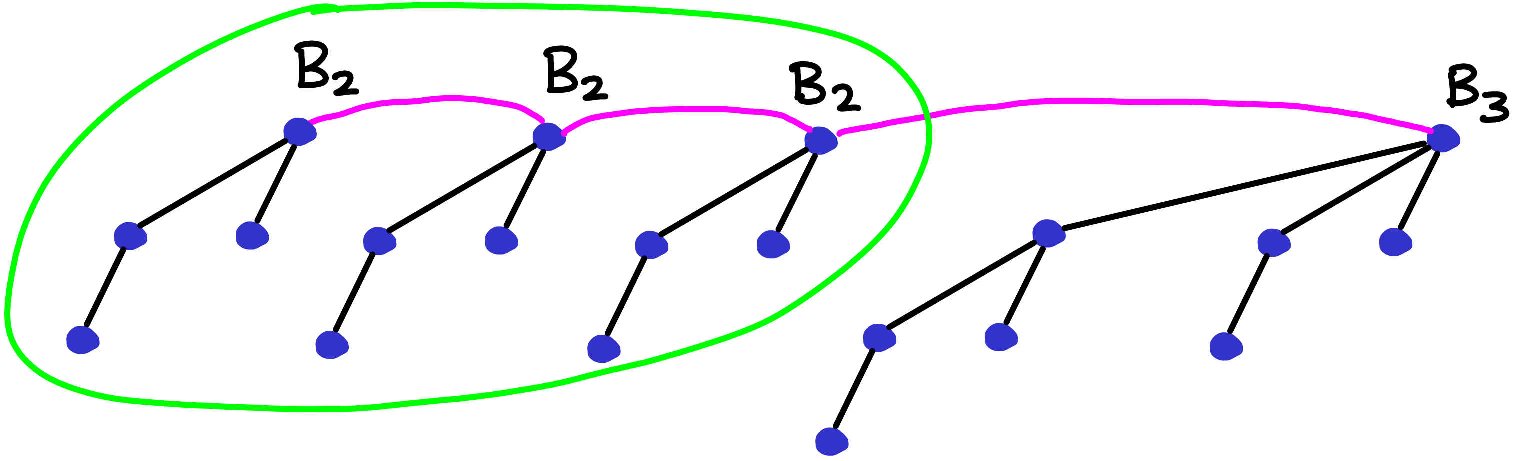
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\}$ $O(\log n)$



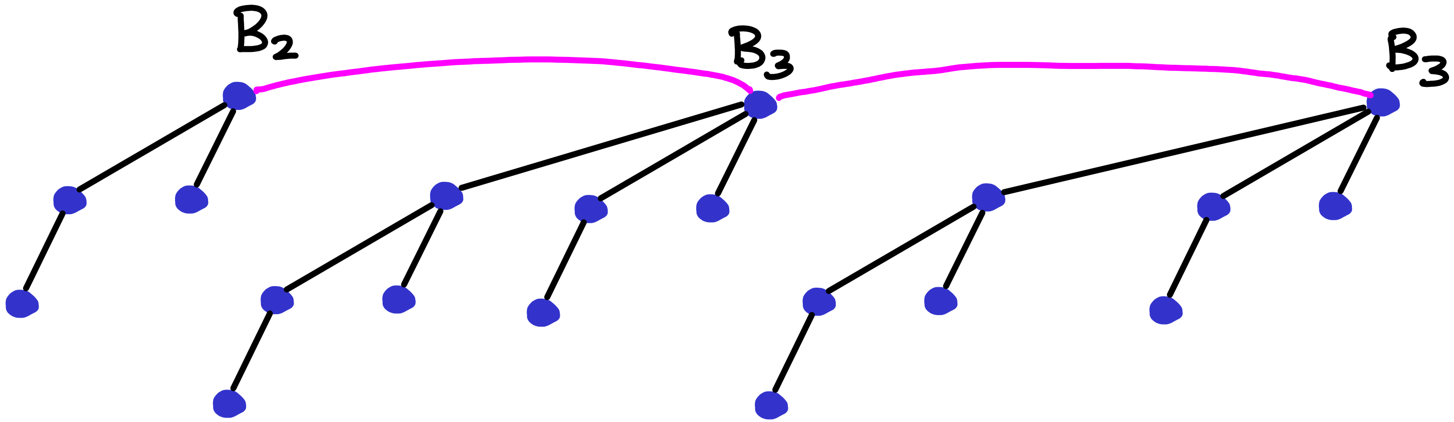
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union } $O(\log n)$



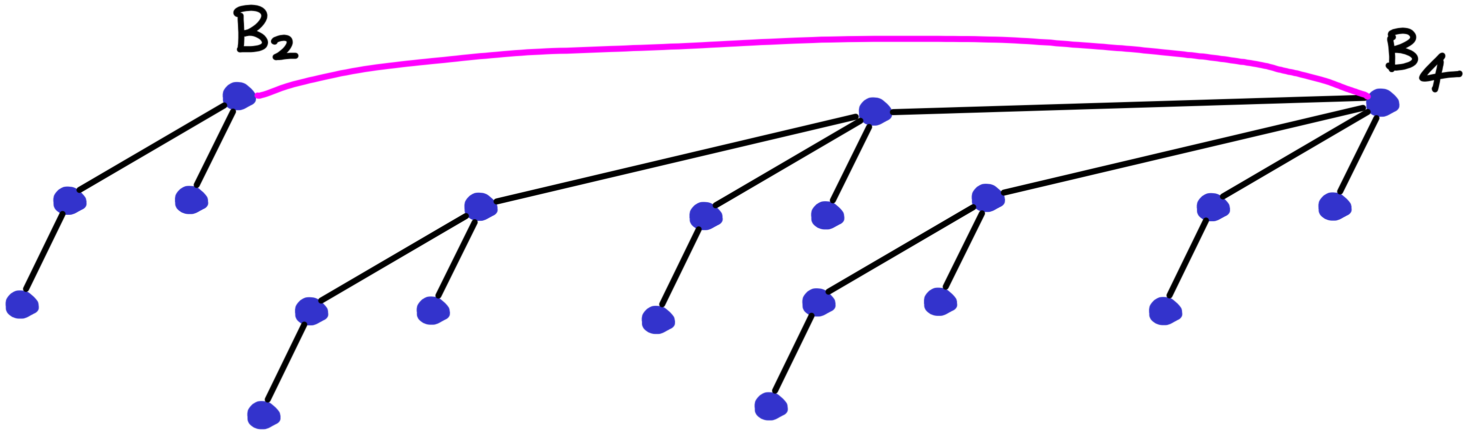
- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\Big\}$ $O(\log n)$



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union } $O(\log n)$



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union } $O(\log n)$



- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists - (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ - but $O(1)$ amortized for n inserts
- EXTRACT MIN: $\left.\text{(i) find min, (ii) make heap from children, (iii) union}\right\}$ $O(\log n)$
- DECREASE KEY: ?

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: $\left.\text{(i) find min, (ii) make heap from children, (iii) union}\right\}$ $O(\log n)$
- DECREASE KEY: like regular heap $O(\log n)$

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union } $O(\log n)$
- DECREASE KEY: like regular heap $O(\log n)$
- DELETE: decrease to $-\infty$ & extract min $O(\log n)$

- REPORT MIN: look at roots, $O(\log n)$ but can also keep a pointer to min root *
- UNION: (i) merge lists – (ii) restore distinct degrees $O(\log n)$
- INSERT: = union $O(\log n)$ – but $O(1)$ amortized for n inserts
- EXTRACT MIN: (i) find min, (ii) make heap from children, (iii) union $\}$ $O(\log n)$
- DECREASE KEY: like regular heap $O(\log n)$
- DELETE: decrease to $-\infty$ & extract min $O(\log n)$

* can be updated during other operations $O(1)$