Median with 'Quickselect'

- The first partitioning leaves us with two subarrays, but we need only recurse on the one that contains the median

$n = 7$ median $\rightarrow 4^{th}$ pos

p

3 6 4 2 1 7 5

3 1 4 2 6 7 5

3 1 4 2 $\boxed{5}$ 7 6

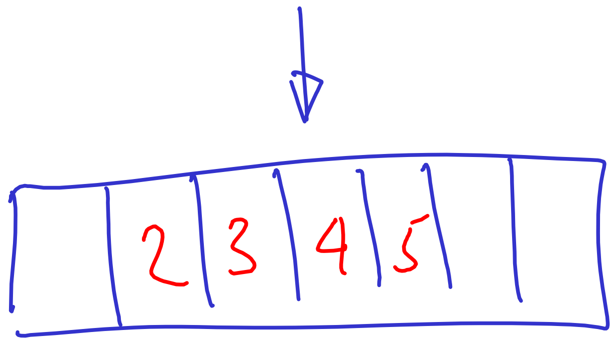
3 1 4 2

1 3 4 2

p

1 $\boxed{2}$ 4 3

1 $\boxed{2}$ $\boxed{3}$ $\boxed{4}$



Quickselect Best Case

Best case ⇒ Evenly partioning (halving)

$$f(n) = f\left(\tfrac{n}{2}\right) + kn$$

$n = 2^m$

$$= f(2^{m-1}) + k2^m$$

$$= f(2^{m-2}) + k2^{m-1} + k2^m$$

$$= f(2^{m-a}) + k\sum_{i=m-a+1}^{m} 2^i$$

$m = a$.

$$= f(2^0) + k\sum_{i=1}^{m} 2^i$$

$$= k_2 + k \cdot 2 \sum_{i=0}^{m-1} 2^i$$

$$= k_2 + k \cdot 2 \cdot (2^m - 1)$$

$$= k_2 + k \cdot 2\,(n-1) \Rightarrow O(n)$$

10111

11111

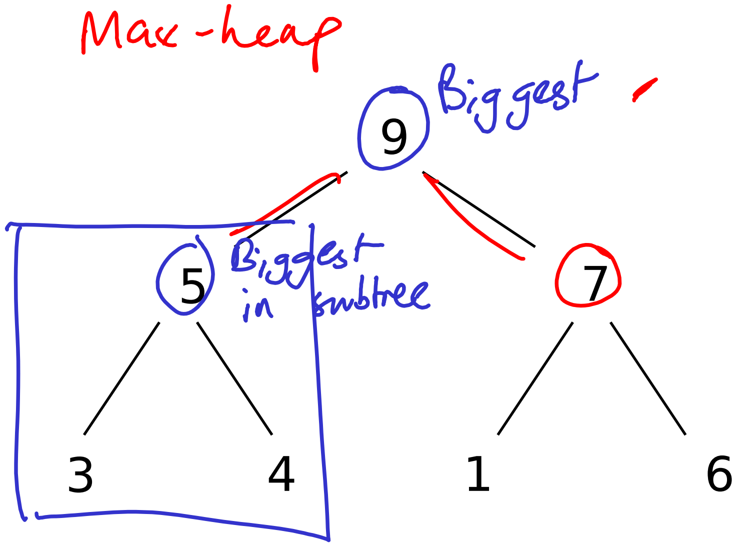
$2^0 + 2^1 + 2^2$

- Exercise: show the worst case is still $O(n^2)$

Heapsort

- One last interesting algorithm
- Sorts in place and guarantees O(nlogn) for any input!
  - Although the constant of proportionality is greater than for quicksort
- Particularly interesting for us because it is based on a data structure called a heap (which we will use later on too)

Introducing Heaps

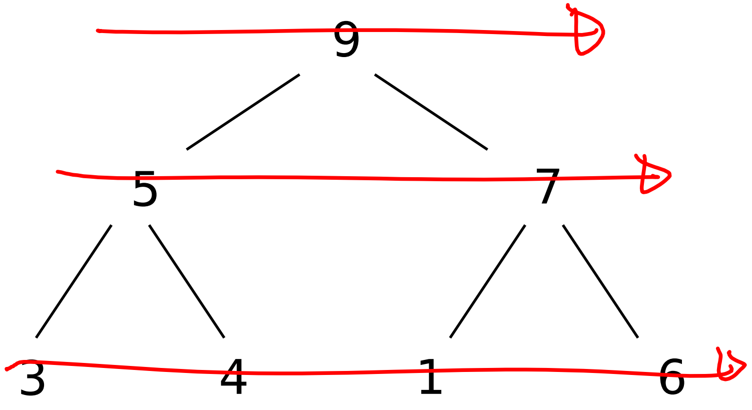


- Consider a simple binary tree (two branches out of each node)
- There is only one rule for our heap: the value of the two children must each be less than the value of the parent

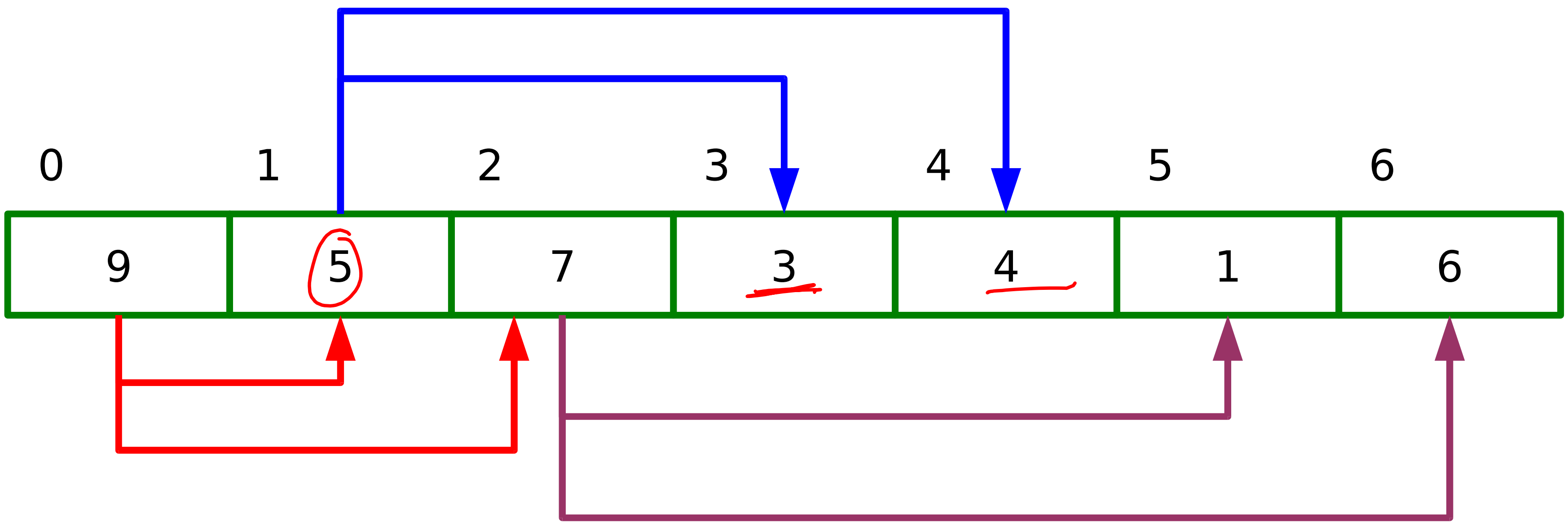
Note:

- The root of the tree is always the biggest number
- The root of any subtree is the biggest number in that subtree

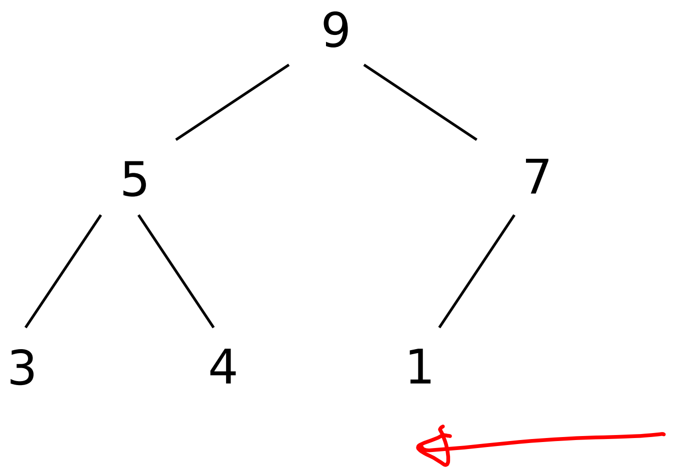
Introducing Heaps

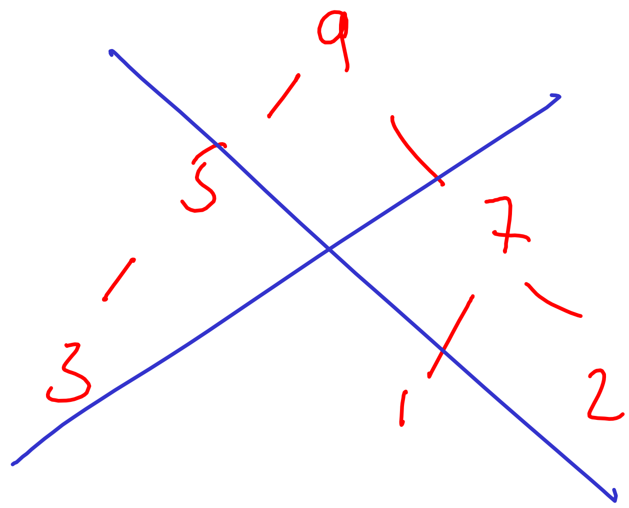


- Now we represent this tree using an array in a certain way:
- The children of the node at [i] can be found at [2i+1] and [2i+2] (array starts at zero)



Must be 'almost full' or full





0	1	2	3	4	5	6
9	5	7	3	4	1	

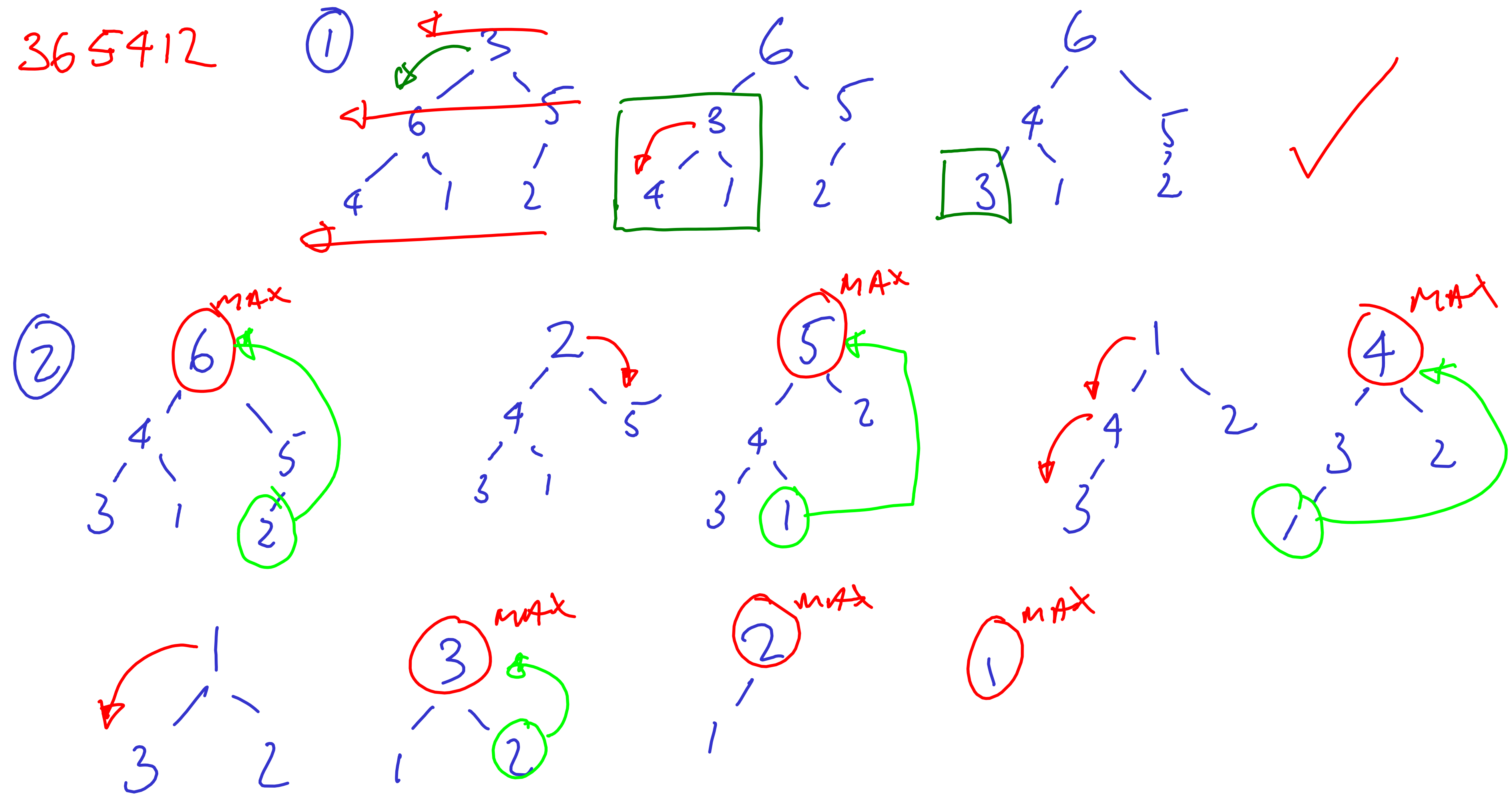
Heapsort

- Heap<u>sort</u> proceeds as follows:
  1. Make your data into a heap in the array [0...n]
  2. for (k=n-1...0)
     1. Swap element 0 and element k
     2. Make the array [0...k] a valid heap

- Trick is in the 'heapify' bit

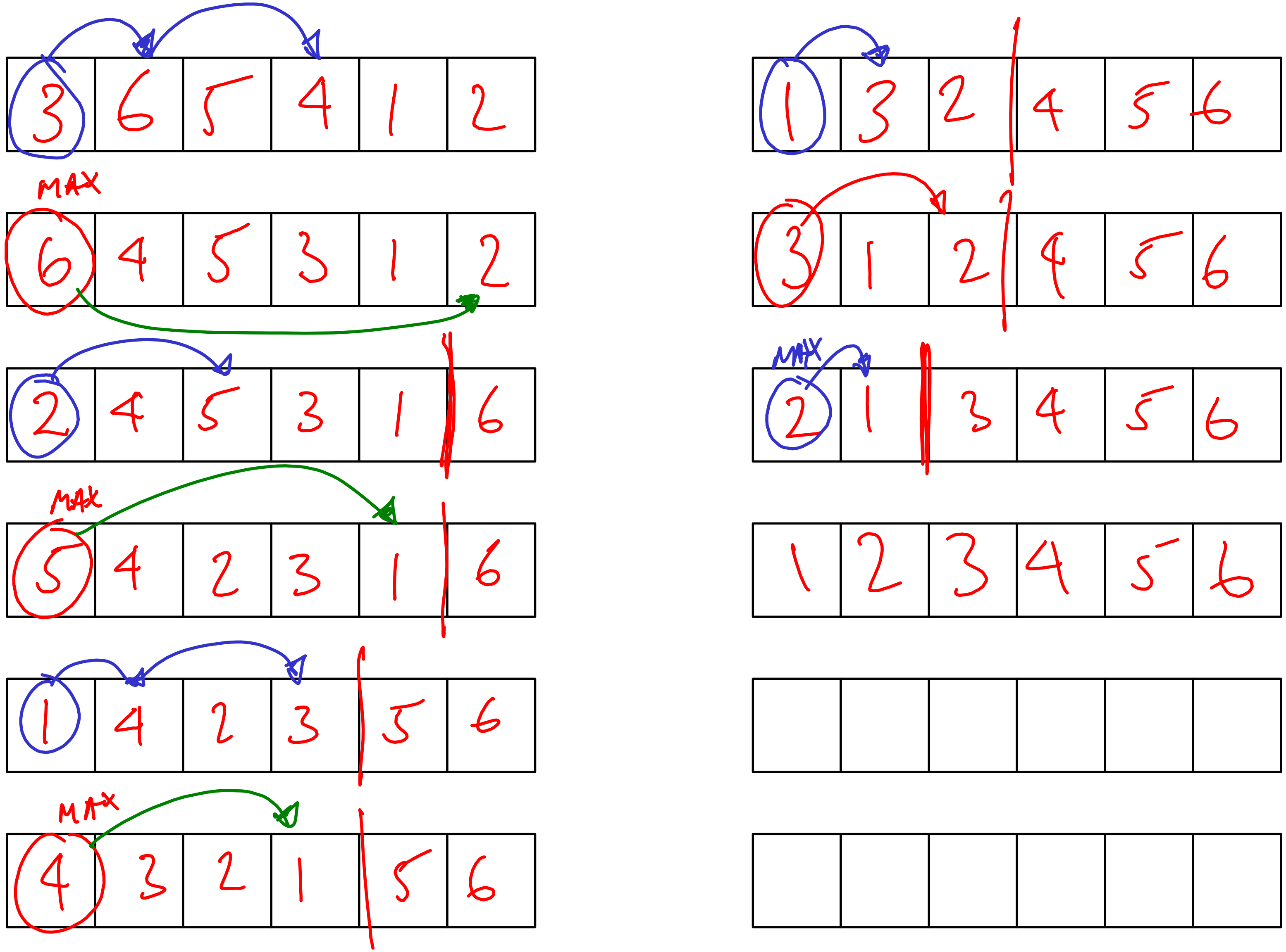
Heapsort: Graph View

365412



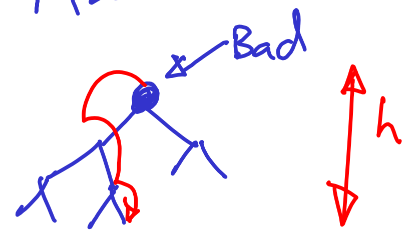
1	2	3	4	5	6

Heapsort: Array View

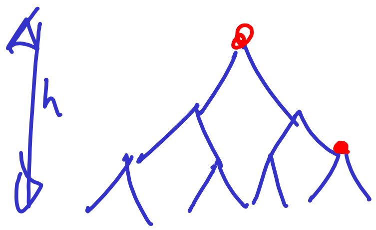


Heapsort: Analysis of Step 1

Assume a valid heap except first element



$O(h)$ to fix that one element $(h = \lg n)$



j	Num nodes
0	2^0
1	2^1
2	2^2
3	2^3

Each level j have 2^j nodes

Each node O(height of node)

For given level Num # nodes × cost $= 2^j \times (h-j)k$

Heapsort: Analysis of Step 1

$$\text{Total cost} = \sum_{j=0}^{h-1} k\,2^j (h-j)$$

$$= k\,2^h \sum_{j=0}^{h-1} 2^{j-h}(h-j) \qquad \text{subs } m = h-j$$

$$= k \cdot 2^h \sum_{m=1}^{h} 2^{-m} m \qquad \sum_{0}^{\infty} \underbrace{a}_{\frac{1}{2}} x^a = \frac{x}{(1-x)^2} \quad |x|<1$$

$$= k 2^h \sum \left(\tfrac{1}{2}\right)^m m$$

$$= \frac{k\left(\frac{1}{2}\right) 2^h}{\left(1-\frac{1}{2}\right)^2}$$

$$= O(2^h) \qquad h = \ln n$$

$$= \underline{\underline{O(n)}}$$

Heapsort: Analysis of Step 2

Fixing up heaps where only first element is bad

$\Rightarrow$ Cost $O(h) = O(\lg n) \Longleftarrow$ one fix

How many fixes (iterations)? n

$$\underline{O(n \lg n)}$$

Heapsort $O(n \lg n + n)$

$$\underline{\underline{O(n \lg n)}}$$

Heapsort

- Achieves guaranteed $O(n \lg n)$ performance
- Sorts in place: $O(1)$ space
- But in the average case, quicksort still beats it :-(

O(n) Sorting :-)

- Sometimes we try so hard to solve something using the tools we *think* apply that we miss the opportunity to do it completely differently....

0 1 1 3 2 5 5 5 5

Histogram

0	1	2	3	4	5	6
1	2	1	1	0	4	0

$O(n)$

Read out   0 1 1 2 3 5 5 5 5 $O(n)$

$\underline{O(n)}$

Limitations

- We need to create an array for the histogram that runs from MIN_NUMBER...MAX_NUMBER
  - If this was just a java int than this is $-2^{31} \ldots (2^{31}-1)$
  - Assuming 4 bytes for each slot, that's a total cost of 2^{34} B = 2^{14} MB = 16 GB for the histogram (!)
  - i.e. it's O(MAX_NUMBER-MIN_NUMBER) in space!!

b   s

Scan max/min $O(n)$

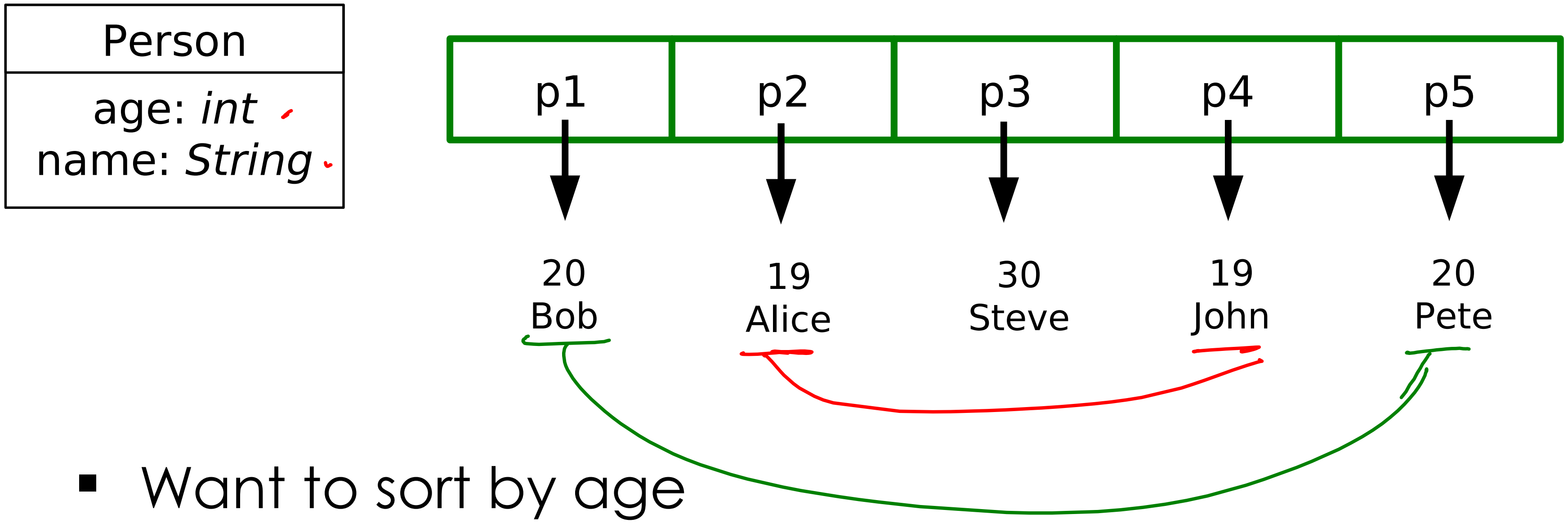
Construct hist $O(n)$

Read result $O(n)$

$O(n)$ perf.

Space $\underline{\underline{O(b-s)}}$

Stability



- Want to sort by age
  - But there are multiple sorted solutions:
    - p2, p4, p1, p5, p3 stable
    - p4, p2, p1, p5, p3
    - p4, p2, p5, p1, p3
    - p2, p4, p5, p1, p3

    Possible sorted by age
  - **Stable** algorithms preserve the order found in the input in these cases