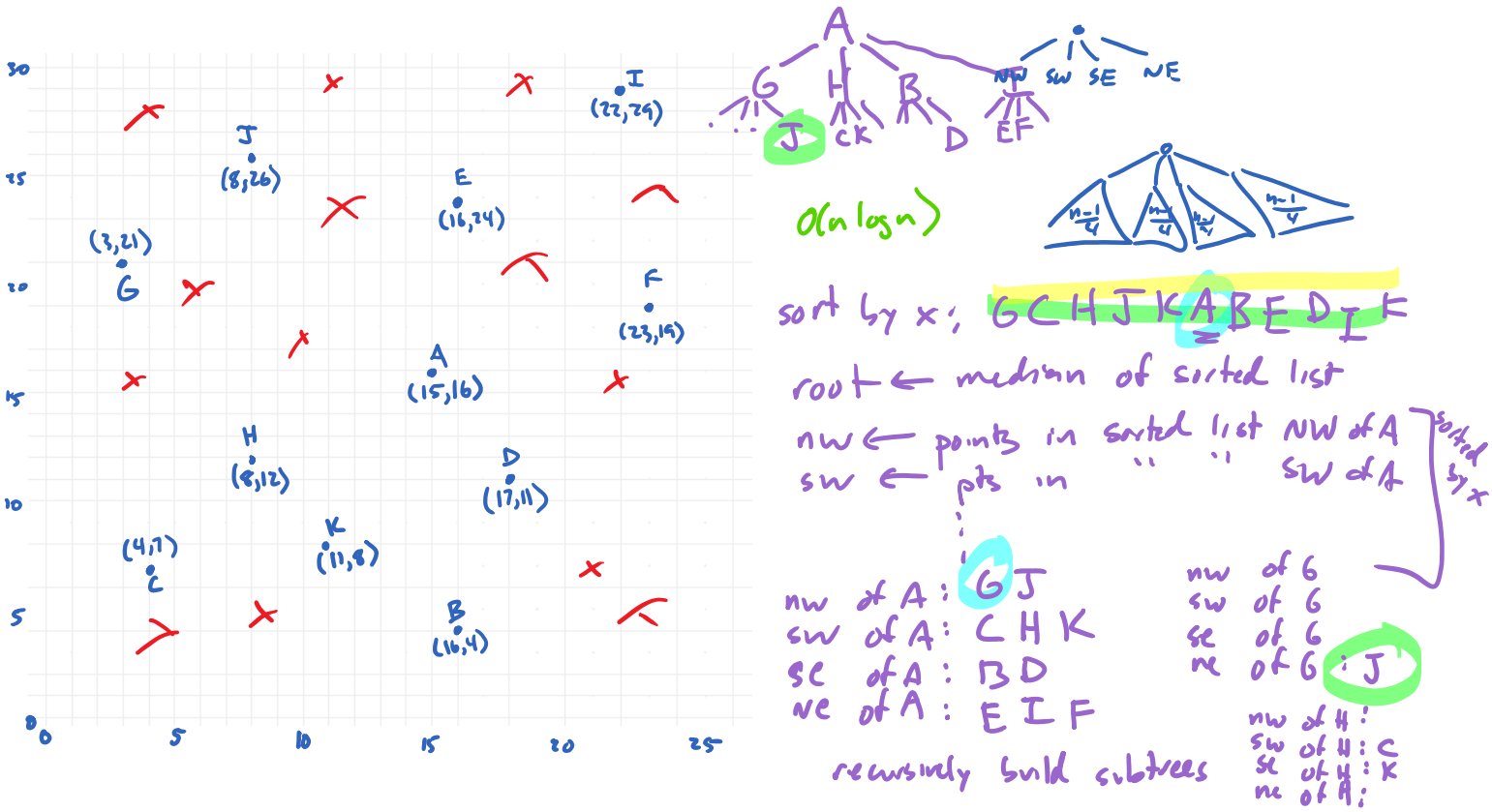
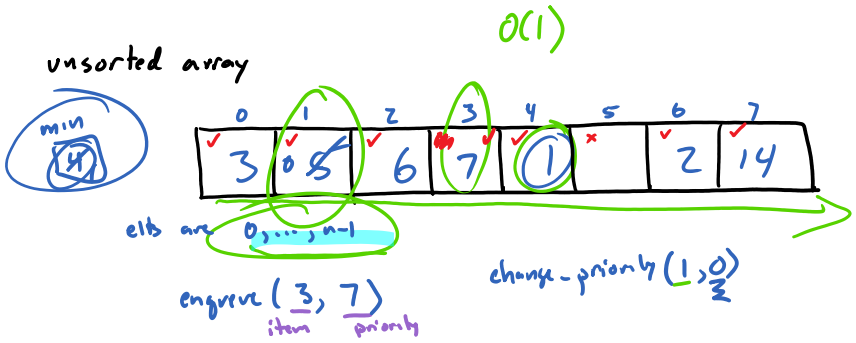
k-nearest

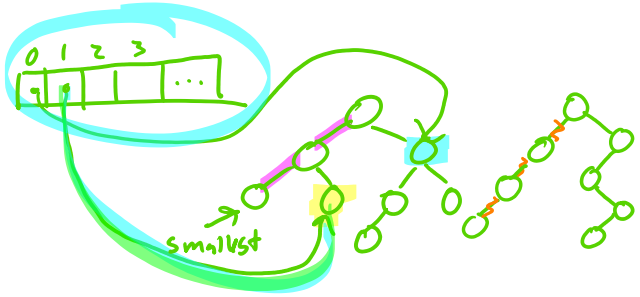

I (22,29)
J (8,26)
E (16,24)
(3,21) G
F (23,19)
A (15,16)
H (8,12)
D (17,11)
K (11,8)
(4,7) C
B (16,4)
0 5 10 15 20 25
A
G H B NW SW SE NE
J CK D EF
O(n log n)
sort by x: G C H J K A B E D I F
root ← median of sorted list
nw ← points in sorted list NW of A
sw ← pts in " " SW of A
sorted by x
nw of A: G J
sw of A: C H K
se of A: B D
ne of A: E I F
nw of G
sw of G
se of G
ne of G: J
nw of H:
sw of H: C
se of H: K
ne of H:
recursively build subtrees

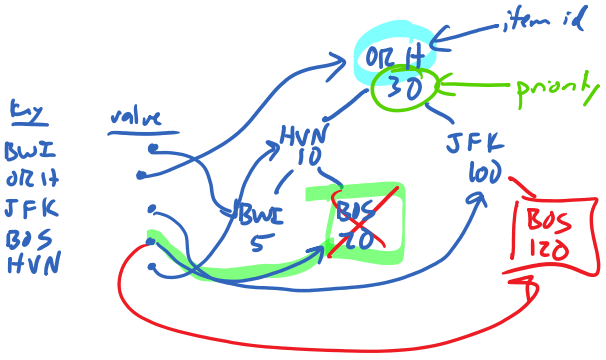
Priority Queue

by priority priority for order

	unsorted array	sorted array	balanced BST	heaps
enqueue (elt, pri)	O(1)	O(n)	O(log n)	O(log n)
dequeue ( )	O(n)	O(1) (ring buffer / wraparound)	O(log n)	O(log n)
change-priority (elt, pri)	O(1)	O(n)	O(log n)	O(log n)

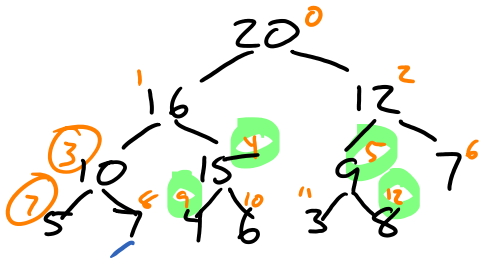






change-priority (BOS, 120)

Heap



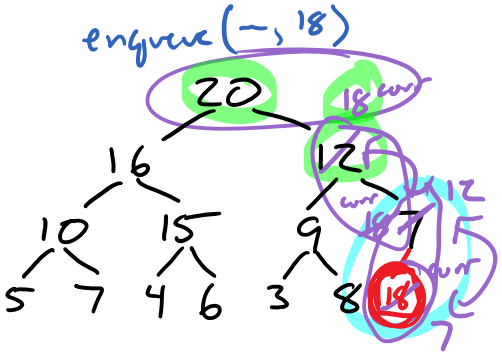
LEFT(n) = 2n+1
RIGHT(n) = 2n+2
PARENT(n) = $\lfloor \frac{n-1}{2} \rfloor$

curr = 8
curr = 17

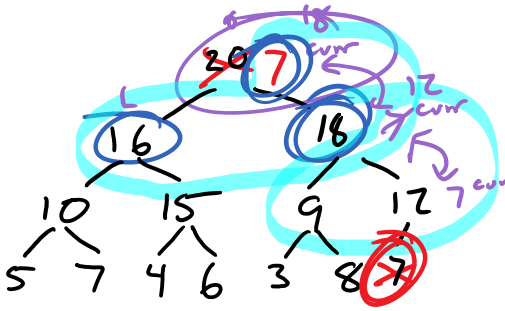
binary tree s.t.

shape: tree is nearly complete
↳ as few levels as possible + leaves as far down/left as possible

(priorities = values for priority queue)
order: value in a node ≥ values in children (highest value in root) — max-heap
≤ (lowest) — min-heap



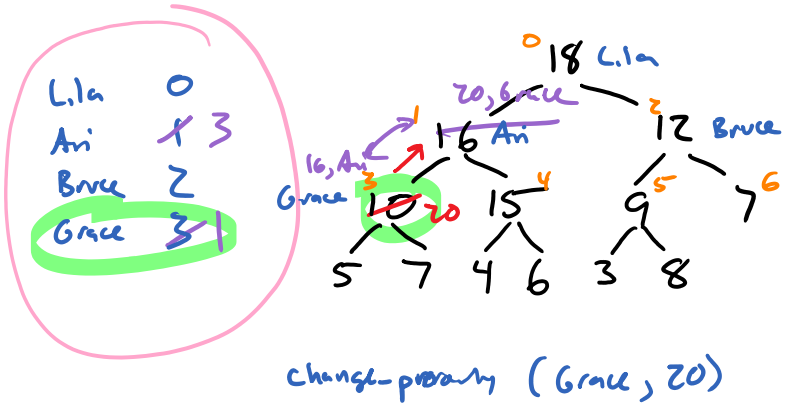
enqueue(18, 18)



dequeue()

enqueue: add at next available spot O(1)
curr ← added node O(1)
while curr not root and curr's value > parent's value: O(h) / O(log n)
  swap(curr, parent) O(1) per iter
  move curr up one level
O(log n) overall — reheap-up
[amortized from array resize]

dequeue: get max priority item from root
move last leaf to root O(1) (shrinking heap)
curr ← root
while curr not a leaf and curr < one child: O(log n) iterations
  swap(curr, largest child) O(1) per iteration
  curr follows old parent
O(log n) overall [amortized from array resize] — reheap-down



change_priority (Grace, 20)

Heapsort

in place (no O(1) extra space)

1) build max heap from array $O(n)$

2) extract max repeatedly from array $O(n \log n)$

total $O(n \log n)$