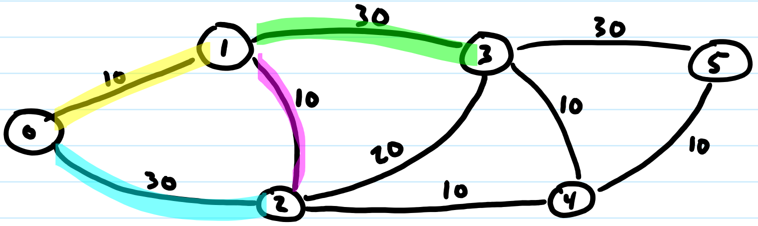
Shortest Paths in Weighted Graphs



adjacency list

0 : (1, 10), (2, 30)
1 : (0, 10), (2, 10), (3, 30)
2 : (0, 30), (1, 10), (4, 10), (3, 20)
3 : (1, 30), (5, 30), (4, 10), (2, 20)
4 : (3, 10), (2, 10), (5, 10)
5 : (3, 30), (4, 10)

Shortest Path 0 ⤳ 5

↳ least total weight

0: runners 0 → 1 (arrives @ 10) ✓
~~0 → 2 (arrive @ 30)~~

10: runner 0 → 1 arrives
dispatch 1 → 3 (arrives @ 40)
1 → 2 (arrives @ 20) beats runner 0 → 2

keep track for each city of earliest arrival
source of earliest arrival
whether runners sent

Priority Queue : maintain keys and priorities
vertices arrival times (distances)

build queue : given keys and initial priorities, initialize
vertices ∞ for all except 0 for source

decrease priority : given key, decrease its priority

extract minimum : get key with lowest priority and remove it

simple implementation : unsorted array

integer items are indices
value @ index is priority

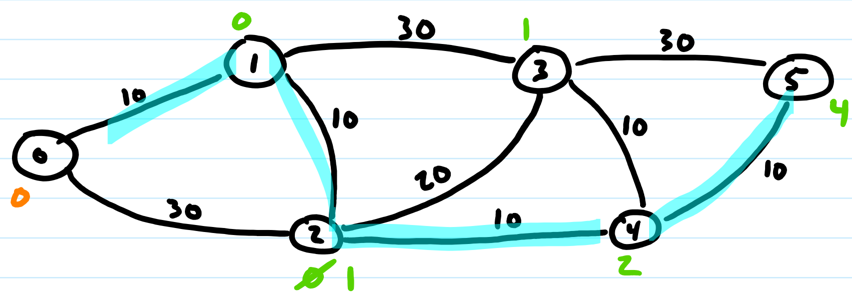
BFS

```
BFS(V,E,s)

for each vertex u in V
   color[u] <- WHITE
   dist[u] <- infinity
   pred[u] <- NULL

color[s] <- GRAY
dist[s] <- 0
pred[s] <- NULL
Q <- [s]  build queue with priorities = dist

while not Q.is_empty()
   u <- Q.dequeue()  extract-min
   for each v such that (u, v) is an edge
      if color[v] = WHITE  color[v] ≠ BLACK and dist[u] + w(u,v) < dist[v]
         pred[v] <- u
         dist[v] <- dist[u] + 1  w(u,v)
         color[v] <- GRAY
         Q.enqueue(v)  decrease-priority
   color[u] <- BLACK
```



shortest path 0 ⇝ 5

	0	1	2	3	4	5
init	~~0~~	∞	∞	∞	∞	∞
extract 0		~~10~~	30	~~∞~~	∞	∞
extract 1			~~20~~	40	∞	∞
extract 2				40	~~30~~	∞
extract 4				~~40~~		40
extract 3						~~40~~
extract 5						

Priority Queue Time Complexity

	unsorted array	balanced BST (use priority for order)	heap
build queue	add all priorities to array $O(n)$	$O(n \log n)$	$O(n)$
decrease priority	$O(1)$ update value @ index	$O(\log n)$ delete readd	$O(\log n)$
extract minimum	$O(n)$ search entire array	$O(\log n)$ find leftmost	$O(\log n)$

simple implementation : unsorted array

integer items are indices
value @ index is priority

0	1	2	3	4	5
0	~~∞~~ 10	~~∞~~ 30	∞	∞	∞

Dijkstra : for least-cost paths in graphs w/no neg-weight edges

```
Dijkstra(V,E,s)

for each vertex u in V
  color[u] <- WHITE
  dist[u] <- infinity
  pred[u] <- NULL

color[s] <- GRAY
dist[s] <- 0
pred[s] <- NULL
Q <- pq_build(n, dist)

while not Q.is_empty()
  u <- Q.extract_min()   <- n times (assuming all verts reachable)
  for each v such that (u, v) is an edge
    if color[v] != BLACK and dist[u] + w(u, v) < dist[v]   <- m times
      pred[v] <- u
      dist[v] <- dist[u] + w(u, v)
      color[v] <- GRAY
      Q.decrease_priority(v, dist[v])
  color[u] <- BLACK
```

worst case m times

for unsorted array (and adj list) : $m \cdot O(1) + n \cdot O(n) + O(n)$

$O(n^2 + m)$

$O(n^2)$

for balanced BST (and heap) : $m \cdot O(\log n) + n \cdot O(\log n) + O(n \log n)$

$O((m+n) \log n)$

$O(m \cdot \log n)$

for $m = \Theta(n)$: unsorted array $O(n^2)$ balanced BST or heap $O(n \log n)$

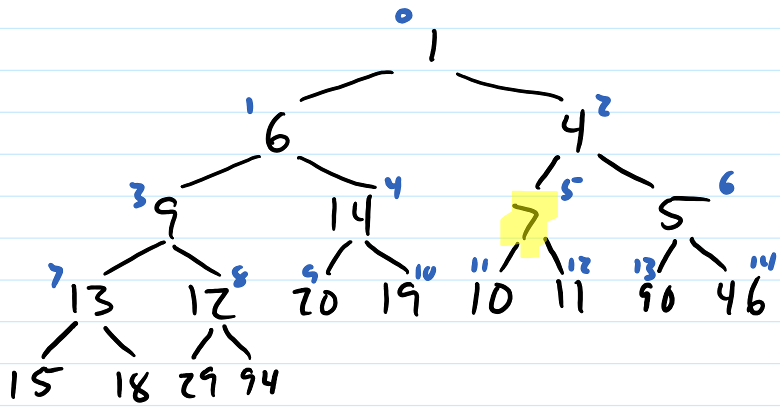
~~$m = O(n^2)$~~ unsorted array $O(n^2)$ balanced BST or heap $O(n^2 \log n)$

(Fib heap : $O(n^2)$ for dense, $O(n \log n)$ for sparse)

Heaps:
(Min-heap)

shape: complete binary tree
↳ every level except last is full
all leaves as far left as possible

order: value at node $\leq$ value in children
(so min item is in root)



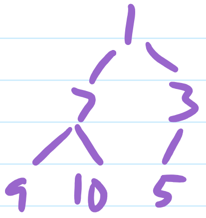
can compute indices of children/parent from index of current node

$curr = 5$
$left = 2 \cdot curr + 1$
$right = 2 \cdot curr + 2$
$parent = \left\lfloor \frac{curr - 1}{2} \right\rfloor$

0	1	2	3	4	5	6												18	
1	6	4	9	14	7	5	13	12	20	19	10	11	90	46	15	18	29	94	

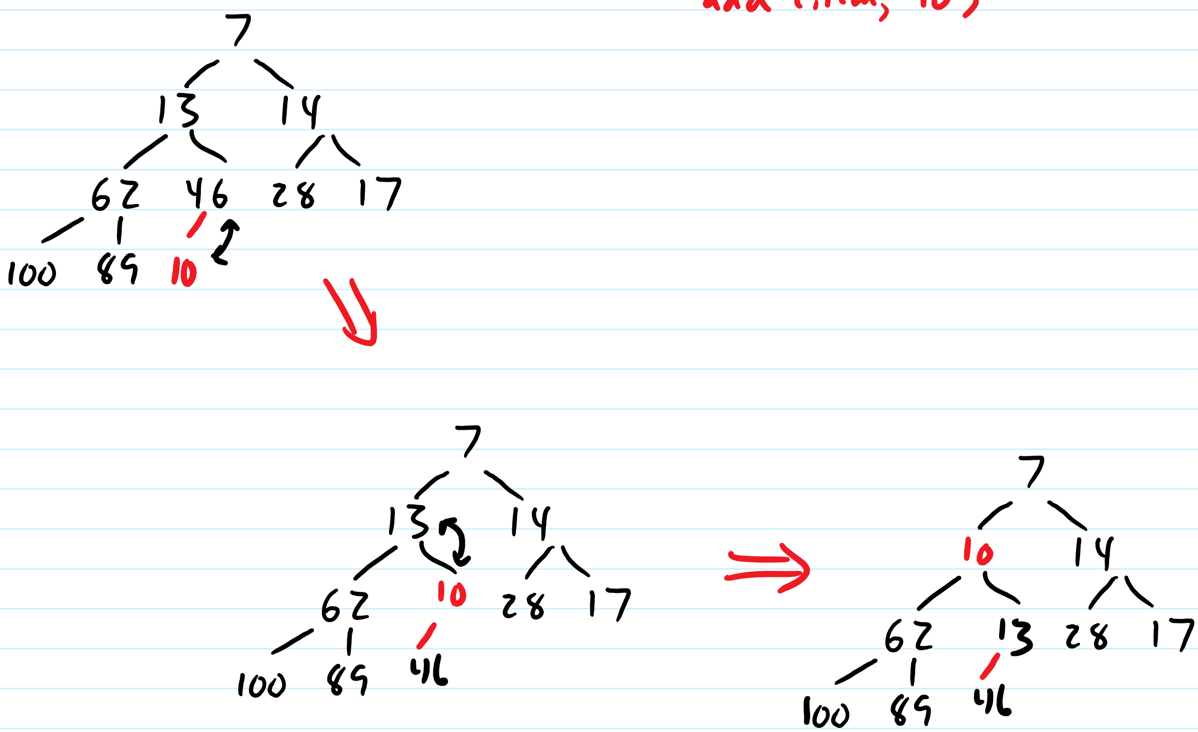
1	7	3	9	10	5

⇓



HEAP-ADD

add (item, 10)



HEAP-ADD(H, k)

A.heap-size ← A.heap-size + 1

A[A.heap-size] ← k

REHEAP-UP(A.heap-size)

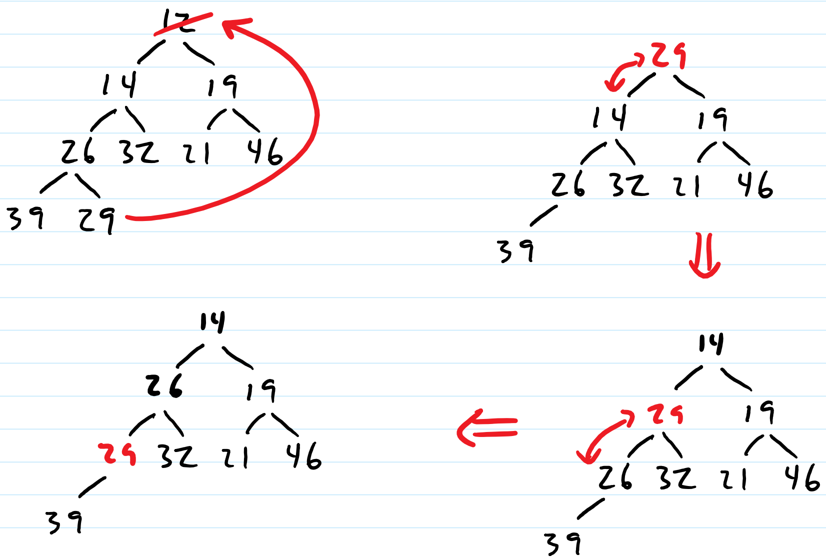
REHEAP-UP (A, i)

while $(\underbrace{i > 0}_{\text{not root}}$ and $\underbrace{A[i] < A[\text{parent}(i)]}_{\text{order violation w/parent}})$ — O(h) = O(log n) iterations

O(1) { swap A[i], A[parent(i)]
i ← parent(i) } — repeat at loc new item swapped into

O(1) per iteration, so O(log n) overall

EXTRACT-MIN



EXTRACT-MIN(A)

min ← A[0]   get min from root

swap (A, 0, A.heap-size - 1)   swap last node into root

A.heap-size ← A.heap-size - 1

HEAPIFY (A, 0)   restore order property

HEAPIFY (A, i)

O(h) = O(log n) iterations

while (left(i) < A.heap_size)   not a leaf

O(1):

smallest ← left(i) if A[left(i)] ≤ A[right(i)] or right(i) ≥ A.heap_size   set index of smallest child

if A[i] > A[smallest]   order violation

swap A[i], A[smallest]

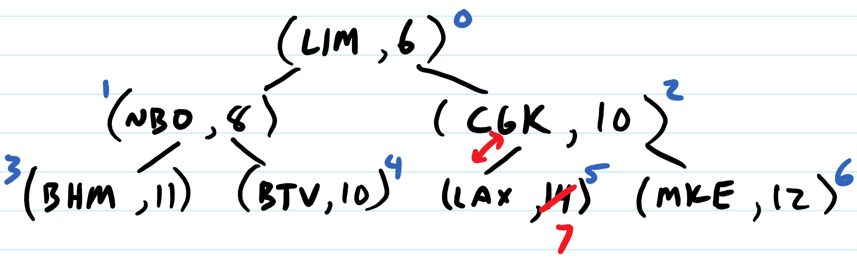
i ← smallest   repeat at loc swapped into

else

i ← A.heap-size   break loop (no order violation)

O(lg n) overall

CHANGE-PRIORITY



map

key	value	
NBO	1	
BTV	4	
LIM	0	
BHM	3	
MKE	6	
CGK	~~2~~	5
LAX	~~5~~	2

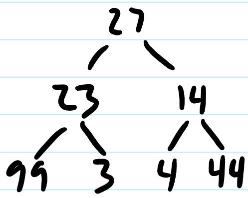
decrease-priority (LAX, 7)

$O(1)$ get index of key from map
$O(1)$ update priority if < current priority
$O(\log n)$ REHEAP_UP (modify to update map on each swap)

$O(\log n)$ overall

$O(1)$ to get/update map (expected if keys are not 0...n-1 so using hash table)

BUILD-HEAP (A)



last non-leaf

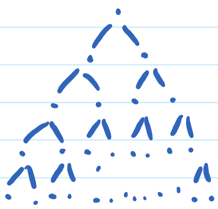
for $i \leftarrow \left\lfloor \frac{A.heap\_size - 1}{2} \right\rfloor$ downto 0 $O(n)$ iterations

heapify (A, i)

$O(\log n)$ per iteration

$O(n \log n)$ overall

(can show $O(n)$ overall w/ more careful math)



$\sim \frac{n}{4}$ nodes at next-to-last level,   1 iteration inside HEAPIFY

$\sim \frac{n}{8}$ nodes at next level up   2 iterations

$\sim \frac{n}{16}$ 3

$\vdots$ $\vdots$

total iterations in HEAPIFY $\approx \frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \frac{n}{16} \cdot 3 + \cdots$

$$= n \cdot \sum_{i=1}^{\infty} i \cdot \left(\tfrac{1}{2}\right)^{i+1}$$

for $0 \leq r < 1$ $\quad \sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$

$\frac{d}{dr}$ both sides $\quad \sum_{i=0}^{\infty} i \cdot r^{i-1} = \frac{1}{(1-r)^2}$

$$= \frac{1}{4} n \cdot \sum_{i=0}^{\infty} i \cdot \left(\tfrac{1}{2}\right)^{i-1}$$

$$= O(n)$$

$O(1)$ work per iteration, so $O(n)$ overall