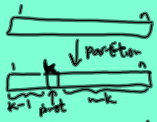
1. Finish analysis of randomized QuickSort
2. Heaps + Heapsort



$E[T(n)]$ = expected time to QS. n elements

$= E(n)$

$T(n)$ is a r.v.

For $k = 1, \ldots, n$, let X_k be the indicator r.v. for the event, "the pivot ends up at index k after the first partition"

Then, (time assigning pivot to index k)

$$T(n) = \underbrace{\Theta(n)}_{\text{time for partition}} + \sum_{k=1}^{n} \left(T(k-1) + T(n-k)\right) X_k$$

$$= n + \sum_{k=1}^{n} \underbrace{\left(T(k-1) + T(n-k)\right) X_k}_{\text{time assuming pivot at index } k}$$

$$E(n) = E[T(n)]$$

$$= \Theta(n) + \sum_{k=1}^{n} E\left[\left(T(k-1) + T(n-k)\right) X_k\right] \quad \text{(independent)}$$

$$= n + \sum_{k=1}^{n} E\left[T(k-1) + T(n-k)\right] \underbrace{E[X_k]}_{\text{prob}\{\text{index } k\} = 1/n}$$

$$= n + \frac{1}{n} \sum_{k=1}^{n} \left(E(k-1) + E(n-k)\right)$$

$$= \boxed{n + \frac{2}{n} \sum_{k=0}^{n-1} E(k) = E(n)}$$

recurrence on $E(n)$

Substitution method:

$E(n) = \Theta(n \lg n)$

Assume $E(m) \le c m \lg m$ for $m < n$.

$$E(n) = n + \frac{2}{n} \sum_{k=0}^{n-1} E(k)$$

$$= n + \frac{2}{n} \sum_{k=1}^{n-1} E(k)$$

$$\le n + \frac{2}{n} \sum_{k=1}^{n-1} c k \lg k$$

$$\le n + \frac{c2}{n} \lg n \sum_{k=1}^{n-1} k$$

$$= n + c \lg n \left(\frac{2}{n} \cdot \frac{n(n-1)}{2}\right)$$

$$= n + (c \lg n)(n-1)$$

$$= cn \lg n \underbrace{- c \lg n + n}_{> 0}$$

$$\not\le cn \lg n$$

Try: $E(n) \le cn \lg n - dn$

... sol'n to be posted.

Heaps & Heapsort

Priority queue:

collection of items where each item x has a numerical value $x.key$

Basic operations supported:

- Insert new element x into the priority queue
- Delete an element with min key value
- Find element with min key value (passive)
- Decrease the key of an element.

Sep 22-12:31 PM

A heap (minheap) is an implementation of a priority queue. A maxheap is like a minheap, but sense of ordering is reversed.

* Find/remove max
* Increase key

Binary heap: array $H[1..n]$ (n = capacity of the heap)

Actual portion used is $H[1..H.size]$

Assume elements of H are just the keys (numbers)

(Any satellite data has constant size)

(otherwise, just store pointer to satellite data with the heap item.

2 kinds: min heap, max heap
(describe) (similar)

An array $H[1..s]$ is in min heap order iff the following holds:

For every $i \in \{1,...,s\}$,

$H[i] \leq H[2i]$ (assuming $2i \leq s$)

$H[i] \leq H[2i+1]$ (assuming $2i+1 \leq s$)

A binary (min)heap is just an array of numbers satisfying (min) heap order.

	1	2	3	4	5	6	7	8
H:	10	13	11	14	17	11	18	16

[equivalently: For all $j \in \{2,...,s\}$

$H[\lfloor \frac{j}{2} \rfloor] \leq H[j]$

Full binary tree layout of H:

write entries of H as nodes of a binary tree, by increasing level and left→right within each level.

For $i \geq 1$, define

$left(i) := 2i$

$right(i) := 2i+1$

For $j \geq 2$, define

$parent(j) := \lfloor \frac{j}{2} \rfloor$

Heap order property:

For all $2 \leq j \leq s$

$H[parent(j)] \leq H[j]$

Find Min — passively returns item with min key value

return H[1].

Time = O(1).

Insert(x, H)

// assume H.size < capacity

H.size ++

$H[H.size] := x$

$j := H.size$

while $j \geq 2$ and $H[j] < H[parent(j)]$ do

$H[j] \leftrightarrow H[parent(j)]$

$j := parent(j)$

// Invariant: $H[j] < x$

end while

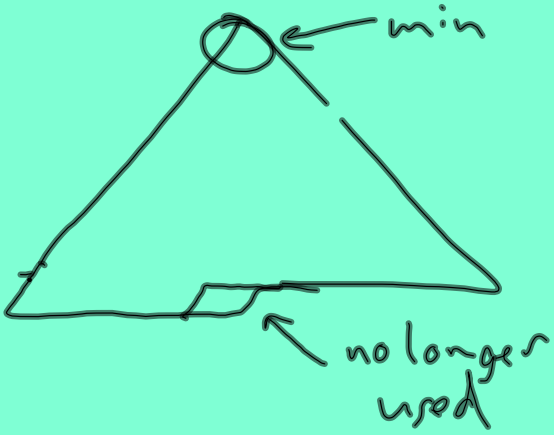
"cascading up"

DeleteMin

Sep 22-1:04 PM

Delete Min

// assume H.size > 0



H.size --

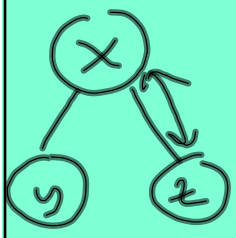
temp := H[H.size + 1]

min := H[1]

H[1] := temp

j := 1

while H[j] has a child that is < H[j]



H[j] ⟷ H[whichever child is smaller]

end-while

return min

y < x

z < x

Sep 22-1:36 PM