Ex 4-1. ✓ 7/7

Ex 4-2.

6/6

1 2 1 4 1 1 1 8 1 1 1 1 1 1 1 16

for $2 \rightarrow \frac{1+2}{2}$

$4 \rightarrow \frac{2.1+2+4}{4}$

$8 \rightarrow \frac{5.1+4+2+8}{8}$

$16 \rightarrow \frac{12.1+16+8+4+2}{16}$

goes to n

$32 \rightarrow \frac{27.1+32+16+8+4+2}{32}$

$\leq n$

$O(n) = \frac{n.1 + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \cdots}{n}$

$= \frac{n\left(1+\frac{1}{2}+\frac{1}{4}\cdots\right)}{n} = \frac{n.2}{n}$

$= 2$

$T(n) = O(2)$ which is

$O(1)$ since it is constant.

$\Rightarrow$

Ex 4-3.

6/6

Say if i is not power of 2 such as 3, 5 etc. we take "-" 2 of that operation else we take normally so we have,

-2 2 -2 4 -2 -2 -2 8 -2 -2 -2 -2 -2 -2 -2 16

-2 -2 -2 -2

Here each operation is withdraw 2$ from bank, but if power of 2, bank has 2$ fee for deposit of money. So that money is 0 at start and we can't withdraw since no money is left we deposit 2$ but fee is 2$ so we have 0. ... We deposit 4 it becomes 2 then we withdraw 2 it is 0, ... it is 8-2 we withdraw ... then ...

Another example could be, we take 1 from bank and there is no fee but when we deposit bank has fixed fee of $1 so as a result, at each deposit day we have $0 of our bank account.

-1 2 -1 4 -1 -1 -1 8 -1 -1 -1 -1 -1 -1 -1 16

-1 -1 -1

Exercise 4-4.

Suppose each operation increases potential by $i \% 2$ where i is ith op and we take remainder of i when divided to 2.

For non 2^i we have $\Phi = 1$ and if 2^i th op we have $\Phi = 0$

$\hat{c}_i = 1 + 1 - 0 = 2$ or $\hat{c}_i = 1 + 0 - 1 = 0$ so we have amortized
(c_i c_{i-1} for the first; c_i c_{i-1} for the second)

↓ $i \% 2 \neq 0$ — if $i \% 2 = 0$

OP cost

cost $O(1)$ since going to infinity wouldn't affect.

3/6

Exercise 4-5. read ✓

~~7~~ 7

Ex 4-6.

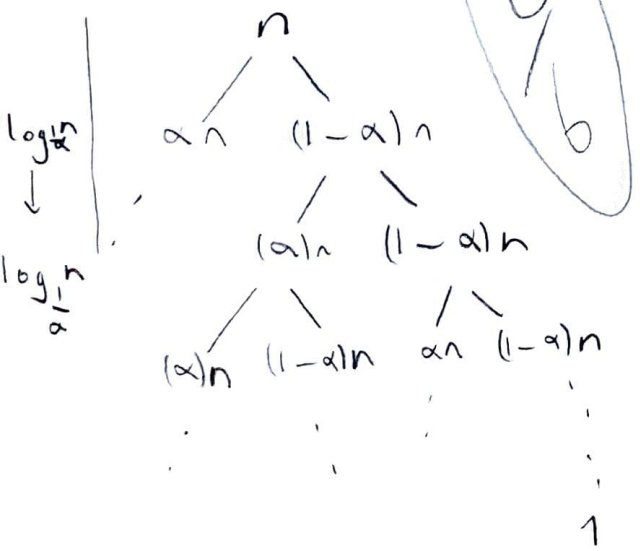
6/6

Since in partition we choose a pivot and for each element in the array we compare pivot to that element and in the data structure we have n elements to compare that results in n operations. And after comparison putting value to array is O(1) which results in a linear time complexity that is n * O(1) which is $T(n) = O(n)$.

Ex 4-7.

6/6

$$T(n) = T((1-\alpha)n) + T(\alpha n) + cn$$



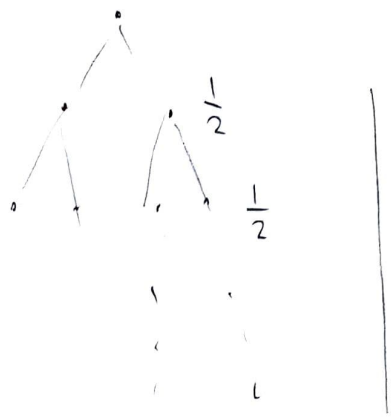
min depth is $\log_{\frac{1}{\alpha}} n$ since $\alpha \leq \frac{1}{2}$ and if we divided always with α at $\log_{\frac{1}{\alpha}} n$ we would reach node that is 1. If α was $\frac{1}{2}$ than $\log_2 n$ step would be 1 for example.

$$\frac{-\lg n}{\lg a} = -\log_{\alpha} n = \log_{\frac{1}{\alpha}} n$$ sowe concluded.

With some logic max depth would be $\log_{\frac{1}{1-\alpha}} n$ which is equal to $\rightarrow \frac{-\lg n}{\lg(1-\alpha)} = -\log_{(1-\alpha)} n = \log_{\frac{1}{1-\alpha}} n$ //

Ex 4-8.

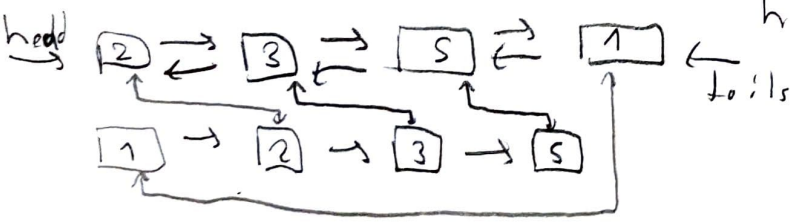
Since in randomized quicksort we throw a coin and chances are $\frac{1}{2}$ that we may either go 1 step deeper or stay at that depth. With tree



At worst case, we move for $\log n$ steps to deep. And, since in each step, we can choose n operations. $\rightarrow n \cdot \log n$

P 4-1.

a) This is kind of a linked list. We also have a list that is sorted. Each element in sorted points to its real value in normal list. This linked list is doubly linked list so we can do dequeue op. faster. A scheme is here, we also have head and tails pointers.



b) Enqueue: We are adding x, first x.next → head, head → x. We added x to normal list, we need to add it to sorted list. For that, from start while(x<curr curr→curr.next we iterate until we find right place, then we add x to there.

5/5

Dequeue: In main, tail → last element. prev,

for sorted (last element. sorted. prev. next = last element. sorted. next

free (last element. sorted), free (last element)

Find min: First element of sorted list.

c) In enqueue, insert to main list is trivial doubly linked list insert is sufficient. In sorted it is again same, for example we have → 2 ⇄ 1 ⇄ 5 ←

1 ⇄ 2 ⇄ 5

Enqueue(3) → 3 ⇄ 2 ⇄ 1 ⇄ 5 ←

1 ⇄ 2 ⇄ 3 ⇄ 5

2/5

Findmin ⟶ get first of sorted → 1.

Dequeue → 3 ⇄ 2 ⇄ 1 ←

→ 1 ⇄ 2 ⇄ 3

Algorithms are trivial and they work right.

d) • Enqueue: For the main insert we do constant number of operations so, it is $O(1)$. For the insertion to sorted list if our inserted element is high it might take $O(n)$ but,

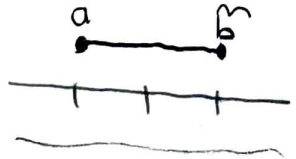
7/10

• Dequeue: We get to last element with tail in $O(1)$ with last element, sorted we delete it from sorted list in $O(1)$ since we can directly go to it. After doing constant number of ops for pointers we are done. This is then $O(1)$.

3. • Find min: Return head of sorted array it is $O(1)$.

P 4.2

a) Prob of choosing x_i first, $\rightarrow \frac{1}{n}$ since n many numbers in subarray:



for subarray to be lower equals to $\frac{n \cdot 3}{4}$ at most, the x needs to be between a and b and since this is $\frac{1}{2}$ chance since $\frac{[a,b]}{[a,n]} = \frac{1}{2}$ we see this prob is at least $\frac{1}{2}$.

5/5

b)

$\frac{0}{9}$

Tol:

$\frac{64}{100}$