Problem Set 4

This problem set is due **at 11:59pm** on **Thursday, March 5, 2015.**

Please turn in each problem solution separately. Each submitted solution should start with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Exercise 4-1. Read CLRS, Chapter 17.

Exercise 4-2. Exercise 17.1-3.

Exercise 4-3. Exercise 17.2-2.

Exercise 4-4. Exercise 17.3-2.

Exercise 4-5. Read CLRS, Chapter 7.

Exercise 4-6. Exercise 7.1-3.

Exercise 4-7. Exercise 7.2-5.

Exercise 4-8. Exercise 7.4-4.

Problem 4-1. Extreme FIFO Queues [25 points]

Design a data structure that maintains a FIFO queue of integers, supporting operations ENQUEUE, DEUEUE, and FIND-MIN, each in $O(1)$ amortized time. In other words, any sequence of m operations should take time $O(m)$. You may assume that, in any execution, all the items that get enqueued are distinct.

(a) [5 points] Describe your data structure. Include clear invariants describing its key properties. *Hint:* Use an actual queue plus auxiliary data structure(s) for bookkeeping.

(b) [5 points] Describe carefully, in words or pseudo-code, your ENQUEUE, DEQUEUE and FIND-MIN procedures.

(c) [5 points] Prove that your operations give the right answers. *Hint:* You may want to prove that their correctness follows from your data structure invariants. In that case you should also sketch arguments for why the invariants hold.

(d) [10 points] Analyze the time complexity: the worst-case cost for each operation, and the amortized cost of any sequence of m operations.

17.1 Aggregate Analysis

1. multipop(S, k) ⟶ min(size(S), k) operation.
most O(n), n for size of S.
Any sequence of operations, is at most the number of push.
operation. O(n)/n → O(1).

2.
```
INCREMENT(A)
1  i = 0
2  while i < A.length and A[i] == 1
3      A[i] = 0
4      i = i + 1
5  if i < A.length
6      A[i] = 1
```

⟶ worst case O(k).
for n increment, O(nk).
not tight.
we see. A[0] change every time,
A[1] change ⌊n/2⌋ time
A[2] chang ⌊n/4⌋ time
.....

so $\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < \sum_{i=0}^{k-1} \frac{n}{2^i} = 2n.$

O(2n)/n ⇒ O(1).

17.1-3
Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

17.1-3.

oper ≤ n + n + $\frac{n}{2}$ + $\frac{n}{4}$ ..... 1.

≤ 3n . O(n)/n = 1.

17.2 The accounting method.
-. amortized cost ≥ actual cost.

stack:
push : 1
pop : 1
multipop : min(k, S)

⟹

push : 2 (save 1 credit every time for. latter use)
pop : 0.
multipop : 0

Increment 0 → 1 : 2, charge for 1 → 0.

17.2-2
Redo Exercise 17.1-3 using an accounting method of analysis.

17.2-2.

every op : 3.

if $i \neq 2^k$, charge 2.

if $i = 2^k$, pay by credit

$(2^{k-1}, 2^{k-1}+1, \cdots, 2^k) \times 2 = 2^k = i$.

2^{k-1} 个.

17.3 the potential method.

(associate the potential as a whole rather than specific objects).

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

so if define a potential function, that $\Phi(D_n) \geq \Phi(D_0)$, then amortized cost $\geq$ actual cost.

---

stack : Φ = # objects in stack.

binary counter : Φ = # 1

17.3-2
Redo Exercise 17.1-3 using a potential method of analysis.

17-3-2.

How to define Φ ?

1	2	3	4	5	6	7	8	9	...	16
	2		4				8			16.

Φ 会 cover 2^k.

normal : if $\hat{\Phi}(i) - \hat{\Phi}(i-1) = 3$.

2^k : $i + \Phi(i) - \Phi(i-1) = 2$

$\Phi = 2i - 2^{1+\lfloor \lg i \rfloor}$, for $i > 0$, and $\Phi(0) = 0$.

for operation 1 : $\Rightarrow 1 + 2 \times 1 - 2^1 - 0 = 1$.

so. amortized $\Theta(3n) \Rightarrow O(n)$.

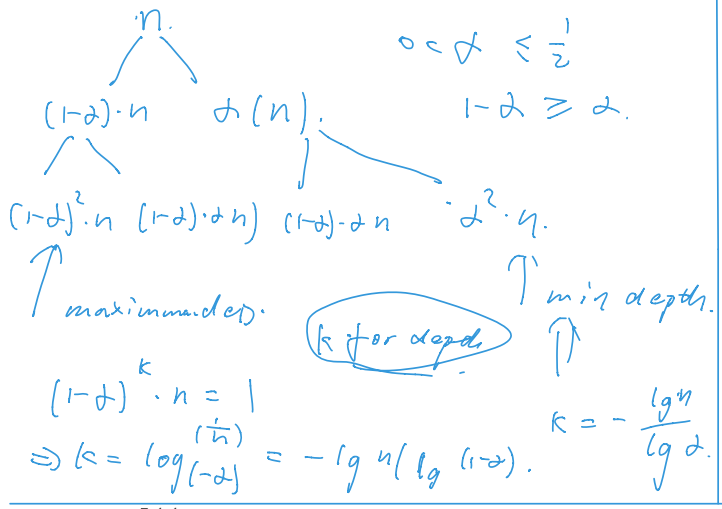
7. Quicksort

7.1-3
Give a brief argument that the running time of PARTITION on a subarray of size n is $\Theta(n)$.

7.1-3.
the cost is in for loop.
at most n. and at least n.
$\Rightarrow \Theta(n)$.

7.2-5.

7.2-5
Suppose that the splits at every level of quicksort are in the proportion $1-\alpha$ to α, where $0 < \alpha \le 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately $-\lg n / \lg \alpha$ and the maximum depth is approximately $-\lg n / \lg(1-\alpha)$. (Don't worry about integer round-off.)



QUICKSORT(A, p, r)

```
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Partitioning the array

The key to the algorithm is the PARTITION procedure, which rearranges the subarray $A[p..r]$ in place.

PARTITION(A, p, r)

```
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

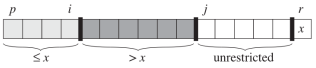


Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p..r]$. The values in $A[p..i]$ are all less than or equal to x, the values in $A[i+1..j-1]$ are all greater than x, and $A[r] = x$. The subarray $A[j..r-1]$ can take on any values.

RANDOMIZED-PARTITION(A, p, r)

```
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

7.4-4.

7.4-4
Show that RANDOMIZED-QUICKSORT's expected running time is $\Omega(n \lg n)$.

same as text book, z_i --- z_j
$\Pr(i \text{ compare to } j) = \Pr(i \text{ is pivot or } j \text{ is pivot})$.
because Pivot is randomly choose. so
$\Pr = \frac{2}{j-i+1}$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

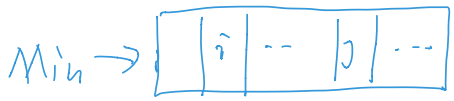
$k = j - i \Rightarrow \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\lg n)$$

$$= O(n \cdot \lg n).$$

4-1. a)

Queue for data.



ascending order linkedlist. Min.

two invariant:

① $x_i \le x_j$. for $i < j$. ($x_i < x_j$. increasingly)

② item x appear in min if x in main of.

---

b). Enqueue (x) some tail-segment of Main.

```
enque (Main, x).
while (Min is not empty and x < Min.last)
    detele Min.last.
enque (Min, x).
```

Dequeue:

```
x = deque(Main).
if x == Min.first
    deque(Min)
return x.
```

Find-Min:

```
return Min.first.
```

---

c) Enque keep the invariants.

1. while loop ensure ↑.
2. every item in min is minimum of some tail-segment of Main

Deque. delete if x == Min.first.

ensure variant 2. variant 1 never change

d.) Find-Min $\Theta(1)$ } easy.
Deque $\Theta(1)$

Enque. $\Theta(1)$. :)

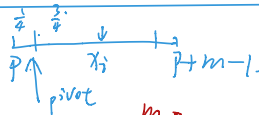
for main. 1.

for min, use counting method. enque: 2.

1 for enque, left 1 for delete.

so $\Theta(1)$.

4-2. $A[p \sim p+m-1]$ $len(A) = m$.



~~P_r (pivot = x_i || lents~~

suppose pivot $= x \in [\frac{m}{4}]+1 \le x \le m - [\frac{m}{4}]$, atleast $\frac{m}{2}$ valuse.

the two array split by pivot is at most $\frac{3}{4}m$, so $p \ge \frac{1}{2}$.

if $x_i \in$ range. then x_i is pivot or in one of two subarray will len $(\cdots) \le \frac{3}{4}m$

$P_r \ge \frac{1}{2}$.

---

b) $(12+c)\cdot \lg n$ toss, at least $c \cdot \lg n$ heads, $P_r \ge 1 - \frac{1}{n^2}$

$(6+2c)\cdot \lg n$, atmost. $P_r \ge 1 - \frac{1}{n^2}$

$\Downarrow d = 6+2c$

compare x_i with pivot. $\le d \cdot \lg n$.

use a). $x_i \in$ array which len $\le \frac{3}{4}m$, $P_r \ge \frac{1}{2}$.

successful for a) match.

at most $\log_{\frac{4}{3}} n$ succeful call.

$c \lg n = \log_{\frac{4}{3}} n \Rightarrow c = \log_{\frac{4}{3}} 2$.

$c = \log_{\frac{4}{3}} 2$, $\alpha = 2$

with probabile at least $1 - \frac{1}{n^2}$ where have at least $c \lg n$.

succeed; the total call is $d \lg n$.
where $d = 3(2+c)$
≤ 14

Problem 4-2. Quicksort Analysis [25 points]

In this problem, we will analyze the time complexity of QUICKSORT in terms of error probabilities, rather than in terms of expectation. Suppose the array to be sorted is $A[1\,..\,n]$, and write x_i for the element that starts in array location $A[i]$ (before QUICKSORT is called). Assume that all the x_i values are distinct.

In solving this problem, it will be useful to recall a claim from lecture. Here it is, slightly restated:

Claim: Let $c > 1$ be a real constant, and let α be a positive integer. Then, with probability at least $1 - \frac{1}{n^\alpha}$, $3(\alpha + c) \lg n$ tosses of a fair coin produce at least $c \lg n$ heads.

Note: High probability bounds, and this Claim, will be covered in Tuesday's lecture.

(a) [5 points] Consider a particular element x_i. Consider a recursive call of QUICKSORT on subarray $A[p\,..\,p+m-1]$ of size $m \geq 2$ which includes element x_i. Prove that, with probability at least $\frac{1}{2}$, either this call to QUICKSORT chooses x_i as the pivot element, or the next recursive call to QUICKSORT containing x_i involves a subarray of size at most $\frac{3}{4}m$.

(b) [9 points] Consider a particular element x_i. Prove that, with probability at least $1 - \frac{1}{n^2}$, the total number of times the algorithm compares x_i with pivots is at most $d \lg n$, for a particular constant d. Give a value for d explicitly.

(c) [6 points] Now consider all of the elements $x_1, x_2, \ldots, x_n$. Apply your result from part (b) to prove that, with probability at least $1 - \frac{1}{n}$, the total number of comparisons made by QUICKSORT on the given array input is at most $d' n \lg n$, for a particular constant d'. Give a value for d' explicitly. *Hint:* The Union Bound may be useful for your analysis.

(d) [5 points] Generalize your results above to obtain a bound on the number of comparisons made by QUICKSORT that holds with probability $1 - \frac{1}{n^\alpha}$, for any positive integer α, rather than just probability $1 - \frac{1}{n}$ (i.e., $\alpha = 1$).

c) $1 - \left(\frac{1}{n^2}\right) \times n = 1 - \frac{1}{n}$, every value comare at most $d \lg n$

so total $d \cdot n \lg n$; $d' = d \leq 14$

d) $1 - \left(\frac{1}{n^{\alpha+1}}\right) \cdot n = 1 - \frac{1}{n^2}$.

$\Rightarrow d = 3(\alpha + 1 + c)$

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015