# Announcements

# Announcements

- HW6 is out, due May 1 (just updated today- typo in P1)
- For planning purposes:
  - Final exam will be scheduled the week of May 17-21
  - Will cover everything since the last midterm
  - Same structure as past 2 exams
  - Will be out May 9, due May 16

after classes end

# Announcements

- Clarification on algorithm design strategies

# Amortized Analysis

# In-Class Exercise: Implementing a Queue Using Stacks

- Show how to implement a queue using two stacks and O(1) extra space

EQ

for element x: 1. definitely pushed onto A
$1

cost:
$3/op
(worst case)
or
$3N for sequence

2. might get popped from A + pushed onto B: $2

3. if ↑ happens, it might get popped from B: $1

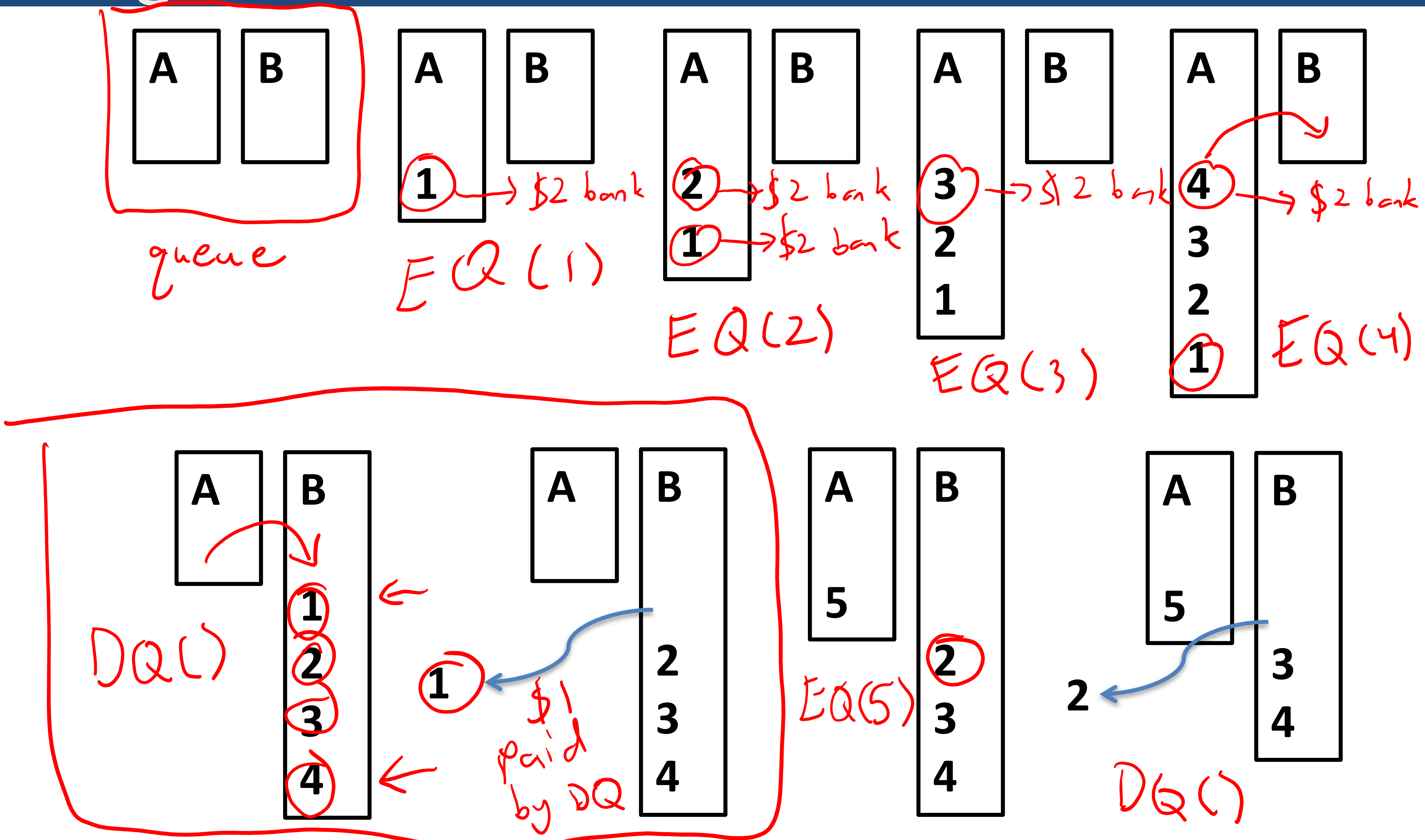
DQ

# In-Class Exercise: Implementing a Queue Using Stacks

- We will use 2 stacks to implement a queue
- Stacks A and B → behind-the-scenes
- Enqueue: push onto A
- Dequeue:
  - If B is empty, then move all the elements from A to B (using pops/pushes), then pop from B
  - If B is not empty, then pop from B
  - Each push/pop costs $1 → actual cost of atomic opers.

user-level operations

# In-Class Exercise: Implementing a Queue Using Stacks



# In-Class Exercise: Implementing a Queue Using Stacks

- Do an amortized running time analysis for a sequence of *n* enqueue/dequeue operations
- Hint: How many times, in total, can an element be pushed/popped onto some stack?

EQ(x) : push $1

DQ() : could be just pop

could be many pops/pushes + pop

EQ: $3 → $1 push
→ $2 bank

DQ: $1 → $1 for pop from B
→ rest comes from bank

# Randomized Algorithms

# Worst-Case Running Time

- So far, we have talked about *worst-case* running time ~~standard~~ standard + amortized
- $T_{WC}(N) = \max\{T(X)\text{: all inputs } X \text{ of size } N\}$

size of input

# Expected Running Time: Definition #1

weighted average

- We can also define ~~*expected*~~ (or *average-case*) running time

prob. that we see input X

- $T_{\cancel{wc}\ AC}(N) = \overset{\text{sum}}{\{}T(X) * Pr(X)\text{: all inputs X of size N}\}$
- Problem with this: it's hard to know the probability of getting a certain input
- Examples?

you will never have this unless you're dealing with very small N and a highly-predictable system

# Deterministic vs. Randomized Algorithms

no randomness

same running time

- A **deterministic** algorithm always makes the same sequence of actions when given the same input → same output
- A **randomized** algorithm bases its behavior not just on input but also random choices

# Expected Running Time: Definition #2

- We can define worst-case expected running time for randomized algorithms
- $T_{wc}(N) = \max\{E[T(X)] : \text{all inputs } X \text{ of size } N\}$

$T_{WCE}(N)$

$\lim_{t \to \infty} \frac{1}{t} \sum_{i=1}^{t} T_i(\cancel{X})$

Expected running time

→ weighted average

- Note how this is different from the previous definition of expected running time!

average

# Divide-and-Conquer: Median

- How do we find the median of an array?

  n　　unsorted

- Sort the array; pick the midpoint

- What is the running time of this?

  $O(n \log n)$

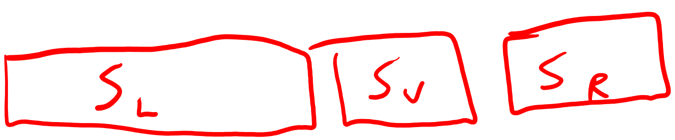
- But sorting does a lot of extra work… can we do better?

  [5 2 8 0 1 11 -1]

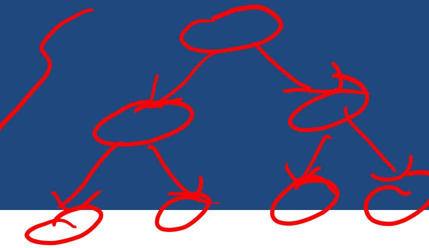
  [-1 0 1 2 5 8 11]

# Divide-and-Conquer: Median

- Let's define a more general problem: given an unsorted array $S$, find the $k^{th}$ smallest value of the array
- A divide-and-conquer solution to the above problem:
  - Pick a value $v$ from the array $S$ (how?)
  - Split $S$ into three arrays: $S_L$, $S_v$, and $S_R$
    - $S_L$ contains values smaller than $v$
    - $S_v$ contains values equal to $v$
    - $S_R$ contains values greater than $v$



# Divide-and-Conquer: Median



$k=1$ $v=36$ log n

- Example:

1 2 4 5 5 8 11 13 20 21 36

$S$ :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 36 | 5 | 21 | 8 | 13 | 11 | 20 | 5 | 4 | 1 |

$$v = 5$$

$S_L$ :

| | | |
|---|---|---|
| 2 | 4 | 1 |

$S_v$ :

| | |
|---|---|
| 5 | 5 |

$S_R$ :

| | | | | | |
|---|---|---|---|---|---|
| 36 | 21 | 8 | 13 | 11 | 20 |

- For a given *v*, what is the $\boxed{\text{time}}$ required to perform this split? $O(n)$ actual

deterministic

bad! N N-1 N-2 ⋮

# Divide-and-Conquer: Median

- From this split, how do we decide which array holds the ~~median~~?

k<u>th</u> smallest value?

# Divide-and-Conquer: Median

- From this split, how do we decide which array holds the ~~median~~?

kth smallest

$S_L$ $S_v$ $S_R$

kth

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

# Divide-and-Conquer: Median

- What is the worst-case running time of this method? (Hint: think about the size of the subproblem)

Suppose we pick v randomly

in the worst case, ~~recursion~~ problem only shrinks by 1 in each level of recursion. Running time is $O(n+n-1+n-2+\dots+1) = O(n^2)$

- What factor determines whether we'll be in the worst-case or not? ~~how much~~ where is v located in the (sorted) array. If v very small or very large, size of array is not reduced by much

# Divide-and-Conquer: Median

- So how can we strategically pick *v* to give us a good running time?
- In the best case, *v* should be the median!

$k^{\text{th}}$ smallest value – we don't know this!

- Idea: pick *v* randomly!
- What is the *expected* running time of this algorithm?
- Turns out to be O(*n*)!

even for worst case input

better than sorting and picking S[k]

# Proof Sketch

we do not know the median ahead of time!

What is a good choice of *v*?  Something close to the median. because this splits the problem in half

Let's define 'good' as a value in the 25th – 75th percentile (middle half). of v

of values

we do not know which values are "good" ahead of time!

What is the probability that a random *v* will be good? $\boxed{1/2}$

– when we pick a v we have no way of telling if it is good or bad!

# Proof Sketch

25th – 75th %

If you pick a 'good' *v*, how much will the array shrink by? by at least

at least 25%

$\boxed{S_L}$ $\boxed{S_v}$ $\boxed{S_R}$

| | $S_L$ | $S_R$ |
|---|---|---|
| v: 25% | 1/4 | 3/4 |
| v: 75% | 3/4 | 1/4 |

On average,

How many times do you have to pick *v* before you expect to find one that is 'good'?

2 because each draw has 50% chance of being "good"

$$E(\#\text{ draws}) = \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot\frac{1}{2}\cdot 2 + \cdots \approx 2$$

# Proof Sketch

$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

Write the expected running time as a recurrence relation

$$E[T(N)] = \underbrace{1 \cdot E\left[T\left(\frac{3N}{4}\right)\right] + O(2N)}_{\text{describing 2 levels of recursion tree}}$$

What is the expected running time?

$$E[T(N)] = O(N)$$