CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu

Dr. Oliver Kennedy
okennedy@buffalo.edu

212 Capen Hall

Day 14
Induction Review, Stacks and Queues
Textbook Ch. 15
Announcements

● PA2 is actually up now
  ○ Tests due 10/10
  ○ Full project due 10/17
Recap

QuickSort

- Divide and Conquer sorting algorithm like MergeSort
  - All of the work for Merge Sort happened during the combine step
  - QuickSort attempts to move the work to the divide step
- Divide: Move small elements to the left, and big elements to the right
- Conquer: Recursively call QuickSort on left and right halves
- Combine: ...nothing
Recap

QuickSort

- Divide and Conquer sorting algorithm like MergeSort
  - All of the work for Merge Sort happened during the combine step
  - QuickSort attempts to move the work to the divide step
- **Divide:** Move small elements to the left, and big elements to the right
- **Conquer:** Recursively call QuickSort on left and right halves
- **Combine:** ...nothing
QuickSort Review

**Divide:** Move *small* elements to the left and *big* elements to the right

How do we define what is *big* and what is *small*?
QuickSort Review

**Divide:** Move *small* elements to the left and *big* elements to the right

How do we define what is *big* and what is *small*?

Pick a pivot value
QuickSort Review

**Divide:** Move *small* elements to the left and *big* elements to the right

How do we define what is *big* and what is *small*?

**Pick a pivot value**

[ smaller than pivot ], pivot, [ larger than pivot ]
QuickSort Review

**Divide:** Move *small* elements to the left and *big* elements to the right

How do we define what is *big* and what is *small*?

**Pick a pivot value**

[smaller than pivot], pivot, [larger than pivot]

How do we pick a pivot?
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3, 4, 6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]

[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]

[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]

[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]

[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]

[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]

1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [11, 10, 9], 12, [14, 13, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [11, 10, 9], 12, [14, 13, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [11, 10, 9], 12, [14, 13, 15]
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, [14, 13, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]
[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]
[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]
1, 2, 3, 4, 5, 6, 7, 8, [11, 10, 9], 12, [14, 13, 15]
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, [14, 13, 15]
QuickSort Review

[4, 1, 8, 13, 12, 6, 2, 14, 7, 9, 3, 5, 11, 10, 15]

[4, 1, 7, 3, 6, 2, 5], 8, [14, 13, 9, 12, 11, 10, 15]

[1, 2, 3], 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]

1, 2, 3, 4, [6, 7, 5], 8, [14, 13, 9, 12, 11, 10, 15]

1, 2, 3, 4, 5, 6, 7, 8, [14, 13, 9, 12, 11, 10, 15]

1, 2, 3, 4, 5, 6, 7, 8, [11, 10, 9], 12, [14, 13, 15]

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, [14, 13, 15]

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
QuickSort Review

If our pivot was the median value, then our list would be split in half by the divide step, resulting in the same runtime as MergeSort $O(n\log(n))$. But finding the median value is expensive... (it also costs $n\log(n)$).

So what if we pick one randomly instead?
Expected Value

If I roll a 6-sided die, the probability of a particular side being rolled is \( \frac{1}{6} \).

If \( X \) is a random variable representing this die roll, then the expected value of \( X \) is:

\[
E[X] = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6
\]

\[
E[X] = \sum_{i=1}^{6} \frac{1}{6} \cdot i = 3.5
\]
Expected Value

If I roll a 20-sided die, the probability of a particular side being rolled is 1/20.

If $X$ is a random variable representing this die roll, then the expected value of $X$ is:

$$E[X] = \frac{1}{20} \cdot 1 + \frac{1}{20} \cdot 2 + \ldots + \frac{1}{20} \cdot 20 = \sum_{i=1}^{20} \frac{1}{20} i$$
Expected Value

If I roll an $n$-sided die, the probability of a particular side being rolled is $1/n$.

If $X$ is a random variable representing this die roll, then the expected value of $X$ is:

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

$$E[X] = \sum_{i} P_i \cdot X_i$$
Picking a pivot value randomly from the $n$ elements of our sequence is the same as rolling an $n$-sided die.

There is a $1/n$ probability in any particular value being selected.

$X = k$ means that $X$ is the $k$th largest value, and the expected value of $X$ corresponds to the median value.
QuickSort Review

\[ T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
T(0) + T(n - 1) + \Theta(n) & \text{if } n > 1 \land X = 1 \\
T(1) + T(n - 2) + \Theta(n) & \text{if } n > 1 \land X = 2 \\
T(2) + T(n - 3) + \Theta(n) & \text{if } n > 1 \land X = 3 \\
\vdots & \\
T(n - 2) + T(1) + \Theta(n) & \text{if } n > 1 \land X = n - 1 \\
T(n - 1) + T(0) + \Theta(n) & \text{if } n > 1 \land X = n 
\end{cases} \]
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
E[T(X - 1) + T(n - X)] + \Theta(n) & \text{otherwise}
\end{cases} \]
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
E[T(X - 1)] + T(n - X) + \Theta(n) & \text{otherwise}
\end{cases} \]

Expected value of two independent events can be split up
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
E[T(X - 1)] + E[T(n - X)] + \Theta(n) & \text{otherwise}
\end{cases} \]
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
E[T(X - 1)] + E[T(n - X)] + \Theta(n) & \text{otherwise} 
\end{cases} \]

How are these two terms related?
QuickSort Review

\[ E[T(X - 1)] \]
QuickSort Review

\[ E[T(X - 1)] \]

\[ = \sum_{i=1}^{n} P_i \cdot T(X_i - 1) \]
QuickSort Review

\[ E[T(X - 1)] \]

\[ = \sum_{i=1}^{n} P_i \cdot T(X_i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(i - 1) \]
QuickSort Review

\[ E[T(X - 1)] \]

\[ = \sum_{i=1}^{n} P_i \cdot T(X_i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(n - i) \]
QuickSort Review

\[ E[T(X - 1)] \]

\[ = \sum_{i=1}^{n} P_i \cdot T(X_i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(n - i) = E[T(n - X)] \]
QuickSort Review

\[ E[T(X - 1)] = \sum_{i=1}^{n} P_i \cdot T(X_i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(i - 1) \]

\[ = \sum_{i=1}^{n} \frac{1}{n} \cdot T(n - i) = E[T(n - X)] \]

They are equivalent!!
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
2E[T(X - 1)] + \Theta(n) & \text{otherwise}
\end{cases} \]
QuickSort Review

\[ E[T(n)] = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
2E[T(X - 1)] + \Theta(n) & \text{otherwise}
\end{cases} \]

Each \( T(X - 1) \) is independent, so the expected values can be split out.
QuickSort Review

$$E[T(n)] = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ \frac{2}{n} \left( \sum_{i=0}^{n-1} E[T(i)] \right) + \Theta(n) & \text{otherwise} \end{cases}$$
Inductive Proof
Hypothesis: $E[T(n)] \in O(n \log(n))$
Base Case: $E[T(2)] \leq c \cdot (2 \log(2))$
Base Case: $E[T(2)] \leq c \cdot (2 \log(2))$

$2 \cdot E_i[T(i - 1)] + 2c_1 \leq 2c$
Base Case: $E[T(2)] \leq c \cdot (2 \log(2))$

$$2 \cdot E_i[T(i - 1)] + 2c_1 \leq 2c$$

$$2 \cdot (T(0)/2 + T(1)/2) + 2c_1 \leq 2c$$
Base Case

Base Case: \( E[T(2)] \leq c \ (2 \log(2)) \)

\[
2 \cdot E_i[T(i - 1)] + 2c_1 \leq 2c
\]

\[
\n\text{Wrong}
\]

\[
2 \cdot (T(0)/2 + T(1)/2) + 2c_1 \leq 2c
\]

\[
T(0) + T(1) + 2c_1 \leq 2c
\]
**Base Case**

**Base Case:** \( E[T(2)] \leq c \cdot (2 \log(2)) \)

\[
2 \cdot E_i[T(i - 1)] + 2c_1 \leq 2c
\]

\[
2 \cdot (T(0)/2 + T(1)/2) + 2c_1 \leq 2c
\]

\[
T(0) + T(1) + 2c_1 \leq 2c
\]

\[
2c_0 + 2c_1 \leq 2c
\]
Base Case: $E[T(2)] \leq c \cdot (2 \log(2))$

$$2 \cdot E_i[T(i - 1)] + 2c_1 \leq 2c$$

$$2 \cdot (T(0)/2 + T(1)/2) + 2c_1 \leq 2c$$

$$T(0) + T(1) + 2c_1 \leq 2c$$

$$2c_0 + 2c_1 \leq 2c$$

True for any $c \geq c_0 + c_1$
Inductive Case

Assume: \( E[T(n')] \leq c \ (n' \log(n')) \) for all \( n' < n \)

Show: \( E[T(n)] \leq c \ (n \log(n)) \)
Inductive Case

Assume: \( E[T(n')] \leq c \ (n' \log(n')) \) for all \( n' < n \)

Show: \( E[T(n)] \leq c \ (n \log(n)) \)

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} E[T(i)] \right) + c_1 \leq cn \log(n)
\]
Inductive Case

Assume: \( E[T(n')] \leq c \ (n' \log(n')) \) for all \( n' < n \)

Show: \( E[T(n)] \leq c \ (n \log(n)) \)

Our \( i \) here is always less than \( n \), so we can use our assumption to substitute:

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} E[T(i)] \right) + c_1 \leq cn \log(n)
\]

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} ci \log(i) \right) + c_1 \leq cn \log(n)
\]
Inductive Case

Assume: \( E[T(n')] \leq c \ (n' \log(n')) \) for all \( n' < n \)

Show: \( E[T(n)] \leq c \ (n \log(n)) \)

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} E[T(i)] \right) + c_1 \leq cn \log(n) \\
\frac{2}{n} \left( \sum_{i=0}^{n-1} ci \log(i) \right) + c_1 \leq cn \log(n) \\
\frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n)
\]
Inductive Case

$$\frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n)$$
Inductive Case

\[ c \frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n) \]

\[ c \frac{2 \log(n)}{n} \left( \sum_{i=0}^{n-1} i \right) + c_1 \leq cn \log(n) \]
Inductive Case

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n)
\]

\[
\frac{2 \log(n)}{n} \left( \sum_{i=0}^{n-1} i \right) + c_1 \leq cn \log(n)
\]

\[
\frac{2 \log(n)}{n} \left( \frac{(n-1)(n-1+1)}{2} \right) + c_1 \leq cn \log(n)
\]
Inductive Case

\[ \frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n) \]

\[ \frac{2 \log(n)}{n} \left( \sum_{i=0}^{n-1} i \right) + c_1 \leq cn \log(n) \]

\[ \frac{2 \log(n)}{n} \left( \frac{(n-1)(n-1+1)}{2} \right) + c_1 \leq cn \log(n) \]

\[ \frac{\log(n)}{n} \left( n^2 - n \right) + c_1 \leq cn \log(n) \]
Inductive Case

\[
\frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n)
\]

\[
\frac{2 \log(n)}{n} \left( \sum_{i=0}^{n-1} i \right) + c_1 \leq cn \log(n)
\]

\[
\frac{2 \log(n)}{n} \left( \frac{(n - 1)(n - 1 + 1)}{2} \right) + c_1 \leq cn \log(n)
\]

\[
\frac{\log(n)}{n} \left( n^2 - n \right) + c_1 \leq cn \log(n)
\]

\[
 cn \log(n) - c \log(n) + c_1 \leq cn \log(n)
\]
Inductive Case

\[ \frac{2}{n} \left( \sum_{i=0}^{n-1} i \log(n) \right) + c_1 \leq cn \log(n) \]

\[ \frac{2 \log(n)}{n} \left( \sum_{i=0}^{n-1} i \right) + c_1 \leq cn \log(n) \]

\[ \frac{2 \log(n)}{n} \left( \frac{(n-1)(n-1+1)}{2} \right) + c_1 \leq cn \log(n) \]

\[ \frac{\log(n)}{n} (n^2 - n) + c_1 \leq cn \log(n) \]

\[ cn \log(n) - c \log(n) + c_1 \leq cn \log(n) \]

\[ c_1 \leq c \log(n) \]
So...is QuickSort $O(n \log(n))$...?

No!
What guarantees do you get?

If \( f(n) \) is a **Tight Bound**
- The algorithm always runs in \( cf(n) \) steps

If \( f(n) \) is a **Worst-Case Bound**
- The algorithm always runs in at most \( cf(n) \)

If \( f(n) \) is an **Amortized Worst-Case Bound**
- \( n \) invocations of the algorithm always run in \( cnf(n) \) steps

If \( f(n) \) is an **Average Bound**
- ...we don't have any guarantees
mutable.Seq ADT

mutable.IndexedSeq (ie Array)
Efficiency apply(), update()

mutable.Buffer (ie ArrayBuffer, ListBuffer)
Efficiency apply(), update(), append()
Stacks

A stack of objects on top of one another

**Push** Put a new object on top of the stack

**Pop** Remove the object on top of the stack

**Top** Peek at what's on top of the stack
Stacks

s.push("Bob")
Stacks

s.push("Bob")

s.push("Mary")
Stacks

```javascript
s.push("Bob")
s.push("Mary")
s.push("Sue")
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Sue&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Mary&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stacks

```javascript
s.push("Bob")
s.push("Mary")
s.push("Sue")
s.pop()
```

![Output]("Mary"

"Bob")
Stacks

s.push("Bob")
s.push("Mary")
s.push("Sue")
s.pop()
s.push("Steve")

| "Steve" |
| "Mary" |
| "Bob"  |
Stacks

```javascript
s.push("Bob")
s.push("Mary")
s.push("Sue")
s.pop()
s.pop()
s.push("Steve")
s.push("Bob")
```

"Mary"

"Bob"
Stacks in Practice

- Storing function variables in a "call stack"
- Certain types of parsers ("context free")
- Backtracking search
- Reversing Sequences
trait Stack[A] {
  def push(element: A): Unit
  def top: A
  def pop: A
}
Stacks

class ListStack[A] extends Stack[A] {
  val _store = new SinglyLinkedList()

  def push(element: A): Unit =
    _store.prepend(element)

  def top: A =
    _store.head

  def pop: A =
    _store.remove(0)
}
class ListStack[A] extends Stack[A] {
  val _store = new SinglyLinkedList()

  def push(element: A): Unit =
    _store.prepend(element)

  def top: A =
    _store.head

  def pop: A =
    _store.remove(0)
}
Stacks

class ListStack[A] extends Stack[A] {
  val _store = new SinglyLinkedList()

  def push(element: A): Unit = _store.prepend(element) \(\Theta(1)\)

  def top: A = _store.head \(\Theta(1)\)

  def pop: A = _store.remove(0) \(\Theta(1)\)
}

What is the runtime?
Stacks

class ArrayBufferStack[A] extends Stack[A] {
  val _store = new ArrayBuffer()

  def push(element: A): Unit =
    _store.append(element)

  def top: A =
    _store.last

  def pop: A =
    _store.remove(_store.length-1)
}
class ArrayBufferStack[A] extends Stack[A] {
    val _store = new ArrayBuffer()

    def push(element: A): Unit =
        _store.append(element)

    def top: A =
        _store.last

    def pop: A =
        _store.remove(_store.length - 1)
}
class ArrayBufferStack[A] extends Stack[A] {
  val _store = new ArrayBuffer()

  def push(element: A): Unit =
    _store.append(element)

  def top: A =
    _store.last

  def pop: A =
    _store.remove(store.length-1)
}

Amortized $O(1)$

What is the runtime?
Stacks in Scala

Scala's Stack implementation is based on ArrayBuffer; Keeping memory together is worth the overhead of amortized $O(1)$. 
Queue

Outside of the US, "queueing" is lining up, i.e., at Starbucks

**Enqueue** Put a new object at the end of the queue

**Dequeue** Remove the next object in the queue

**Head** Peek at the next object in the queue
Queue

Front

Back
enqueue("Bob")
enqueue ("Bob")
enqueue ("Mary")
Queue

enqueue("Bob")
enqueue("Mary")
enqueue("Sue")
Queue

enqueue("Bob")
enqueue("Mary")
enqueue("Sue")

decqueue()
enqueue("Bob")
enqueue("Mary")
enqueue("Sue")
dequeue()
enqueue("Steve")
Queue

tenqueue("Bob")
tenqueue("Mary")
tenqueue("Sue")
dequeue()
tenqueue("Steve")
dequeue()
Queues vs Stacks

**Queue** First in, First Out (FIFO)

**Stacks** Last in, First Out (LIFO / FILO)
Queues in Practice

- Delivering network packets, emails, twitter/tiktok/instagram
- Scheduling CPU cycles
- Deferring long-running tasks
trait Queue[A] {  def enqueue(element: A): Unit  def dequeue: A  def head: A}
class ListQueue[A] extends Queue[A] {
    val _store = new DoublyLinkedList()

    def enqueue(element: A): Unit =
        _store.append(element)

    def head: A =
        _store.head

    def dequeue: A =
        _store.remove(0)
}
class ListQueue[A] extends Queue[A] {
  val _store = new DoublyLinkedList()

  def enqueue(element: A): Unit =
    _store.append(element)

  def head: A =
    _store.head

  def dequeue: A =
    _store.remove(0)
}

What is the runtime?
class `ListQueue[A]` extends `Queue[A]` {
  val _store = new DoublyLinkedList()

  def enqueue(element: A): Unit = \( \Theta(1) \)
  _store.append(element)

  def head: A = \( \Theta(1) \)
  _store.head

  def dequeue: A = \( \Theta(1) \)
  _store.remove(0)
}

What is the runtime?
Queues

**Thought Experiment:** How can we use an array to build a queue?