CSE 250: Quicksort (contd); Stacks and Queues

Lecture 14

Sept 27, 2024

Reminders

- WA2 due Sun, Sept 29 at 11:59 PM
- Midterm 1 in class on Fri, Oct 04.
  - Covers: Asymptotics, Sequences/Lists, Arrays, Linked Lists, Recursion
  - Bounds: Tight Upper/Lower, Unqualified vs Amortized

Quicksort

Merge Sort

- Split up the problem $(O(1))$
- Merge the sub-solutions $(O(N))$

Quicksort

Merge Sort

- Split up the problem ($O(1)$)
- Merge the sub-solutions ($O(N)$)

Quicksort

- Split up the problem ($O(N)$)
- Merge the sub-solutions ($O(1)$)

Quicksort

- If the list has only 1 element, it's sorted. Return.
- Find the **median** value ('pivot').
- Create a list of elements $<$ **median**.
- Create a list of elements $>$ **median**.
- Sort each of the two lists
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot').
- Create a list of elements $<$ **median**.
- Create a list of elements $>$ **median**.
- Sort each of the two lists
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**.
- Create a list of elements $>$ **median**.
- Sort each of the two lists
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**. $O(N)$
- Create a list of elements $>$ **median**.
- Sort each of the two lists
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**. $O(N)$
- Create a list of elements $>$ **median**. $O(N)$
- Sort each of the two lists
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**. $O(N)$
- Create a list of elements $>$ **median**. $O(N)$
- Sort each of the two lists $2 \cdot T(\frac{N}{2})$
- Concatenate the lists

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**. $O(N)$
- Create a list of elements $>$ **median**. $O(N)$
- Sort each of the two lists $2 \cdot T(\frac{N}{2})$
- Concatenate the lists $O(N)$ or $O(1)$

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N)$
- Create a list of elements $<$ **median**. $O(N)$
- Create a list of elements $>$ **median**. $O(N)$
- Sort each of the two lists $2 \cdot T(\frac{N}{2})$
- Concatenate the lists $O(N)$ or $O(1)$

$$T_{qs}(N) = \begin{cases} 1 & \textbf{if } N \leq 1 \\ O(N) + 2T_{qs}(\frac{N}{2}) + T_{pivot}(N) & \textbf{otherwise} \end{cases}$$

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $O(N\log(N))$
- Create a list of elements < **median**. $O(N)$
- Create a list of elements > **median**. $O(N)$
- Sort each of the two lists $2 \cdot T(\frac{N}{2})$
- Concatenate the lists $O(N)$ or $O(1)$

$$T_{qs}(N) = \begin{cases} 1 & \textbf{if } N \leq 1 \\ O(N) + 2T_{qs}(\frac{N}{2}) + N\log(N) & \textbf{otherwise} \end{cases}$$

Quicksort

Idea: If we pick a pivot value at random,
in expectation, half of the values will be lower.

The Worst-Case Pivot

What is the worst case runtime?

The Worst-Case Pivot

What if we always pick the worst pivot?

1. `[ 8, 7, 6, 5, 4, 3, 2, 1 ]`

The Worst-Case Pivot

What if we always pick the worst pivot?

1. [8, 7, 6, 5, 4, 3, 2, 1]
2. [7, 6, 5, 4, 3, 2, 1], 8, []

The Worst-Case Pivot

What if we always pick the worst pivot?

1. [8, 7, 6, 5, 4, 3, 2, 1]
2. [7, 6, 5, 4, 3, 2, 1], 8, []
3. [6, 5, 4, 3, 2, 1], 7, [], 8, []

The Worst-Case Pivot

What if we always pick the worst pivot?

1. `[ 8, 7, 6, 5, 4, 3, 2, 1 ]`
2. `[ 7, 6, 5, 4, 3, 2, 1 ], 8, []`
3. `[ 6, 5, 4, 3, 2, 1 ], 7, [], 8, []`
4. `[ 5, 4, 3, 2, 1 ], 6, [], 7, [], 8, []`

The Worst-Case Pivot

What if we always pick the worst pivot?

1. [8, 7, 6, 5, 4, 3, 2, 1]
2. [7, 6, 5, 4, 3, 2, 1], 8, []
3. [6, 5, 4, 3, 2, 1], 7, [], 8, []
4. [5, 4, 3, 2, 1], 6, [], 7, [], 8, []

...

The Worst-Case Pivot

What if we always pick the worst pivot?

1. `[ 8, 7, 6, 5, 4, 3, 2, 1 ]`
2. `[ 7, 6, 5, 4, 3, 2, 1 ], 8, []`
3. `[ 6, 5, 4, 3, 2, 1 ], 7, [], 8, []`
4. `[ 5, 4, 3, 2, 1 ], 6, [], 7, [], 8, []`

...

- For each level, $O(N) work$
- At worst, $O(N)$ levels

Total:

The Worst-Case Pivot

What if we always pick the worst pivot?

1. [8, 7, 6, 5, 4, 3, 2, 1]
2. [7, 6, 5, 4, 3, 2, 1], 8, []
3. [6, 5, 4, 3, 2, 1], 7, [], 8, []
4. [5, 4, 3, 2, 1], 6, [], 7, [], 8, []

...

- For each level, $O(N)$*work*
- At worst, $O(N)$ levels

Total: $T_{quicksort}(N) \in O(N^2)$

Quicksort

Expected Runtime

Is the worst case runtime representative?

Expected Runtime

Is the worst case runtime representative?

No! (in typical cases, it will be faster)

Expected Runtime

Is the worst case runtime representative?

No! (in typical cases, it will be faster)

Is there something we <u>can</u> say about the runtime?

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$:

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N)$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0)$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$
- $X = 2$: $\theta(N) + T_{qs}(1) + T_{qs}(N-2)$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$
- $X = 2$: $\theta(N) + T_{qs}(1) + T_{qs}(N-2)$
- $X = 3$: $\theta(N) + T_{qs}(2) + T_{qs}(N-3)$
- $X = 4$: $\theta(N) + T_{qs}(3) + T_{qs}(N-4)$
- $\ldots$
- $X = N-1$: $\theta(N) + T_{qs}(N-2) + T_{qs}(1)$
- $X = N$: $\theta(N) + T_{qs}(N-1) + T_{qs}(0)$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$
- $X = 2$: $\theta(N) + T_{qs}(1) + T_{qs}(N-2)$
- $X = 3$: $\theta(N) + T_{qs}(2) + T_{qs}(N-3)$
- $X = 4$: $\theta(N) + T_{qs}(3) + T_{qs}(N-4)$
- ...
- $X = N-1$: $\theta(N) + T_{qs}(N-2) + T_{qs}(1)$
- $X = N$: $\theta(N) + T_{qs}(N-1) + T_{qs}(0)$

$$T_{qs}(N) = \theta(N) + T_{qs}(X-1) + T_{qs}(N-X)$$

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$
- $X = 2$: $\theta(N) + T_{qs}(1) + T_{qs}(N-2)$
- $X = 3$: $\theta(N) + T_{qs}(2) + T_{qs}(N-3)$
- $X = 4$: $\theta(N) + T_{qs}(3) + T_{qs}(N-4)$
- ...
- $X = N-1$: $\theta(N) + T_{qs}(N-2) + T_{qs}(1)$
- $X = N$: $\theta(N) + T_{qs}(N-1) + T_{qs}(0)$

$$T_{qs}(N) = \theta(N) + T_{qs}(X-1) + T_{qs}(N-X) \qquad (\text{for } X \in [1, N])$$

Expectation

If X is a variable representing a random outcome, we call this number the **expectation** of X, or $E[X]$.

Expectation

If X is a variable representing a random outcome, we call this number the **expectation** of X, or $E[X]$.

$$E[X] = \sum_i P_i \cdot X_i$$

Quicksort

$$T_{qs}(N) = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ T_{qs}(X-1) + T_{qs}(N-X) + \theta(N) & \textbf{otherwise} \end{cases}$$

Quicksort

$$T_{qs}(N) = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ T_{qs}(X-1) + T_{qs}(N-X) + \theta(N) & \textbf{otherwise} \end{cases}$$

… but X is random!

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1) + T_{qs}(N-X) + \theta(N)] & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1)] + E[T_{qs}(N-X)] + E[\theta(N)] & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1)] + E[T_{qs}(N-X)] + \theta(N) & \textbf{otherwise} \end{cases}$$

Expected Runtimes

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

Expected Runtimes

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

$$= \sum_{i=1}^{N} P_i \cdot T(X_i - 1)$$

Expected Runtimes

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

$$= \sum_{i=1}^{N} P_i \cdot T(X_i - 1)$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(i-1) \qquad (T(0) \text{ up to } T(n-1))$$

Expected Runtimes

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

$$= \sum_{i=1}^{N} P_i \cdot T(X_i - 1)$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(i-1) \qquad (T(0) \text{ up to } T(n-1))$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(n-i) \qquad (T(n-1) \text{ down to } T(0))$$

Expected Runtimes

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

$$= \sum_{i=1}^{N} P_i \cdot T(X_i - 1)$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(i-1) \qquad (T(0) \text{ up to } T(n-1))$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(n-i) \qquad (T(n-1) \text{ down to } T(0))$$

$$= E[T(N-X)]$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1)] + E[T_{qs}(N-X)] + \theta(N) & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot E[T_{qs}(X-1)] + \theta(N) & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot E[T_{qs}(X-1)] + \theta(N) & \textbf{otherwise} \end{cases}$$

The X we pick at each step is independent.

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=1}^{N} \frac{1}{N} E[T_{qs}(i-1)]\right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Back to Induction!

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Back to Induction! **Inductive Hypothesis:**
$E[T_{qs}(N)] \in O(N \log(N))$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Back to Induction! **Inductive Hypothesis:**

$E[T_{qs}(N)] \in O(N \log(N))$

$E[T_{qs}(N)] \leq c \cdot N \cdot \log(N)$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case

$$E[T_{qs}(1)] \stackrel{?}{\leq} c \cdot 1 \cdot \log(1)$$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case

$$E[T_{qs}(1)] \stackrel{?}{\leq} c \cdot 1 \cdot \log(1)$$

$$E[T_{qs}(1)] \stackrel{?}{\leq} c \cdot 1 \cdot 0$$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case

$$E[T_{qs}(1)] \overset{?}{\leq} c \cdot 1 \cdot \log(1)$$

$$E[T_{qs}(1)] \overset{?}{\leq} c \cdot 1 \cdot 0$$

$$E[T_{qs}(1)] \not\leq 0$$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case

$$E[T_{qs}(1)] \overset{?}{\leq} c \cdot 1 \cdot \log(1)$$

$$E[T_{qs}(1)] \overset{?}{\leq} c \cdot 1 \cdot 0$$

$$E[T_{qs}(1)] \not\leq 0$$ ... oops

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 N & \textbf{otherwise} \end{cases}$$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case (with $N_0 = 2$)

$$E[T_{qs}(2)] \overset{?}{\leq} c \cdot 2 \cdot \log(2)$$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case (with $N_0 = 2$)

$$E[T_{qs}(2)] \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

$$2 \cdot \left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)] \right) + 2c_2 \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case (with $N_0 = 2$)

$$E[T_{qs}(2)] \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

$$2 \cdot \left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + 2c_2 \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c$$

Quicksort's Expected Runtime

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c$$

Quicksort's Expected Runtime

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + c_2 \overset{?}{\leq} c$$

$$\frac{1}{2}(c_1 + c_1) + c_2 \overset{?}{\leq} c$$

Quicksort's Expected Runtime

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c$$

$$\frac{1}{2}\left(c_1 + c_1\right) + c_2 \stackrel{?}{\leq} c$$

$$c_1 + c_2 \leq c$$

(true if we set $c \geq c_1 + c_2$)

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 N & \textbf{otherwise} \end{cases}$$

Inductive Step

Assume: $E[T_{qs}(N')] \leq c \cdot N' \log(N')$ for all $2 \leq N' \leq N$

Show: $E[T_{qs}(N)] \leq c \cdot N \log(N)$

$$2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(\sum_{i=0}^{N-1} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$2 \cdot \frac{1}{N}\left(\sum_{i=0}^{N-1} E[T_{qs}(i)]\right) + c_2 \overset{?}{\leq} c \cdot N\log(N)$$

Quicksort's Expected Runtime

$$2 \cdot \frac{1}{N}\left(\sum_{i=0}^{N-1} E[T_{qs}(i)]\right) + c_2 \overset{?}{\leq} c \cdot N\log(N)$$

$$2 \cdot \frac{1}{N}\left(E[T_{qs}(0)] + E[T_{qs}(1)] + \sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + c_2 \overset{?}{\leq} c \cdot N\log(N)$$

Quicksort's Expected Runtime

$$2 \cdot \frac{1}{N} \left(\sum_{i=0}^{N-1} E[T_{qs}(i)] \right) + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(E[T_{qs}(0)] + E[T_{qs}(1)] + \sum_{i=2}^{N-1} E[T_{qs}(i)] \right) + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(2 \cdot c_1 + \sum_{i=2}^{N-1} E[T_{qs}(i)] \right) + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$2 \cdot \frac{1}{N}\left(\sum_{i=0}^{N-1} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c \cdot N\log(N)$$

$$2 \cdot \frac{1}{N}\left(E[T_{qs}(0)] + E[T_{qs}(1)] + \sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c \cdot N\log(N)$$

$$2 \cdot \frac{1}{N}\left(2 \cdot c_1 + \sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + c_2 \stackrel{?}{\leq} c \cdot N\log(N)$$

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N\log(N)$$

Quicksort's Expected Runtime

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} c \cdot i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} c \cdot i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$\frac{2c}{N}\left(\sum_{i=2}^{N-1} i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} c \cdot i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

$$\frac{2c}{N}\left(\sum_{i=2}^{N-1} i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

The following left-hand-side is strictly bigger than the preceding.

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i \log(N)\right) + \frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N\log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2\cdot c_1}{N} + c_2 \stackrel{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\sum_{i=1}^{N-1} i\right) + \frac{2\cdot c_1}{N} + c_2 \stackrel{?}{\leq} c\cdot N\log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\sum_{i=1}^{N-1} i\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\frac{(N-1)\cdot N}{2}\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\sum_{i=1}^{N-1} i\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\frac{(N-1)\cdot N}{2}\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$c\cdot N\log(N) - c\cdot\log(N) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\sum_{i=1}^{N-1} i\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\frac{(N-1)\cdot N}{2}\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$c\cdot N\log(N) - c\cdot\log(N) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot\log(N)$$

Quicksort's Expected Runtime

$$\frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot \log(N)$$

Quicksort's Expected Runtime

$$\frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot \log(N)$$

The following left-hand-side is strictly bigger than the preceding.

$$2 \cdot c_1 + c_2 \stackrel{?}{\leq} c \cdot \log(N)$$

True for any $N \geq 2$ if $c \geq 2 \cdot c_1 + c_2$

Quicksort's Expected Runtime

$$\frac{2 \cdot c_1}{N} + c_2 \overset{?}{\leq} c \cdot \log(N)$$

The following left-hand-side is strictly bigger than the preceding.

$$2 \cdot c_1 + c_2 \overset{?}{\leq} c \cdot \log(N)$$

True for any $N \geq 2$ if $c \geq 2 \cdot c_1 + c_2$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] \in O(N \log(N))$$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] \in O(N\log(N))$$

So is Quicksort $O(N\log(N))$?

Quicksort's Expected Runtime

$$E[T_{qs}(N)] \in O(N \log(N))$$

So is Quicksort $O(N \log(N))$? **No!**

Quicksort's Expected Runtime

$$E[T_{qs}(N)] \in O(N \log(N))$$

So is Quicksort $O(N \log(N))$? **No!**

Quicksort's **Expected** runtime is $O(N \log(N))$

Bound Guarantees

- $f(N)$ is a Worst-Case Bound $(T(N) \in O(f(N)))$
  The algorithm **always** runs in at most $c \cdot f(N)$ steps.
- $f(N)$ is an Amortized Worst-Case Bound
  N **invocations** of the algorithm **always** run in at most $N \cdot c \cdot f(N)$ steps.
- $f(N)$ is an Expected Worst-Case Bound $(E[T(N)] \in O(f(N)))$
  The algorithm is **statistically likely** to run in at most $c \cdot f(N)$ steps.

Back to Sequence ADTs

- **Sequence**
  - `get(i)`, `set(i, v)`
- **List**
  - ... and `add(v)`, `add(i, v)`, `remove(i)`,

Back to Sequence ADTs

- **Sequence**
  - `get(i)`, `set(i, v)`
- **List**
  - ... and `add(v)`, `add(i, v)`, `remove(i)`,
- **Stack**
  - `push(v)`, `pop()`, `peek()`
- **Queue**
  - `add(v)`, `remove()`, `peek()`

The Stack ADT

A stack of objects on top of one another.

- **Push**
  Put a new object on top of the stack.
- **Pop**
  Remove the object from the top of the stack.
- **Top**
  Peek at what's on top of the stack.

Stacks

Demo

Stacks in Practice

- Storing method-local variables ("call stack")
- Certain types of parsers ("context free")
- Backtracking search (more on Friday)
- Reversing sequences

The Stack Interface

```
public interface Stack<E> extends List<E> {
  public boolean empty();
  public E peek();
  public E pop();
  public E push(E item);
  /* ... */
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { /* ??? */ }

  public E peek() { /* ??? */ }

  public E pop(E item) { /* ??? */ }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(0, item); }

  public E peek() { /* ??? */ }

  public E pop(E item) { /* ??? */ }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(0, item); }

  public E peek() { return data.get(0); }

  public E pop(E item) { /* ??? */ }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(0, item); }

  public E peek() { return data.get(0); }

  public E pop(E item) { return data.remove(0); }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; } O(1)

  public E push(E item) { return data.add(0, item); }

  public E peek() { return data.get(0); }

  public E pop(E item) { return data.remove(0); }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(0, item); }

  public E peek() { return data.get(0); }

  public E pop(E item) { return data.remove(0); }
}
```

$O(1)$ (empty)

$O(1)$ (push)

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; }    O(1)

  public E push(E item) { return data.add(0, item); }    O(1)

  public E peek() { return data.get(0); }          O(i) = O(1)

  public E pop(E item) { return data.remove(0); }
}
```

LinkedListStack

```
class LinkedListStack<E> implements Stack<E> {
  LinkedList<E> data = new LinkedList();

  public boolean empty() { return data.size() == 0; } O(1)

  public E push(E item) { return data.add(0, item); } O(1)

  public E peek() { return data.get(0); }       O(i) = O(1)

  public E pop(E item) { return data.remove(0); }     O(1)
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { /* ??? */ }

  public E peek() { /* ??? */ }

  public E pop(E item) { /* ??? */ }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(item); }

  public E peek() { /* ??? */ }

  public E pop(E item) { /* ??? */ }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(item); }

  public E peek() { return data.get(data.size()-1); }

  public E pop(E item) { /* ??? */ }
}
```

ArrayListStack

```java
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }

  public E push(E item) { return data.add(item); }

  public E peek() { return data.get(data.size()-1); }

  public E pop(E item) {
      return data.remove(data.size()-1); }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }   O(1)

  public E push(E item) { return data.add(item); }

  public E peek() { return data.get(data.size()-1); }

  public E pop(E item) {
      return data.remove(data.size()-1); }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }   O(1)

  public E push(E item) { return data.add(item); }   am.O(1)

  public E peek() { return data.get(data.size()-1); }

  public E pop(E item) {
      return data.remove(data.size()-1); }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }  O(1)

  public E push(E item) { return data.add(item); }  am. O(1)

  public E peek() { return data.get(data.size()-1); }  O(1)

  public E pop(E item) {
      return data.remove(data.size()-1); }
}
```

ArrayListStack

```
class ArrayListStack<E> implements Stack<E> {
  ArrayList<E> data = new ArrayList();

  public boolean empty() { return data.size() == 0; }   O(1)

  public E push(E item) { return data.add(item); }   am. O(1)

  public E peek() { return data.get(data.size()-1); }   O(1)

  public E pop(E item) {
      return data.remove(data.size()-1); }   O(N − i) = O(1)
}
```

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)`

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

What other assumptions make this ok?

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

What other assumptions make this ok?

Stacks usually have a maximum capacity.

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

What other assumptions make this ok?

Stacks usually have a maximum capacity.

- $2\times$ capacity is wasteful when you have an upper limit.

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

What other assumptions make this ok?

Stacks usually have a maximum capacity.

- $2\times$ capacity is wasteful when you have an upper limit.
- Each call to **pop** gives you another **push** credit.

Stacks in Java

Java's `Stack` is implemented using a `Vector`.
(Like an `ArrayList`, but grows by adding a constant to its capacity)

What does this mean for the runtime of `push(item)` $O(N)$

What other assumptions make this ok?

Stacks usually have a maximum capacity.

- $2\times$ capacity is wasteful when you have an upper limit.
- Each call to **pop** gives you another **push** credit.
- Keeping elements together in memory is worth the overhead.