QuickSort

- Input: Array A of unique integers
- Output: Sorted A

1. If |A| = 1: Return A
2. \( A[\text{pivot}] \) - chosen by a method
3. \( \text{QuickSort}(A[0...\text{pivot}]) \)
4. \( \text{QuickSort}(A[\text{pivot}...\text{last}]) \)

**Effect of Partition:**
- \( A_{\text{left}} \) and \( A_{\text{right}} \)
- Initial: \( \text{unsorted} \)
- After partitioning: \( \text{sorted} \)

- \( \text{Partition} \) (A, pivot)

**Key Points**
- Partition is doing most of the work in QuickSort
- Runtime of partition scales like the # of comparisons

**Idea:** To determine runtime of QuickSort, count comparisons over the whole array

Q: How many comparisons are done by partition on an array of size \( n \)?

A: B) \( O(n) \)

**Effect of Partition:**
- \( A_{\text{left}} \) vs. \( A_{\text{right}} \)
- \( n+1 \)

1. Suppose you get very lucky and pivot is always chosen to be median of A. Every time partition is called:
   - Create recurrence relation for runtime of QuickSort
   - Solve
2. Suppose you got very unlucky and pivot is always chosen to be minimum of A. Every time partition is called:
   - Create recurrence relation for runtime of QuickSort
   - Solve

1. \( T(n) = O(1), n = 1 \)
2. \( T(n) = 2T(\frac{n}{2}) + O(n) \)

\( T(n) = 2O(n\log n) = O(n\log n) \)

**Lucky vs. Unlucky**

\( O(n\log n) \) vs. \( O(n^2) \)

**Which is likely?**
**Which is average?**