QuickSort

To sort $A[\text{left}, \text{right}]$:

**Divide:** Partition $A[\text{left}, \text{right}]$ into two parts:

$$A[\text{left}, \text{p}], \ A[\text{p+1}, \text{right}]$$

where $x \leq y$ for all $x \in A[\text{left}, \text{p}]$ and $y \in A[\text{p+1}, \text{right}]$

**Conquer:**

- Quick-Sort($A[\text{left}, \text{p}]$)
- Quick-Sort($A[\text{p+1}, \text{right}]$)

**Combine:** Automatic (no action needed)
Quick-Sort

Quick-Sort(A,left,right)
1 if left < right
2 \[ p \leftarrow \text{Partition}(A,\text{left},\text{right}) \]
3 \[ \text{Quick-Sort}(A,\text{left},p) \]
4 \[ \text{Quick-Sort}(A,p+1,\text{right}) \]

To sort \( A[1..n] \): \[ \text{Quick-Sort}(A,1,n) \]
Partition

- Choose an element of $A$: $e \leftarrow A[left]$
- Scan from right until $A[R] \leq e$ is found
- Scan from left until $A[L] \geq e$ is found
- If $L < R$ then exchange $A[L] \leftrightarrow A[R]$
- Continue scanning and exchanging until $R \leq L$
Partition

Partition(A, left, right)

1 \( e \leftarrow A[left] \)
2 \( L \leftarrow left-1 \)
3 \( R \leftarrow right+1 \)
4 while TRUE
5 \( \) repeat \( R \leftarrow R-1 \) until \( A[R] \leq e \)
6 \( \) repeat \( L \leftarrow L+1 \) until \( A[L] \geq e \)
7 \( \) if \( L < R \) then
8 \( \) \( A[L] \leftrightarrow A[R] \)
9 \( \) else
10 \( \) return \( R \)

Running Time:
\[ T(n) = \Theta(n) \]
for \( n = r-l+1 \)
Quick-Sort: Running Time

\[ T(n) = \Theta(n) + T(p - \text{left}) + T(\text{right} - p) \]

\[ n = \text{right} - \text{left}. \]

\( T(n) \) depends on the position of \( p \) in the range \([\text{left}, \ldots, \text{right}]\)
**Worst-Case**

Running Time:

\[ T(n) = T(n - 1) + T(1) + \Theta(n) = \]
\[ = T(n - 1) + \Theta(n) = \]
\[ \downarrow \]
\[ = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2) \]

\[ \Theta(n^2) \]
Worst-Case Running Time

\[ T(n) = \max_{1 \leq p \leq n-1} \{ T(p) + T(n - p) + \Theta(n) \} = \]

\[ = \max_{1 \leq p \leq n-1} \{ T(p) + T(n - p) \} + \Theta(n) \]

We "guess" \( T(n) \leq cn^2 \) and verify inductively

\[ T(n) \leq \max_{1 \leq p \leq n-1} \{ cp^2 + c(n - p)^2 \} + \Theta(n) = \]

\[ = c \cdot \max_{1 \leq p \leq n-1} \{ p^2 + (n - p)^2 \} + \Theta(n) = \]

the maximum
\[ = c \cdot (1^2 + (n - 1)^2 ) + \Theta(n) = \]

is achieved at
\[ = \Theta(n^2) \]

one of the ends
Quick-Sort

Best-Case Running Time

\[
T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n)
\]
Quick-Sort

Best-Case Running Time

\[
T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n)
\]
Balanced Partitioning

\[ T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + n = \Theta(n \log n) \]
Average Case: Intuition

\[
\begin{align*}
\text{n} & \quad \text{l} \quad \text{n-1} \\
\text{Θ(n)} \\
(\text{n - 1})/2 & \quad (\text{n - 1})/2 \\
\text{Θ(n)} \\
(\text{n - 1})/2 + 1 & \quad (\text{n - 1})/2
\end{align*}
\]
Average-Case Running Time

\[ T(n) = n + T(p - 1) + T(n - p) \]

\[ T_{\text{avg}}(n) = \sum_{k=1}^{n} \Pr[p = k] \cdot (n + T_{\text{avg}}(k - 1) + T_{\text{avg}}(n - k)) \]

The partitioning element has probability \( \frac{1}{n} \) to go into each of \( n \) final positions

\[ T_{\text{avg}}(n) = n + \frac{1}{n} \cdot \sum_{k=1}^{n} \left( T_{\text{avg}}(k - 1) + T_{\text{avg}}(n - k) \right) \]

Partition probability recursion
Average-Case Running Time

\[ T_{\text{avg}}(n) = n + \frac{2}{n} \cdot \sum_{k=1}^{n} T_{\text{avg}}(k - 1) \]

Multiply both sides by \( n \)

\[ n \cdot T_{\text{avg}}(n) = n^2 + 2 \cdot \sum_{k=1}^{n} T_{\text{avg}}(k - 1) \]

Substitute \( n-1 \) for \( n \)

\[ (n - 1) \cdot T_{\text{avg}}(n - 1) = (n - 1)^2 + 2 \cdot \sum_{k=1}^{n-1} T_{\text{avg}}(k - 1) \]

Subtract

\[ nT_{\text{avg}}(n) - (n - 1)T_{\text{avg}}(n - 1) = n^2 - (n - 1)^2 + 2T(n - 1) \]
Average-Case Running Time

\[ nT_{avg}(n) - (n - 1)T_{avg}(n - 1) = n^2 - (n - 1)^2 + 2T(n - 1) \]

Simplify: \[ nT_{avg}(n) = (n + 1)T_{avg}(n - 1) + 2n - 1 \]

Drop the \(-1\) to turn = into \(\leq\) and

Divide by \(n(n+1):\)

\[ \frac{T_{avg}(n)}{n + 1} \leq \frac{T_{avg}(n - 1)}{n} + \frac{2}{n + 1} \]
Average-Case Running Time

\[
\frac{T_{avg}(n)}{n + 1} \leq \frac{T_{avg}(n - 1)}{n} + \frac{2}{n + 1}
\]

Define: \[S(n) = \frac{T_{avg}(n)}{n + 1}\]

Substitute in the above formula to get:

\[
S(n) \leq S(n - 1) + \frac{2}{n}
\]
Average-Case Running Time

\[ S(n) \leq S(n - 1) + \frac{2}{n} \]

Expand, to get:

\[ S(n) \leq S(n - 2) + \frac{2}{n - 1} + \frac{2}{n} \]

\[ \leq 2 \left( 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n - 1} + \frac{1}{n} \right) \]

But,

\[ S(n) \leq 2 \sum_{k=1}^{n} \frac{1}{k} = 2H_n = \Theta(\log n) \]
Average-Case Running Time

\[ S(n) = O(\log n) \]

But \[ S(n) = \frac{T_{avg}(n)}{n + 1} \]

So that \[ T_{avg}(n) = O(n \log n) \]
Randomization

• Randomly permute the input (before sorting)
  – tree of all possible executions
    most of them finish fast

• Choose partitioning element $e$ randomly
  at each iteration
  – easier to analyze
  – same "good" behavior
Randomized-Partition

Rand-Partition (A, left, right)

1. q ← Random(left, right)
3. return Partition(A, left, right)

Running Time: \( T(n) = \Theta(n) \)
Randomized Quick-Sort

Rand-Quick-Sort(A, left, right)
1    if  left < right
2    p ← Rand-Partition(A, left, right)
3    Rand-Quick-Sort(A, left, p)
4    Rand-Quick-Sort(A, p+1, right)

Expected Running Time: \( T(n) = O(n \log n) \)
Variants of Quick-Sort

• When we get to small sub-arrays, do not call Quick-Sort recursively
  – Run Insertion-Sort on the sub-arrays
  – Go back up recursion and run Insertion-Sort on the nearly sorted array

• Different ways to pick the pivot element:
  – Always pick the first element
  – Pick a random element
  – Pick the median of some elements
Using Medians of 3

Median-of-3-Partition(A, left, right)

1    if left < right
2    sort (A[left],
        A[(left+right)/2],
        A[right])
3    A[left] ↔ A[(left+right)/2]
4    return Partition(A, left, right)

Expected Running Time: \( T(n) = \Theta(n) \)
Randomized Medians of 3

\textbf{Rand-Median-of-3-Partition} (A, left, right)

1. if left < right
2. Randomly choose $j \leq k \leq l \leq \text{right}$
4. $A[left] \leftrightarrow A[k]$
5. return $\text{Partition}(A, left, right)$

\textbf{Expected Running Time:} $T(n) = \Theta(n)$
Notes on Quick-Sort

• Fast on average - $\Theta(n \log n)$

• Worst-case $\Theta(n^2)$ but unlikely with medians-of-3

• Good cache performance

• In place sorting
  (does use extra storage, for recursion stack)

• Not a stable sort