CSE 373: Analysis of Algorithms

Lectures 11, 12 & 13
(Quicksort and Average Case Analysis)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2014
The Divide-and-Conquer Process in Merge Sort

Suppose we want to sort a typical subarray $A[p..r]$.

**DIVIDE:** Split $A[p..r]$ at midpoint $q$ into two subarrays $A[p..q]$ and $A[q+1..r]$ of equal or almost equal length.

**CONQUER:** Recursively sort $A[p..q]$ and $A[q+1..r]$.

**COMBINE:** Merge the two sorted subarrays $A[p..q]$ and $A[q+1..r]$ to obtain a longer sorted subarray $A[p..r]$.

The **DIVIDE** step is cheap — takes only $\Theta(1)$ time.

But the **COMBINE** step is costly — takes $\Theta(n)$ time, where $n$ is the length of $A[p..r]$. 
The Divide-and-Conquer Process in Quicksort

Suppose we want to sort a typical subarray $A[p..r]$.

**Divide:** Partition $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ and find index $q$ such that

- each element of $A[p..q-1]$ is $\leq A[q]$, and
- each element of $A[q+1..r]$ is $\geq A[q]$.

**Conquer:** Recursively sort $A[p..q-1]$ and $A[q+1..r]$.

**Combine:** Since $A[q]$ is larger and smaller than everything to its left and right, respectively, and both left and right parts are sorted, subarray $A[p..r]$ is also sorted.

The Combine step is cheap — takes only $\Theta(1)$ time.

But the Divide step is costly — takes $\Theta(n)$ time, where $n$ is the length of $A[p..r]$. 
**Quicksort**

**Input:** A subarray $A[ p : r ]$ of $r - p + 1$ numbers, where $p \leq r$.


**QUICKSORT** ($A, p, r$)

1. **if** $p < r$ **then**
3. $q = \text{PARTITION} (A, p, r)$
4. // recursively sort the left part
5. **QUICKSORT** ($A, p, q - 1$)
6. // recursively sort the right part
7. **QUICKSORT** ($A, q + 1, r$)
Partition

**Input:** A subarray $A[ p : r ]$ of $r - p + 1$ numbers, where $p \leq r$.

**Output:** Elements of $A[ p : r ]$ are rearranged such that for some $q \in [p, r]$ everything in $A[ p : q - 1 ]$ is $\leq A[q]$ and everything in $A[ q + 1: r ]$ is $\geq A[q]$. Index $q$ is returned.

```
PARTITION ( A, p, r )

1. x = A[r]
2. i = p - 1
3. for j = p to r - 1
4.     if A[j] $\leq$ x
5.         i = i + 1
7.     exchange A[i + 1] with A[r]
8.     return i + 1
```
Correctness of Partition

**Input:** A subarray $A[ p : r ]$ of $r - p + 1$ numbers, where $p \leq r$.

**Output:** Elements of $A[ p : r ]$ are rearranged such that for some $q \in [p, r]$ everything in $A[ p : q - 1 ]$ is $\leq A[q]$ and everything in $A[ q + 1 : r ]$ is $\geq A[q]$. Index $q$ is returned.

### Partition ($A, p, r)$

1. $x = A[r]$
2. $i = p - 1$
3. for $j = p$ to $r - 1$
4. if $A[j] \leq x$
5. $i = i + 1$
7. exchange $A[i + 1]$ with $A[r]$
8. return $i + 1$

**Loop Invariant**

At the start of each iteration of the for loop of lines 3–6, for any array index $k$,

1. if $p \leq k \leq i$,
   then $A[k] \leq x$.
2. if $i + 1 \leq k \leq j - 1$,
   then $A[k] > x$.
3. if $k = r$,
   then $A[k] = x$. 
Running Time of Partition

**Input:** A subarray $A[p : r]$ of $r - p + 1$ numbers, where $p \leq r$.

**Output:** Elements of $A[p : r]$ are rearranged such that for some $q \in [p, r]$ everything in $A[p : q - 1]$ is $\leq A[q]$ and everything in $A[q + 1 : r]$ is $\geq A[q]$. Index $q$ is returned.

---

**Partition (A, p, r)**

1. $x = A[r]$
2. $i = p - 1$
3. \textbf{for} $j = p \text{ to } r - 1$
4. \hspace{1em} \textbf{if} $A[j] \leq x$
5. \hspace{2em} $i = i + 1$
6. \hspace{1em} exchange $A[i]$ with $A[j]$
7. exchange $A[i + 1]$ with $A[r]$
8. \textbf{return} $i + 1$

---

Let $n = r - p + 1$.

The loop of lines 3–6 takes $\Theta(r - 1 - p + 1) = \Theta(n)$ time.

Lines 1, 2, 7 and 8 take $\Theta(1)$ time each.

Hence, the overall running time is $\Theta(n)$.
Worst-case Running Time of Quicksort

Quicksort (A, p, r)

1. if p < r then
3. q = Partition (A, p, r)
4. // recursively sort the left part
5. Quicksort (A, p, q - 1)
6. // recursively sort the right part
7. Quicksort (A, q + 1, r)

Assuming n = r − p + 1, the worst-case running time of quicksort:

\[ T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ \max_{p \leq q \leq r} \{T(q - p) + T(r - q)\} + \Theta(n) & \text{if } n > 1. \end{cases} \]

Replacing q with k + p − 1, we get:

\[ T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ \max_{1 \leq k \leq n} \{T(k - 1) + T(n - k)\} + \Theta(n) & \text{if } n > 1. \end{cases} \]
Worst-case Running Time of Quicksort (Upper Bound)

For $n > 1$ and a constant $c > 0$,

$$T(n) = \max_{1 \leq k \leq n} \{T(k - 1) + T(n - k)\} + cn$$

Our guess for upper bound: $T(n) \leq c_1 n^2$ for constant $c_1 > 0$.

Using this bound on the right side of the recurrence equation, we get.

$$T(n) \leq \max_{1 \leq k \leq n} \{c_1(k - 1)^2 + c_1(n - k)^2\} + cn$$

$$\Rightarrow T(n) \leq c_1 \max_{1 \leq k \leq n} \{(k - 1)^2 + (n - k)^2\} + cn$$

But $(k - 1)^2 + (n - k)^2$ reaches its maximum value for $k = 1$ and $k = n$. Hence,

$$T(n) \leq c_1((1 - 1)^2 + (n - 1)^2) + cn$$

$$\Rightarrow T(n) \leq c_1(n - 1)^2 + cn$$

$$\Rightarrow T(n) \leq c_1 n^2 - (c_1(2n - 1) - cn)$$
Worst-case Running Time of Quicksort (Upper Bound)

But for $c_1 \geq c$, we have,

$$c_1 (2n - 1) \geq c (2n - 1)$$

$$\Rightarrow c_1 (2n - 1) \geq 2cn - c$$

$$\Rightarrow c_1 (2n - 1) - cn \geq cn - c$$

But $n \geq 1 \Rightarrow cn \geq c \Rightarrow cn - c \geq 0$, and thus

$$c_1 (2n - 1) - cn \geq 0$$

$$\Rightarrow -(c_1 (2n - 1) - cn) \leq 0$$

$$\Rightarrow c_1 n^2 - (c_1 (2n - 1) - cn) \leq c_1 n^2$$

But $T(n) \leq c_1 n^2 - (c_1 (2n - 1) - cn)$. 

Hence, $T(n) \leq c_1 n^2$ for $c_1 \geq c$. 
Worst-case Running Time of Quicksort (Lower Bound)

For $n > 1$ and a constant $c > 0$, 
\[ T(n) = \max_{1 \leq k \leq n} \{ T(k - 1) + T(n - k) \} + cn \]

Our guess for lower bound: $T(n) \geq c_2 n^2$ for constant $c_2 > 0$.
Using this bound on the right side of the recurrence equation, we get.
\[ T(n) \geq \max_{1 \leq k \leq n} \{ c_2 (k - 1)^2 + c_1 (n - k)^2 \} + cn \]
\[ \Rightarrow T(n) \geq c_2 \max_{1 \leq k \leq n} \{(k - 1)^2 + (n - k)^2\} + cn \]

But $(k - 1)^2 + (n - k)^2$ reaches its maximum value for $k = 1$ and $k = n$. Hence,
\[ T(n) \geq c_2 ((1 - 1)^2 + (n - 1)^2) + cn \]
\[ \Rightarrow T(n) \geq c_2 (n - 1)^2 + cn \]
\[ \Rightarrow T(n) \geq c_2 n^2 + (cn - c_2 (2n - 1)) \]
Worst-case Running Time of Quicksort (Lower Bound)

But for $c_2 \leq \frac{c}{2}$, we have,

$$c_2(2n - 1) \leq \frac{c}{2}(2n - 1)$$

$$\Rightarrow c_2(2n - 1) \leq cn - \frac{c}{2}$$

$$\Rightarrow cn - c_2(2n - 1) \geq \frac{c}{2}$$

But $c > 0$, and thus

$$cn - c_2(2n - 1) > 0$$

$$\Rightarrow c_2n^2 + (cn - c_2(2n - 1)) > c_2n^2$$

But $T(n) \geq c_2n^2 + (cn - c_2(2n - 1))$.

Hence, $T(n) \geq c_2n^2$ for $c_2 \leq \frac{c}{2}$. 
Worst-case Running Time of Quicksort (Tight Bound)

We have proved that

\[ T(n) \leq c_1 n^2 \text{ for } c_1 \geq c, \]

and \( T(n) \geq c_2 n^2 \text{ for } c_2 \leq \frac{c}{2}. \)

Thus \( c_2 n^2 \leq T(n) \leq c_1 n^2 \) for constants \( c_1 \geq c \) and \( c_2 \leq \frac{c}{2}. \)

Hence, \( T(n) = \Theta(n^2). \)
Average Case Running Time of Quicksort

\[ T(n) = \begin{cases} \frac{1}{n} \sum_{1 \leq k \leq n} \{T(k - 1) + T(n - k)\} + \Theta(n) & \text{if } n > 1, \\ \Theta(1) & \text{if } n = 1, \end{cases} \]

**Quicksort** \((A, p, r)\)

1. \textbf{if } \text{if } p < r \textbf{ then}
2. // partition \(A[p..r]\) into \(A[p..q - 1]\)
   \[\text{and } A[q + 1..r]\text{ such that everything in } A[p..q - 1] \leq A[q] \text{ and everything in } A[q + 1..r] \geq A[q]\]
3. \(q = \text{PARTITION}(A, p, r)\)
4. // recursively sort the left part
5. \(\text{QUICKSORT}(A, p, q - 1)\)
6. // recursively sort the right part
7. \(\text{QUICKSORT}(A, q + 1, r)\)
Average Case Running Time of Quicksort

For $n > 1$ and a constant $c > 0$,

$$T(n) = \frac{1}{n} \sum_{1 \leq k \leq n} \{T(k - 1) + T(n - k)\} + cn$$

$$\Rightarrow nT(n) = \sum_{1 \leq k \leq n} \{T(k - 1) + T(n - k)\} + cn^2$$

$$\Rightarrow nT(n) = 2 \sum_{0 \leq k \leq n-1} T(k) + cn^2 \quad \cdots (1)$$

Replacing $n$ with $n - 1$,

$$\Rightarrow (n - 1)T(n - 1) = 2 \sum_{0 \leq k \leq n-2} T(k) + c(n - 1)^2 \quad \cdots (2)$$

Subtracting equation (2) from equation (1), we get

$$nT(n) - (n - 1)T(n - 1) = 2T(n - 1) + c(2n - 1)$$

$$\Rightarrow nT(n) - (n + 1)T(n - 1) = c(2n - 1)$$

Dividing both sides by $n(n + 1)$, we get

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{c(2n-1)}{n(n+1)}$$
Average Case Running Time of Quicksort

Assuming \( \frac{T(n)}{n+1} = A(n) \), we get from the equation above,

\[
A(n) - A(n - 1) = \frac{c(2n-1)}{n(n+1)}
\]

\[
\Rightarrow A(n) = A(n - 1) + \frac{c(2n-1)}{n(n+1)}
\]

\[
\Rightarrow A(n) = A(n - 1) + \frac{2c}{n+1} - \frac{c}{n(n+1)}
\]

\[
\Rightarrow A(n) < A(n - 1) + \frac{2c}{n+1}
\]

\[
\Rightarrow A(n) < A(n - 2) + \frac{2c}{n} + \frac{2c}{n+1}
\]

\[
\Rightarrow A(n) < A(n - 3) + \frac{2c}{n-1} + \frac{2c}{n} + \frac{2c}{n+1}
\]

\[
\Rightarrow A(n) < A(n - k) + \frac{2c}{n-k+2} + \frac{2c}{n-k+3} + \cdots + \frac{2c}{n} + \frac{2c}{n+1}
\]

\[
\Rightarrow A(n) < A(1) + \frac{2c}{3} + \frac{2c}{4} + \cdots + \frac{2c}{n} + \frac{2c}{n+1}
\]
Average Case Running Time of Quicksort

Since \( A(1) = \frac{T(1)}{2} = \Theta(1) \), we get,

\[
\Rightarrow A(n) < \Theta(1) + 2c \left( \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} + \frac{1}{n+1} \right)
\]

\[
\Rightarrow A(n) < \Theta(1) + 2c \left( 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} + \frac{1}{n+1} \right) - 2c \left( 1 + \frac{1}{2} \right)
\]

But \( H_{n+1} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} + \frac{1}{n+1} \) is the \( n+1 \)'st Harmonic Number, and \( \lim_{n \to \infty} H_{n+1} = \ln(n + 1) + \gamma \), where \( \gamma \approx 0.5772 \) is known as the Euler-Mascheroni constant.

Hence, for \( n \to \infty \): \( A(n) < 2c (\ln(n + 1) + \gamma) - 3c + \Theta(1) \)

\[
\Rightarrow A(n) < 2c \ln(n + 1) + \Theta(1)
\]

\[
\Rightarrow \frac{T(n)}{n + 1} < 2c \ln(n + 1) + \Theta(1)
\]

\[
\Rightarrow T(n) < 2c (n + 1) \ln(n + 1) + \Theta(n)
\]

\[
\Rightarrow T(n) = O(n \log n)
\]