**Quicksort**

*Ref:* Chapter 7 of text.

**Quicksort:**
- An in-place sorting algorithm.
- Invented by Tony Hoare in 1962.
- Worst-case running time: $\Theta(n^2)$.
- Expected running time: $O(n \log n)$.
- Fastest sorting algorithm in practice.
- Based on divide-and-conquer. (Much of the work is done in the divide step.)

**Outline:** Assume that subarray $A[p..r]$ needs to be sorted into increasing order.

(a) **Divide step:** Partition the array into two subarrays $A_1 = A[p..q-1]$ and $A_2 = A[q+1..r]$ so that each element in $A_1$ is $\leq A[q]$ and $A[q]$ is $\leq$ each element in $A_2$. (Index $q$ is computed as part of this step.)

(b) **Conquer step:** Sort the subarrays $A_1$ and $A_2$ recursively. (Recursion stops when each subarray has size 1.)

(c) **Combine step:** Nothing needs to be done.

**Pseudocode for Quicksort:** Handout 8.1.

**Conceptual view of Partition:**

- $A[p..r]$ is the array to be partitioned.
- $x$ is the pivot element.
- $p$ and $r$ are the start and end indices of the subarray.
- $q$ is the index of the pivot.

Partitioning process:
1. Place the pivot $x$ at the correct position $q$.
2. Move all elements less than $x$ to the left of $q$ and all elements greater than $x$ to the right of $q$.
3. Recursively apply the partitioning process to subarrays $A[p..q-1]$ and $A[q+1..r]$. 

Diagram:
- $A[p..r]$ is the array.
- $x$ is the pivot element.
- $p$ and $r$ are the start and end indices of the subarray.
- $q$ is the index of the pivot.
- Elements $\leq x$ are to the left of $q$.
- Elements $> x$ are to the right of $q$.
Intuitive explanation:

- Initially,
  - The “Keys ≤ x” region is empty.
  - The “Keys > x” region is empty.
  - The “Unrestricted” region extends from index $p$ to index $r - 1$.
  - The pivot element is at index $r$.
- Each iteration of the loop shrinks the “Unrestricted” region by one element.
- At the end of the loop, the “Unrestricted” region is empty.

Pseudocode for Partition: Handout 8.1.

Example for Partition: To be presented in class.

Loop Invariant for Partition: At the beginning of each iteration of the loop, for any 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 $A[k] = x$.

Correctness of Loop Invariant: To be presented in class.

Running time of Partition:
- Algorithm spends $O(1)$ time at each element of subarray $A[p..r]$. (The element is either skipped or exchanged.)
- So, running time is $O(n)$ where $n = r - p + 1$ is the size of the subarray; that is, the running time is linear in the size of the subarray.
**Worst-case analysis:**

(a) **Lower bound on running time:** \( \Omega(n^2) \).

Consider the input \( A[1..n] \) with \( A[i] = a_i \) such that 
\[ a_1 < a_2 < \ldots < a_n. \]
That is, input is already in sorted order.

- Let \( L(n) \) denote the running time for the above input.
- First call to **Partition** spends \( d \) time (for some constant \( d > 0 \)) and returns \( q = n \) with no change to the array.
- Time on the subarray \( A[1..n-1] \) is \( L(n-1) \).

Thus, 
\[
L(n) \geq L(n-1) + d \]
with \( L(1) \) being a positive constant. It is easy to see that \( L(n) = \Omega(n^2) \).

(b) **Upper bound:** \( O(n^2) \).

**Proof idea:**

- \( T(n) \): Running time on array of size \( n \).
- **Partition** time \( \leq c_1 n \).
- Let the resulting two subarrays have sizes \( q \) and \( n - q - 1 \). Then:
\[
T(n) \leq \max_{0 \leq q \leq n-1} \{ T(q) + T(n-q-1) \} + c_1 n
\]
with \( T(1) = c_2 \) for some constant \( c_2 > 0 \).

**Solution:** \( T(n) = O(n^2) \). (Details in class.)

**Intuition for average case:**

- Worst-case requires a “bad” split each time.
- If each split has \( \alpha n \) elements on the low side and \( (1 - \alpha) n - 1 \) elements on the high side for some \( 0 < \alpha < 1 \), then running time would be \( O(n \log n) \).
- In practice, there is a mix of bad and acceptable splits.
**A randomized version of Quicksort:**

- **Recall:** Function `Random(x, y)` returns an integer `i` in `[x .. y]` under uniform distribution.
- **Basic idea:** Before calling `Partition`, swap `A[r]` with `A[i]`, where `i` is the value returned by `Random(p, r)`.

**Pseudocode:** Handout 8.1.

**Observations:**

- Running time of `Quicksort` is dominated by the total time spent in all the calls to `Partition`.
- A pivot element used in some call to `Partition` is never looked at in any other call to `Partition`.
  
  (a) The total number of calls to `Partition` is at most `n`.
  
  (b) Since each call to `Partition` uses `O(n)` time, the worst-case time of `Quicksort` = `O(n^2)`.

- Each call to `Partition` uses `O(1)` time to carry out initialization and final wrap-up; the other part of the time is proportional to the number of comparisons made between the pivot element and other elements of the subarray.
- So, the running time of `Quicksort` is `O(n + X)`, where `X` is the total number of comparisons made in all the calls to `Partition`.
- In the worst-case, `X = \Theta(n^2)`.
- For average-case analysis, treat `X` as a random variable and compute `E[X]`.

**Average-case Analysis:**

**Assumptions and Notation:**

- Array `A` contains some permutation of `n` distinct values `z_1 < z_2 < \ldots < z_n`. (So, `z_i` is the `i`th smallest value.)
- For `1 \leq i < j \leq n`, the set `Z_{i,j}` is defined to be `{\hat{z}_i, z_{i+1}, \ldots, z_j}`.
Further observations:

- Each pair of elements $z_i$ and $z_j$ gets compared at most once in the entire run of Quicksort.
- Until an element from $Z_{ij}$ gets chosen as the pivot, all elements of $Z_{ij}$ are in the same subarray produced by Partition.

Lemma 1: $z_i$ and $z_j$ are compared in a run of Quicksort if and only if $z_i$ or $z_j$ is the first pivot chosen from $Z_{ij}$.

Proof: To be presented in class.

Theorem 1: The expected number of comparisons in Randomized-Quicksort is $O(n \log n)$.

Ideas used in proving Theorem 1: (Details in class)

- Define indicator variables $x_{ij}$, $1 \leq i < j \leq n$, where $x_{ij} = 1$ if $z_i$ and $z_j$ are compared and 0 otherwise.
- The random variable $X$ for the number of comparisons made by Quicksort is given by
  $$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}$$
- Use Lemma 1 to show that $E[x_{ij}] = 2/(j - i + 1)$.
- Use linearity of expectation and known upper bound on harmonic series.