Quicksort

- divide and conquer
- partitioning with respect to a pivot element

\[
\begin{align*}
A[k] &\leq \text{pivot} \leq A[l] \\
A[l] &\leq \text{pivot} \leq A[k]
\end{align*}
\]

quick sort
choose pivot
partition $\rightarrow \begin{cases} A_1 & A_2 \end{cases}$
sort $(A_1)$
sort $(A_2)$

merge sort
\[
\begin{cases} A_1 & A_2 \end{cases}
\]
sort $(A_1)$
sort $(A_2)$
merge
\[
T(n) = 2T(\frac{n}{2}) + n
\]
private static int partition(Comparable[] a, int lo, int hi) {
    // Partition into a[lo..i-1], a[i], a[i+1..hi].
    int i = lo, j = hi+1; // left and right scan indices
    Comparable v = a[lo]; // partitioning item
    while (true) {
        // Scan right, scan left, check for scan complete, and exchange
        while (less(a[top], v)) if (i == hi) break;
        while (less(v, a[bot])) if (j == lo) break;
        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j); // Put v = a[j] into position
    return j; // with a[lo..j-1] <= a[j] <= a[j+1..hi].
}
\[
\text{quickSort} (A, \text{lo}, \text{hi}) \leftarrow \text{pivot}
\]
\[i \leftarrow \text{lo} \quad \text{recurr} \]
\[j \leftarrow \text{partition} (A, \text{lo}, \text{hi})
\] (position of pivot after partition)
\[
\text{quickSort} (A, \text{lo}, j-1)
\]
\[
\text{quickSort} (A, j+1, \text{hi})
\]

**recurrence formula**

\[
T(1) = 1 \quad n = \text{hi} - \text{lo} + 1
\]

\[
T(n) = cn + T(j - \text{lo}) + T(hi - j)
\]

solving it depends on \(j\)!
Changing at every recursion?

Worst case

\[ A B C D E F = n \]

\[ \text{pivot stays at same position} \]

\[ B C D E F = n-1 \]

\[ C D E F = n-2 \]

\[ \vdots \]

\[ 1 \]

\[ F E D C B A \]

Different example

If the array is already sorted, \( T(n) = \frac{n(n-1)}{2} = \Theta(n^2) \)
\[ T(n) = T(0) + T(n-1) + Cn \]

\[ T(1) = 1 \]

is \( \Theta(n^2) \)

worst case at every iteration

when array is sorted?

---

lucky: when pivot lands in the middle

\[ T(n) = 2T \left( \frac{n}{2} \right) + Cn \]

let's see if this case is lucky:

\[ \frac{n}{4} \]

\[ 3\frac{n}{4} \]
\[ T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 3n \]

\[ T(1) = 1 \]

**unbalanced substitution**

**recursion tree**

(you count only the part of the code)

\[ T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 3n \]
$T(n)$ is sum of all nodes.

$T_1(n) = h_1 \cdot c_n$ (how many time division by 4)

$T_2(n) = h_2 \cdot c_n$ (by $\frac{1}{3}$)

$T_2(n) \leq T(n) \leq T_1(n)$

$c_n \log_4 n \leq T(n) \leq c_n \log_4 n$

$\frac{\log_4 n}{\log 4} \leq \frac{\log n}{\log \frac{1}{3}}$ is $\Theta(n \log n)$
problem with all the above
in that pivot lands at
"some" place (middle, %...)

\[ L(n) = 2 U \left( \frac{n}{2} \right) + C_n \]

\[ U(n) = L(0) + L(n-1) + C_n n \]

\[ L(n) = 2 \left( L(0) + L \left( \frac{n}{2} - 1 \right) + C_n \frac{n}{2} \right) + C_n n \]

\[ = 2 L \left( \frac{n}{2} - 1 \right) + C' n + C \]

Alternate between lucky (best)
and unlucky (worst) gives
$\Theta(n \log n)$

Intuition behind choosing randomly the pivot Do:
- shuffling the array
- or choosing any element and swapping it with the first (every time)

Prove that the expectation of the running time when array was pre-shuffled is $\Theta(n \log n)$. 
Proof using recursion is in CLR book.

Here we follow notes (Rajiv G.)

\[ T(n) \] is total number of comparisons.

Comparison is a random event

Elements \( i \) and \( j \) are compared at \( k \)-th recursion

\[
X_{ij}^k = \begin{cases} 
1 & \text{if } A[i] \text{ compared with } A[j] \\
0 & \text{otherwise}
\end{cases}
\]

indicator variable

\[
X_{ij} = \begin{cases} 
1 & \text{if } A[i] \text{ compared with } A[j] \text{ ever} \\
0 & \text{otherwise}
\end{cases}
\]
\[
E[X_{ij}] = \sum_k E[X_{ij}^k] = \sum_k 1 \cdot \text{prob}[X_{ij}^k = 1] + 0 \cdot \text{prob}[X_{ij}^k = 0]
\]

\[
= \sum_k \text{prob}[X_{ij} = 1]
\]

All comparisons involve the pivot (either \(i\) or \(j\) pivot on \(j\)).

\[A[\bullet] \leq \nu \quad (i,j \text{ here is irrelevant to } i,j \text{ of code})\]

\[
\text{prob}[X_{ij}^k = 1] = \text{prob}(i \text{ in pivot}) + \text{prob}(j \text{ is pivot})
\]

\[
= \frac{1}{j-i+1} + \frac{1}{j-i+1}
\]

Why? Because at \(t\)-th iteration
i and j have never been compared before this iteration and will not be compared after!
pivot is one of j-i+1 numbers that's why prob ir \[ \frac{1}{j-i+1} \]

\[
E[X] = \sum_{i=1}^{\eta} \sum_{j=it?}^{\eta} \frac{e}{j-i+1}
\]

continue next time