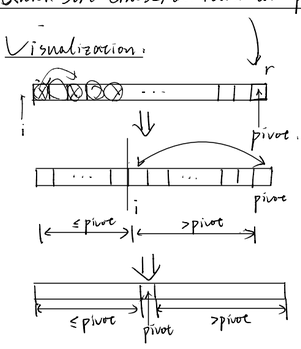
QuickSort

```
Quicksort(A,p,r):
if p≥r then return
q ← partition(A,p,r)
Quicksort(A,p,q-1)
Quicksort(A,q+1,r)
```

```
Partition(A,p,r):
x ← A[r]
i ← p-1
for j ← p to r-1:
    if A[j] ≤ x then:
        i ← i+1
        swap A[i] and A[j]
swap A[i+1] and A[r]
return i+1
```

Quick Sort chooses an item as "pivot"

Visualization:



Running Time:

Partition() → $T(n) = \Theta(n)$

Quick_Sort() → $T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \log n)$

Pivot Selection:

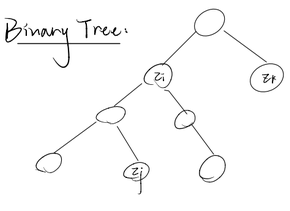
- Best case: always median ⇒ $\Theta(n \log n)$
- Worst case: always smallest ⇒ $\Theta(n^2)$

⇒ Randomly choose an item as pivot ⇒ Worst Case almost never happens!

"Randomized Algorithm"

Observation:

Binary Tree:



z_i, z_j are compared once iff:

- z_i is an ancestor of z_j
- or
- z_j is an ancestor of z_i

Otherwise they have never been compared (e.g. z_i & z_k)

Let X_{ij} be Indicator random variable.

$X_{ij} = 1$ if x_i, x_j have been compared once

$E(X_{ij}) = \frac{2}{j-i+1}$. Since if x_i is x_j's ancestor ($i < j$)

then x_i is the first pivot of subset $\{x_i \dots x_j\}$.

also noted that $\{x_i \dots x_j\}$'s pivot is random selected.

∴ "x_i is x_j's ancestor" has probability $\frac{1}{j-i+1}$

So as "x_j is x_i's ancestor"

∴ $E(X_{ij}) = \frac{2}{j-i+1}$ ($i < j$)

∴ $E(X) = \sum_{1 \le i < j \le n} E(X_{ij}) = \sum_{1 \le i < j \le n} \frac{2}{j-i+1}$

Then, will Show that

$\sum_{1 \le i < j \le n} \frac{2}{j-i+1} = \Theta(\log n)$

i \ j	2	3	4	5	.....	n	Sum of row
1	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	-------	$\frac{2}{n}$	$\Theta(\log n)$
2		$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	-------	$\frac{2}{n-1}$	$\Theta(\log n)$
3			$\frac{2}{2}$	$\frac{2}{3}$	-------	$\frac{2}{n-2}$	$\Theta(\log n)$
⋮					-----	⋮	⋮
n-1						$\frac{2}{2}$	$\Theta(\log n)$

⇒ each row has Sum of $O(\log n)$, n rows ⇒ Total Sum = $\Theta(n \log n)$

∴ $E(X) = \Theta(n \log n)$

∴ $T(n) = \Theta(n \log n)$. Since X is expected number of all comparisons. (Average Case)

Why Quick Sort (better than merge Sort)?

Small hidden Constant & Cache-efficient

Real Algorithmic Engineering

QuickSort (as first)

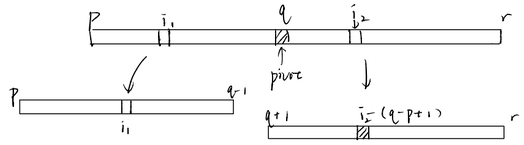
if recursion too deep → insertion Sort / heap Sort.

Random Selection

Unsorted array A[1...n], integer i.

return i-th smallest element of A[1...n]

```
Select(A,p,r,i):
if p=r then return A[p]
Randomly choose an element in A[p...r]
and swap it with A[r]                    // pivot
q ← partition(A,p,r)   // Same with partition in QuickSort
k = q-p+1
if i=k then return A[q]   // A[q] is i-th smallest element.
else if i<k then return Select(A,p,q-1,i)
else return (A,q+1,r,i-k)
```



Analysis:

if pivot falls in this area.

then "at least $\frac{1}{4}$ of items are thrown away"

⇒ Call a pivot { "good" if in this area; "bad" otherwise.

$P(\text{good}) = 1/2$, after i good pivot → total # items $\le (\frac{3}{4})^i n$

Let i-th stage be: time between i & i+1 "good" pivot.

$E(X_i) = 2$ (X_i: # of pivots in i^{th} stage)

↳ equivalent to: flip a coin (not good pivot!), already i heads, want to have i+1 heads.

$E(X_i) = \sum_{i=1}^{\infty} i (1-p)^{i-1} p = \frac{1}{p^2} \cdot p = \frac{1}{p}$

"waiting time problem"

Let Y_i be the running time of ith stage.

$Y_i \le X_i (\frac{3}{4})^i \cdot n$

$E(Y_i) \le E(X_i) \cdot (\frac{3}{4})^i n = 2 \cdot (\frac{3}{4})^i \cdot n$

$E(Y) = \sum_i E(Y_i) = 2n \cdot \sum_i (\frac{3}{4})^i = O(n)$