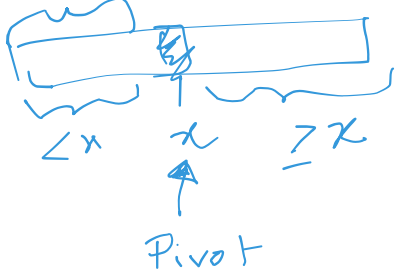
Quick Sort



SORT (L, n)

Base: $(n == 1) \Rightarrow L' = L$

Inductive:

$x = \text{select}(L)$;
$L_1 = \text{split\_GE}(L, x)$;
$L_2 = \text{split\_LT}(L, x)$;
$L_1' = \text{SORT}(L_1)$;
$L_2' = \text{SORT}(L_2)$;
} Dec.
$L' = \text{CONCAT}(L_1', L_2')$; → Rec.

Recursive D&C alg.

Unlike Merge sort, merging is trivial. (Concat)

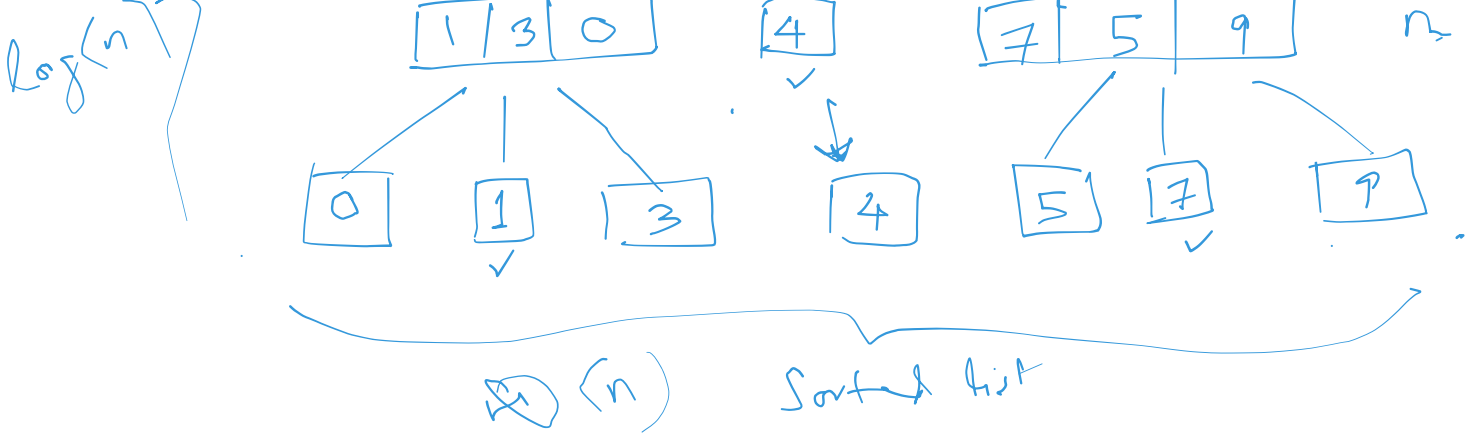
⇒ Fastest comparison-based sorting for arrays.

$\Theta(n^2)$ worst case

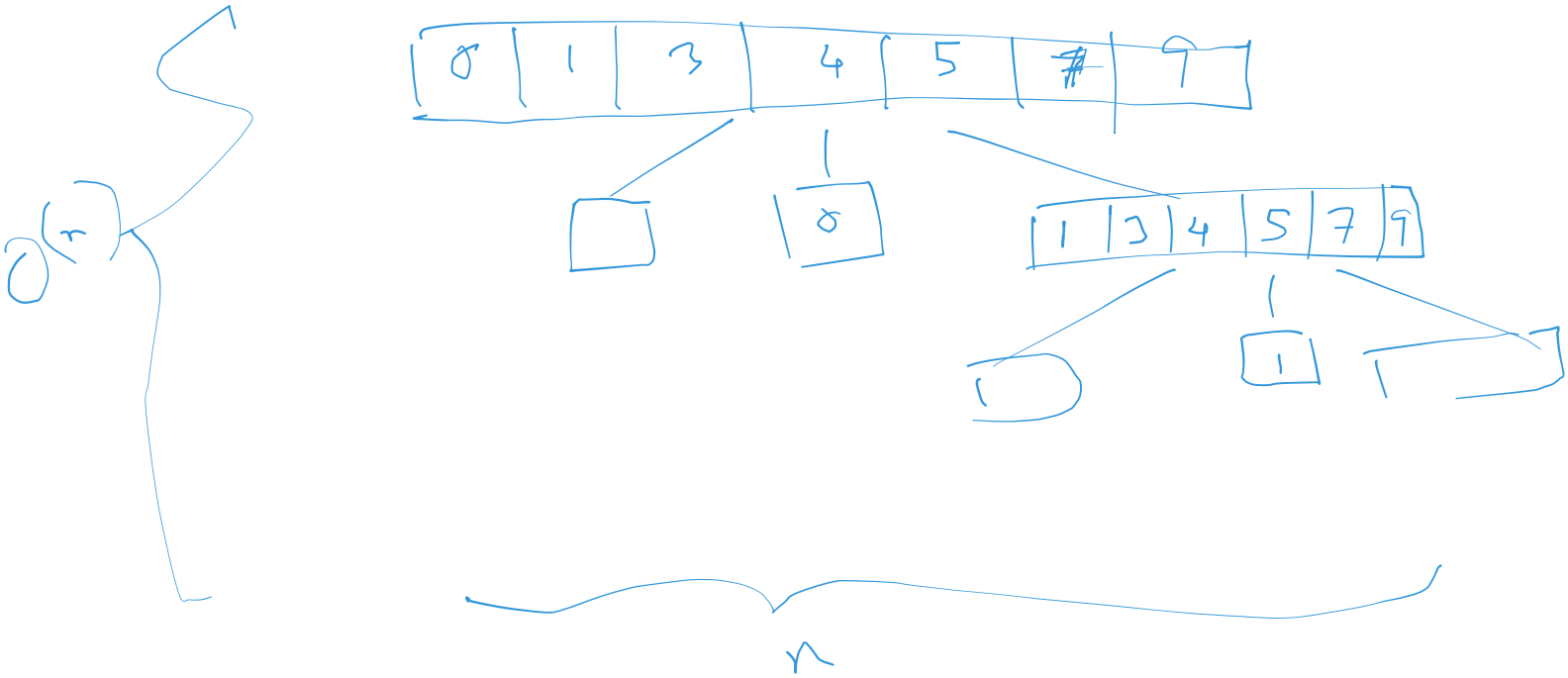
If implemented cleverly, virtually always runs in $\Theta(n \lg n)$ in practice + with a smaller constant factor.

A[0] → pivot

A: | 4 | 7 | 1 | 5 | 9 | 3 | 0 |



$\Omega(n)$ Sorted list



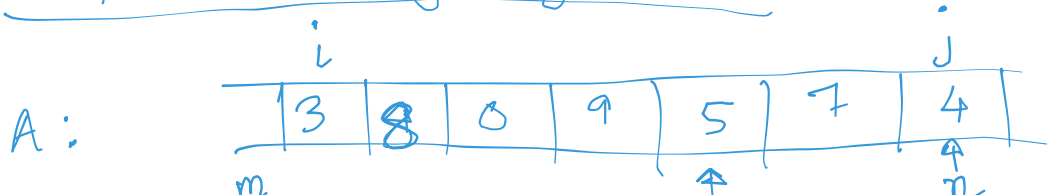
$\Rightarrow \Theta(n^2)$

When the input is already sorted, choosing the first item as pivot is disastrous.

Choice of the pivot :

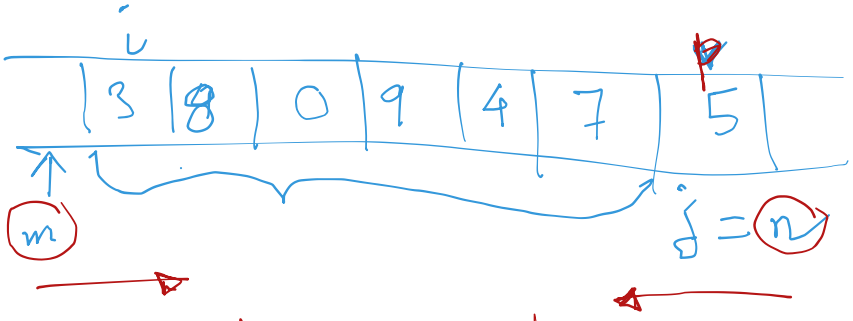
* Random selection.
* Median of 3 strategy : (first, last, mid) Median

Inplace Sorting Algorithm



(i-1)

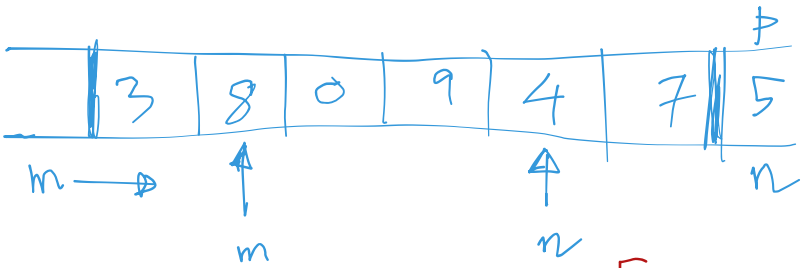
Swap A[j] and the pivot



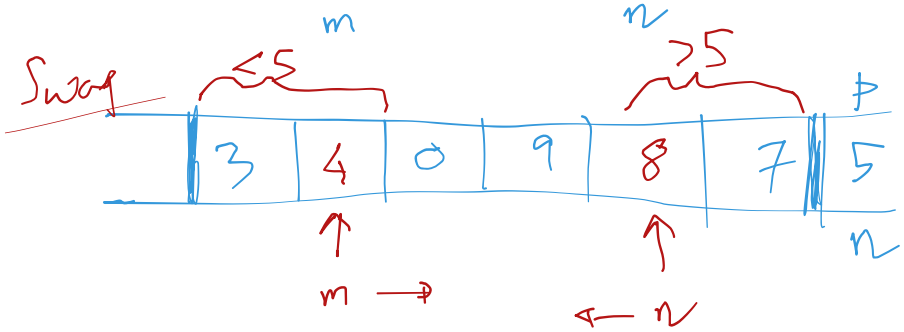
m = i - 1
n = j

Goal: All items left of m have a key ≤ p
" " right " n " " " > p

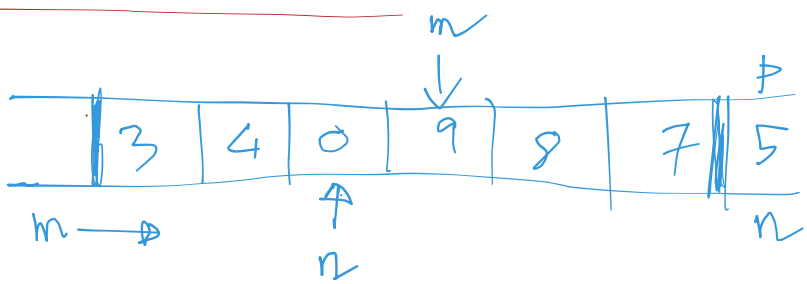
Advance m to key ≥ pivot
Decrement n to key ≤ pivot
Swap A[m] and A[n]



Swap

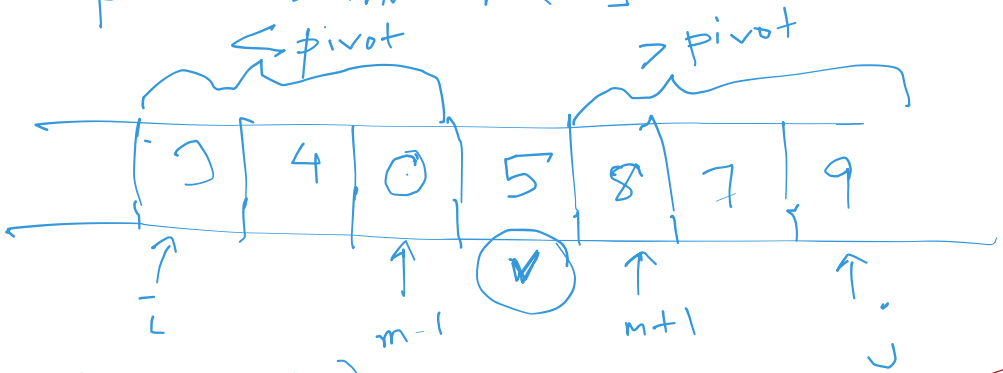


Move m and n



Repeat until m ≥ n.

Swap pivot with A[m]




```
qsort (A, i, j)
{
    if (i >= j) return;
    pindex = <some strategy>;        } Pivot selection
    pivot = A[pindex];
    // Swap pivot and A[j]
    A[pindex] = A[j];
    A[j] = pivot;                    } Swap
    // m sandwiches the array from the left
    m = i - 1
    // n sandwiches the array from the right
    n = j
    do {
      do { m++; } while (a[m] < pivot);
      do { n--; } while (a[n] > pivot &&
                         n > i);
      if (m < n) {
        Swap A[m] and A[n]
      }
    } while (m < n);
    // m >= n and therefore A[m] > pivot
    // Swap the pivot in A[i] with
```

Pivot Saved →

In place qsort

```
    // A[m]
    A[j] = A[m];
    A[m] = pivt;
    qsort (A, i, m-1);
    qsort (A, m+1, j);
}
```