# Loop Invariant for Partition

Initialization: Before the loop starts, all of the loop invariant is satisfied because A[end] = pivot + the subarrays A[start..i] + A[i+1..j-1] are empty.

Maintenance: While the loop is running, If $A[j] \leq$ pivot then A[i+1] + A[j] are swapped + i + j are incremented. If A[j] > pivot then only j is incremented.

Termination: when the loop terminates j = end so all of the elements are partitioned into one of the three cases: A[start..i] $\leq$ pivot, A[i+1..end-1] $\geq$ pivot + A[end] = pivot

Time for partitioning $\Theta(n)$

# Performance of Quicksort

- Can run as well as merge sort if subarrays are balanced + as bad insertion sort otherwise.

- Worst Case
    - Completely unbalanced (sorted in ascending or descending / all same #)
    - Recurrence Relation

$T(0) = c$

$$T(n) = T(n-1) + T(0) + \theta(n)$$

$$T(n-1) = T(n-2) + T(0) + \theta(n-1)$$

$$T(n) = T(n-2) + T(0) + \theta(n-1) + T(0) + \theta(n)$$

$$T(n-2) = T(n-3) + T(0) + \theta(n-2)$$

$$T(n) = T(n-3) + 3T(0) + \theta(n-2) + \theta(n-1) + \theta(n)$$

$\vdots$

$$T(n) = T(0) + cn + \sum_{i=1}^{n} \theta(i)$$

$$T(n) = c(n+1) + \theta\left(\frac{n(n+1)}{2}\right)$$

$$T(n) = \theta(n^2)$$

Best Case

- Completely balanced $\left(\leq \frac{(n-1)}{2}\right)$ both array sizes

$$= T(n) = 2T(n/2) + T(0) + \theta(n)$$

$$= \theta(n \log n)$$

## Average Case

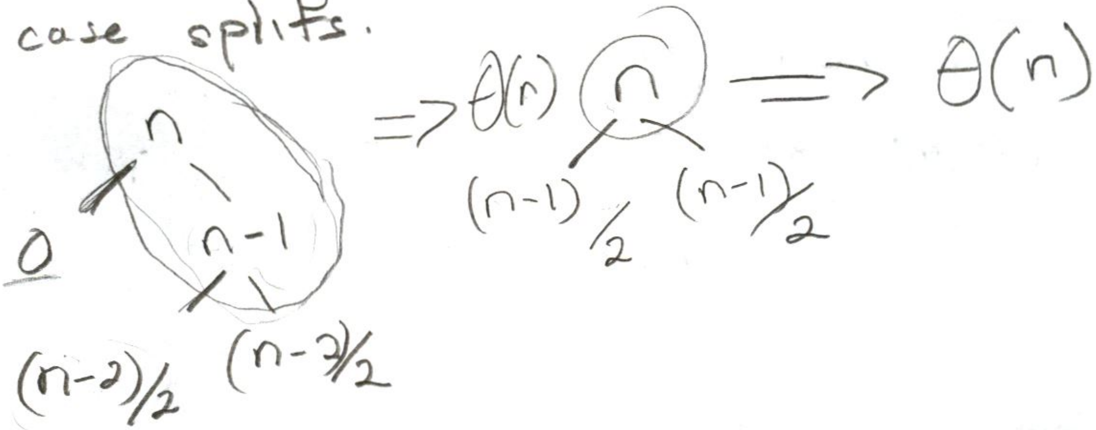
- Imagine partition always produces a 9-1 split

$$T(n) = T(9n/10) + T(n/10) + \Theta(n)$$

- As long as the base of a log is constant, it's still $\log n$ in asymptotic notation.

## Intuition

- Alternating between best + worst case splits.




Both of these recursion trees will be $\Theta(n \log n)$ but the constant hidden from the notation is higher in the left tree.

# Pseudocode

```
def random-partition(A, start, end):
    idx = random.randint(start, end)
    swap(A[idx], A[end])
    return partition(A, start, end)
```

## Analysis on Randomized Quick Sort

- The dominant cost of this algorithm is the partion function.
- Partition removes the pivot element from future consideration each time.
- Therefore, the partition function is called at most n times.
- Partition work = constant + comparisons made in the loop.
- Let X = total # of comparisons performed in all calls to partition.
- Total work for entire execution $= O(n + \underline{X})$

- Now we'll compute a bound on (X)
  - Rename the elements of A to $\{z_0, z_1 \ldots z_n\}$ where $z_i$ is the $i^{th}$ smallest element.
  - Define a set $Z_{ij} = \{z_i, z_{i+1}, \ldots z_j\}$

$A = [5, 3, 1, 4, 2]$   $z_0$ $z_1$ $z_2$ $z_3$ $z_4$ / 1 2 3 4 5   $Z_{13} = \{2\,3\,4\}$

- Each element is at most compared once.
- Let $X_{ij} = I\{z_i \text{ is compared to } z_j\}$

$$X_{23} = 1 \text{ or } 0$$

Total # of comparisons made by this algorithm:

$$X = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = E\left[\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

- All we have to do now is define in order to find the expected amount of comparisons.
- $\Pr(z_i$ is compared to $z_j) = \Pr($either $z_i$ or $z_j$ are the first pivots chosen in $Z_{ij})$