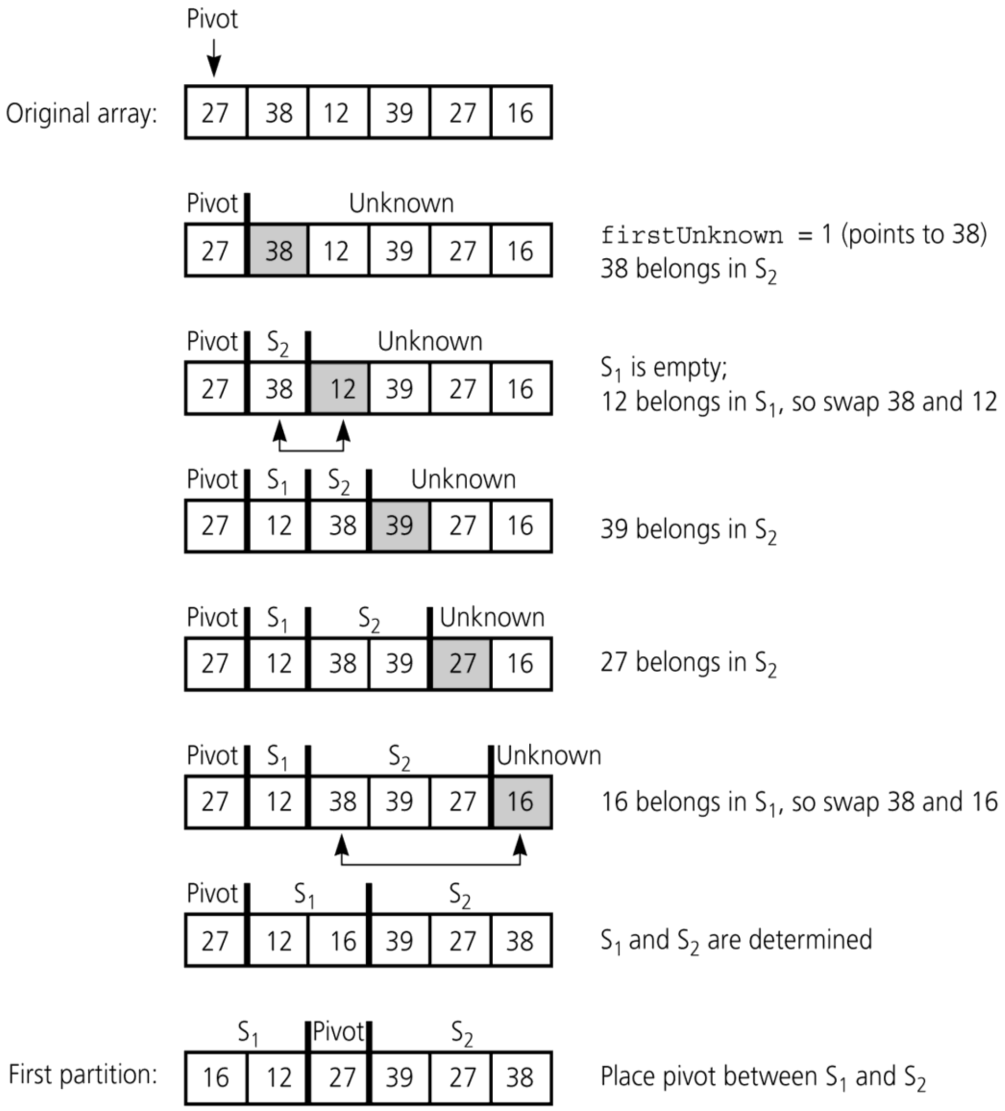
Complexity Analysis

Randomized Quicksort

Remember: QuickSort

Developing the first partition of an array when the pivot is the first item



Pivot

Original array: 27 | 38 | 12 | 39 | 27 | 16

Pivot | Unknown

27 | 38 | 12 | 39 | 27 | 16

firstUnknown = 1 (points to 38)
38 belongs in S_2

Pivot | S_2 | Unknown

27 | 38 | 12 | 39 | 27 | 16

S_1 is empty;
12 belongs in S_1, so swap 38 and 12

Pivot | S_1 | S_2 | Unknown

27 | 12 | 38 | 39 | 27 | 16

39 belongs in S_2

Pivot | S_1 | S_2 | Unknown

27 | 12 | 38 | 39 | 27 | 16

27 belongs in S_2

Pivot | S_1 | S_2 | Unknown

27 | 12 | 38 | 39 | 27 | 16

16 belongs in S_1, so swap 38 and 16

Pivot | S_1 | S_2

27 | 12 | 16 | 39 | 27 | 38

S_1 and S_2 are determined

S_1 | Pivot | S_2

First partition: 16 | 12 | 27 | 39 | 27 | 38

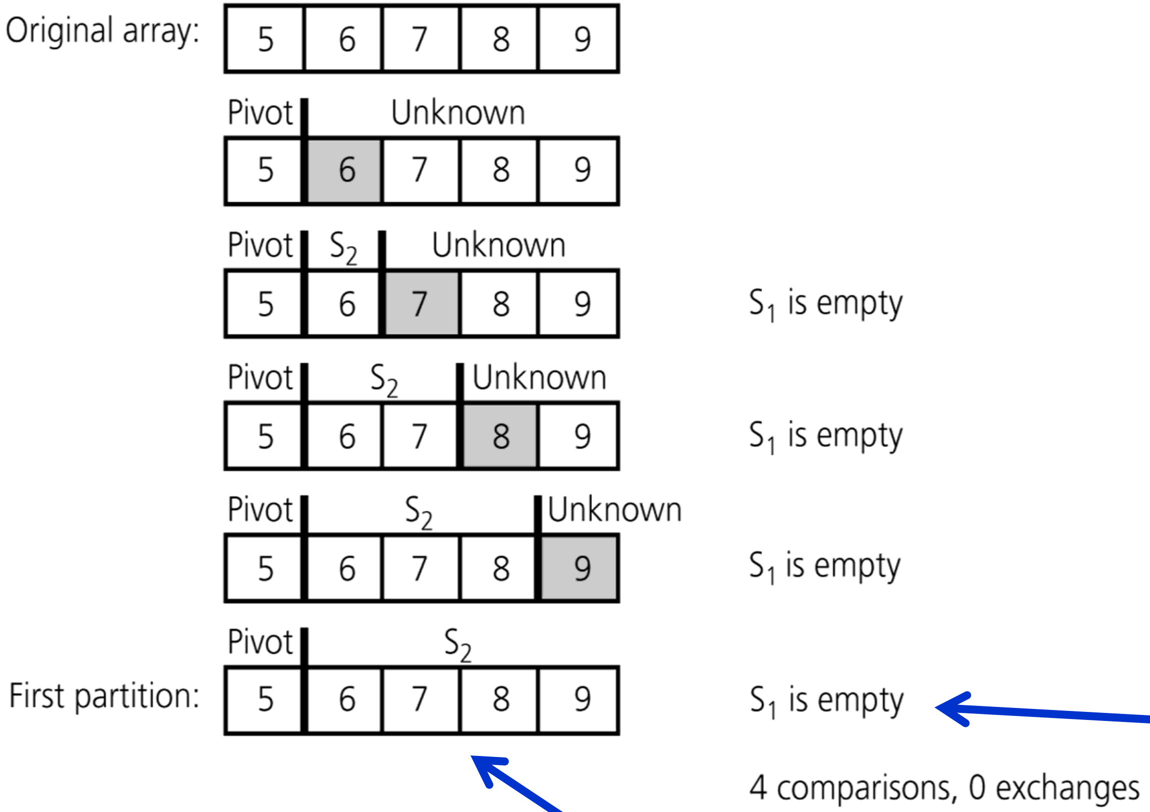
Place pivot between S_1 and S_2

Quicksort Algorithm

```
void quicksort(DataType theArray[], int first, int last) {
   if (first < last) {
        partition(theArray, first, last, pivotIndex);   ← Θ(n)
        quicksort(theArray, first, pivotIndex-1);     ← T(?)
        quicksort(theArray, pivotIndex+1, last);      ← T(?)
    }
}
```

- Partition is always Θ(n), because we need to compare all elements with pivot
- But the time for two recursive calls may depend on two partition size. So we have best case, worst case and average case for best partition, worst partition and average partition

Remember: Quicksort Worst Partition



One partition: empty

Another partition: n-1

Quicksort Worst Case: Bad partition

- Recurrence in the worst case:

  $T(n) \quad = \Theta(1)$

  $\qquad\quad = T(n - 1) + T(0) + \Theta(n) = T(n-1) + cn$

- Solution is:

  $T(n) = \Theta(n^2)$. How? Home Work (use iterative method).

Quicksort Best Case: Good partition

- In the best case:

  $T(n) = \Theta(1) = c$

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

- What does this work out to?
  - Case 2 of Master Method
  - So, we get: $T(n) = \Theta(n \log n)$ How ? Home work.

Randomized Quicksort

Randomized algorithm: if any step randomized

- Random position: 3

0	1	2	3	4	5
8	78	45	32	23	56

After partition

8	23	32	78	45	56

- Random position: 2

Sorted case (which was worst for normal partition)

Now it is not worst.

0	1	2	3	4	5
4	9	22	45	89	100

After partition

4	9	22	45	89	100

Randomized Quicksort Pseudo code

```
void RandomQS(DataType theArray[], int first, int last) {
   if (first < last) {
       Random_partition(theArray, first, last, pivotIndex);
       RandomQS(theArray, first, random_position-1);
       RandomQS(theArray, random_position+1, last);
   }
}

 void Random_partition(DataType theArray[], int first, int last,
                int &pivotIndex) {
     // DataType pivot = theArray[first];    // normal partition
        random_position = random (first, last);
        DataType pivot = theArray[random_position];
        ...
 }
```

Randomize Quicksort Analysis

```
void RandomQS(DataType theArray[], int first, int last) {
   if (first < last) {
       Random_partition(theArray, first, last, pivotIndex);
       RandomQS(theArray, first, pivotIndex-1);
       RandomQS(theArray, pivotIndex+1, last);
   }
}
```

Θ(n)

?

?

- Partition is always Θ(n), because we need to compare all elements with pivot
- But the time for two recursive calls may depend on two partition size. Here, we compute ***expected*** (for randomized algorithms, expected means average) number of comparison for all possible partitions

Some Basics on Probability/Randomness

- Probability {side 1} = ½
  Probability {side 2) = ½

- Probability {1} = 1/6
  Probability {2) =1/6, …….
- So, in general, **Probability = 1/total number**

- Expected number of {1} in100 run/through = 100*1/6 = 16.66
- Expected number of {2} in 100 run/through = 100*1/6 = 16.66

  …
- Expected number of {1} in 1 run/through = 1*1/6 = 1.66
- Expected number of {2} in 1 run/through = 1*1/6 = 1.66

  ….
- In general: **Expected number = count*probability**

Randomized Quicksort: ***Expected number*** of comparison

- **For one partition:** Which two elements are compared? One element is always a pivot
- So, all comparisons are with pivots. No comparison among non-pivot elements.
- How many times two elements are compared? 0 or 1. Because, after partition, pivot is not compared again
- So, two elements a and b are compared when a is pivot or b is pivot
- Probability{a is pivot} = $\frac{1}{total\, elements} = \frac{1}{last - first + 1}$
- Similarly, Probability{b is pivot} = $\frac{1}{last - first + 1}$
- Total Probability{a or b is pivot} = $\frac{1}{last - first + 1} + \frac{1}{last - first + 1} = \frac{2}{last - first + 1}$
- Expected number of comparison with a or b is pivot = $1 * \frac{2}{last - first + 1} = \frac{2}{last - first + 1}$

Randomized Quicksort: Expected number of comparison

- **For all possible partitions:** All possible first, last (for short, *l*: last, *f*: first). *f* can be anything from 1..n-1, and *l* anything after that.

$$= \sum_{f=1}^{n-1} \sum_{l=f+1}^{n} \frac{2}{l-f+1}$$

$$Take: l-f=k. \quad So, l=f+1 \quad means \quad k=1$$

$$and\, l=n \quad means \quad k=n-f$$

$$= \sum_{f=1}^{n-1} \sum_{k=1}^{n-f} \frac{2}{k+1} < \sum_{f=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \qquad // \sum_{k=1}^{k=n} 1/k = O(\log n)$$

$$= \sum_{i=1}^{n-1} 2O(\log n) = \sum_{i=1}^{n-1} O(\log n) = (n-1)O(\log n) = O(n\log n) - O(\log n) = O(n\log n).$$

- So, Randomized Quick sort is O(n log n) on average.
- Best sorting algorithm so far.