Quick Sort
Another Divide and Conquer

- Choose a pivot item $p$
- Partition array $S[\text{first}..\text{last}]$ about $p$
quicksort(int *array, int first, int last) {
    if (first<last) {
        choose a pivot item p from array[first..last];
        partition the items about p
        //the partition is array[first..pivotIdx..last]

        //sort S1
        quicksort(array, first, pivotIdx-1);
        //sort S2
        quicksort(array, pivotIdx+1, last);
    }
    //if first>=last, there is nothing to do
}

Choose a Pivot

- Choose a pivot at random
  - Put it in array[first]

- Initially all items except pivot constitute the unknown region
  - lastS1 = first, firstUnknown = first + 1
During Partition

- Invariant for the partition algorithm
  - Items in S1 are less than pivot
  - Items in S2 are greater than or equal to pivot
- Pivot = array[first], S1[first+1..lastS1], S2[lastS1+1..firstUnknown-1], Unknown[firstUnknown..last]
int partition(int *array, int first, int last) {
    choose pivot and swap it with array[first];
    p = array[first];
    lastS1 = first;
    firstUnknown = first+1;
    //determine the regions S1 and S2
    while (firstUnknown<=last){
        if (array[firstUnknown]<p)
            move array[firstUnknown] into S1
        else
            move array[firstUnknown] into S2
    }
    swap array[first] with array[lastS1];
    return lastS1;  //lastS1 is pivot index
}
Move Operations

- Move array[firstUnknown] into S1

- Only 3 operations!
  - Swap array[firstUnknown] with array[lastS1+1]
  - lastS1++, firstUnknown++;
Move Operations

- Move array[firstUnknown] into S2

- No move at all!
  - firstUnknown++;

![Diagram showing array segmentation with conditions and pointers](image-url)
How Partition Works?

Original array:

```
27 38 12 39 27 16
```

Pivot

```
27 38 12 39 27 16
```

Pivot

```
27 38 12 39 27 16
```

Unknown

```
27 38 12 39 27 16
```

firstUnknown = 1 (points to 38)

38 belongs in S2

S1 is empty;
12 belongs in S1, so swap 38 and 12

Pivot

```
27 38 12 39 27 16
```

S2

```
27 38 12 39 27 16
```

Unknown

```
27 38 12 39 27 16
```

39 belongs in S2

Pivot

```
27 12 38 39 27 16
```

S1

```
27 12 38 39 27 16
```

Unknown

```
27 12 38 39 27 16
```

27 belongs in S2

Pivot

```
27 12 38 39 27 16
```

S1

```
27 12 38 39 27 16
```

S2

```
27 12 38 39 27 16
```

Unknown

```
27 12 38 39 27 16
```

16 belongs in S1, so swap 38 and 16

Pivot

```
27 12 38 39 27 16
```

S1

```
27 12 38 39 27 16
```

S2

```
27 12 38 39 27 16
```

Unknown

```
27 12 38 39 27 16
```

S1 and S2 are determined

First partition:

```
16 12 27 39 27 38
```

Place pivot between S1 and S2
Mergesort v.s. Quicksort

- Similar
- But quicksort does its work before recursive calls, mergesort does it after.

```c
quicksort() {
    if (first<last) {
        prepare array
        quicksort();
        quicksort();
    }
}
```

```c
mergesort() {
    if (first<last) {
        mergesort();
        mergesort();
        tidy up array
    }
}
```
### Analysis - Worst Case

<table>
<thead>
<tr>
<th>Original array:</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pivot</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>Pivot</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>Pivot</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>First partition:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pivot</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

S₁ is empty

4 comparisons, 0 exchanges
Analysis-Worst Case

- If the array[1..n] is already sorted
  - Requires n-1 comparison
  - Decrease the size of array by 1 and pass it to recursive calls
  - All together, it requires (n-1)+...+2+1=n(n-1)/2 comparisons
  - O(n^2)
Analysis-Average Case

Original array:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Pivot Unknown

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Pivot $S_1$ Unknown

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Pivot $S_1$ $S_2$ Unknown

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Pivot $S_1$ $S_2$ Unknown

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Pivot $S_1$ $S_2$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

$S_1$ and $S_2$ are determined

First partition:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Place pivot between $S_1$ and $S_2$
Analysis-Average Case

- Similar analysis to mergesort
  - The number of recursive levels is $\log(n)$ or $1+\log(n)$ [rounded down]
  - Each level requires $m$ comparisons and at most $m$ exchanges, $m<n$
  - It is $O(n)$
- So average-case is $O(n*\log(n))$