Figure 8.10  Quicksort

Select pivot

Partition

Quicksort small items

Quicksort large items
Partition

**Figure A** If 6 is used as pivot, the end result after partitioning is as shown in the Figure B.

![Diagram of partitioning process with 6 as pivot]

**Figure B** Result after Partitioning

```
2 1 4 5 0 3 6 8 7 9
```
algorithm **QuickSort**(array a, integer p, integer r)
{
    if (p < r) then
        q = **Partition**(a, p, r)
        **QuickSort**(a, p, q-1)
        **QuickSort**(a, q+1, r)

}  

algorithm **Partition**(array A, integer p, integer r)
{
    x = a[r]
    i = p-1
    for j = p to r-1 do
        if a[j] <= x) then
            i++
            exchange(a[i], a[j])
    exchange(a[i+1], a[r])
    return i+1
}
QuickSort

<table>
<thead>
<tr>
<th>Array indices</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>p = 0; r = 7; i = -1; j = 0</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 0; a[0] ↔ a[0]; j = 1</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 0; j = 2</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 0; j = 3</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 1; a[1] ↔ a[3]; j = 4</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 2; a[2] ↔ a[4]; j = 5</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 2; j = 6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>i = 2; j = 7 &gt; 6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>a[3] ↔ a[7]</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
QuickSort

<table>
<thead>
<tr>
<th>Iteration k</th>
<th>p</th>
<th>≤x</th>
<th>i</th>
<th>&gt;x</th>
<th>j</th>
<th>r</th>
</tr>
</thead>
</table>

Invariant:
Storing binary trees as arrays
**Heaps (Max-Heap)**

```
43  16  38  4  7  37  20
```

```
43  16  38  4  7  37  20  2  3  6  1  30
```

**HEAP** represents a binary tree stored as an array such that:
- Tree is filled on all levels except last
- Last level is filled from left to right
- Left & right child of i are in locations 2i and 2i+1

**HEAP PROPERTY:**
Parent value is at least as large as child’s value
HeapSort

• First convert array into a heap (BUILD-MAX-HEAP, p133)
• Then convert heap into sorted array (HEAPSORT, p136)
For the HeapSort analysis, we need to compute:

\[ \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \]

We know from the formula for geometric series that

\[ \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \]

Differentiating both sides, we get

\[ \sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2} \]

Multiplying both sides by \( x \) we get

\[ \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \]

Now replace \( x = 1/2 \) to show that

\[ \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \leq \frac{1}{2} \]
Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements
### Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Shaker Sort

- Merge Sort
- Heap Sort
- Quick Sort

- Bucket & Radix Sort
- Counting Sort
Animation Demos

http://www-cse.uta.edu/~holder/courses/cse2320/lectures/applets/sort1/heapsort.html

http://cg.scs.carleton.ca/~morin/misc/sortalg/
Bucket Sort

- N values in the range \([a..a+m-1]\)
- For e.g., sort a list of 50 scores in the range \([0..9]\).

**Algorithm**
- Make m buckets \([a..a+m-1]\)
- As you read elements throw into appropriate bucket
- Output contents of buckets \([0..m]\) in that order

**Time** \(O(N+m)\)
Stable Sort

- A sort is **stable** if equal elements appear in the same order in both the input and the output.
- Which sorts are stable? Homework!
Radix Sort

<table>
<thead>
<tr>
<th></th>
<th>3 2 9</th>
<th>7 2 0</th>
<th>7 2 0</th>
<th>3 2 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5 7</td>
<td>3 5 5</td>
<td>3 2 9</td>
<td>3 5 5</td>
<td></td>
</tr>
<tr>
<td>6 5 7</td>
<td>4 3 6</td>
<td>4 3 6</td>
<td>4 3 6</td>
<td></td>
</tr>
<tr>
<td>8 3 9</td>
<td>4 5 7</td>
<td>8 3 9</td>
<td>4 5 7</td>
<td></td>
</tr>
<tr>
<td>4 3 6</td>
<td>6 5 7</td>
<td>3 5 5</td>
<td>6 5 7</td>
<td></td>
</tr>
<tr>
<td>7 2 0</td>
<td>3 2 9</td>
<td>4 5 7</td>
<td>7 2 0</td>
<td></td>
</tr>
<tr>
<td>3 5 5</td>
<td>8 3 9</td>
<td>6 5 7</td>
<td>8 3 9</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm**

for i = 1 to d do

    sort array A on digit i using a stable sort algorithm

**Time Complexity:** $O((n+k)d)$
Counting Sort

Initial Array

Counts

Cumulative Counts

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>