# Quick Sort

## Outline

# Partitioning

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

## Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Q | U | I | C | K | S | O | R | T | E | X | A | M | P | L | E |

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E | C | A | I | E | K | L | P | U | T | M | Q | R | X | O | S |

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

## Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

# Partitioning

Quick sort works by partitioning the input array, about a pivot, into two subarrays, and then sorting the subarrays independently

Example

a[]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

# Partitioning

# Partitioning

The partitioning procedure rearranges the array such that:

## Partitioning

The partitioning procedure rearranges the array such that:

- The entry `a[j]` is in its final place in the array, for some `j`

## Partitioning

The partitioning procedure rearranges the array such that:

- The entry `a[j]` is in its final place in the array, for some `j`
- No entry in `a[lo]` through `a[j - 1]` is greater than `a[j]`

# Partitioning

The partitioning procedure rearranges the array such that:

- The entry `a[j]` is in its final place in the array, for some `j`
- No entry in `a[lo]` through `a[j - 1]` is greater than `a[j]`
- No entry in `a[j + 1]` through `a[hi]` is less than `a[j]`

# Partitioning

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | | K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 16 | K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 12 | K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 12 | K | C | A | T | E | L | E | P | U | I | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 9 | K | C | A | T | E | L | E | P | U | I | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 9 | K | C | A | I | E | L | E | P | U | T | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 5 | 6 | K | C | A | I | E | L | E | P | U | T | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 5 | 6 | K | C | A | I | E | E | L | P | U | T | M | Q | R | X | O | S |

# Partitioning

| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 5 | K | C | A | I | E | E | L | P | U | T | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 6 | 5 | E | C | A | I | E | K | L | P | U | T | M | Q | R | X | O | S |

# Partitioning

| | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 5 | E | C | A | I | E | K | L | P | U | T | M | Q | R | X | O | S |

# Partitioning

# Partitioning

Quick.java

```java
public class Quick {
    private static int partition(Comparable[] a, int lo, int hi) {
        int i = lo;
        int j = hi + 1;
        Comparable v = a[lo];
        while (true) {
            while (less(a[++i], v)) {
                if (i == hi) {
                    break;
                }
            }
            while (less(v, a[--j])) {
                if (j == lo) {
                    break;
                }
            }
            if (i >= j) {
                break;
            }
            exchange(a, i, j);
        }
        exchange(a, lo, j);
        return j;
    }

    private static int partition(Object[] a, int lo, int hi, Comparator c) {
        int i = lo;
        int j = hi + 1;
        Object v = a[lo];
        while (true) {
            while (less(a[++i], v, c)) {
                if (i == hi) {
                    break;
                }
            }
```

# Partitioning

Quick.java

```
            while (less(v, a[--j], c)) {
                if (j == lo) {
                    break;
                }
            }
            if (i >= j) {
                break;
            }
            exchange(a, i, j);
        }
        exchange(a, lo, j);
        return j;
    }
}
```

# Sorting

| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Q | U | I | C | K | S | O | R | T | E | X | A | M | P | L | E |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | | | K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 5 | 15 | E | C | A | I | E | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 3 | 4 | E | C | A | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 2 | 2 | A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | | 1 | A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | a[] | | | | | | | | | | | | | | | |
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 4 | | 4 | A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 6 | 6 | 15 | A | C | E | E | I | K | L | P | U | T | M | Q | R | X | O | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 9 | 15 | A | C | E | E | I | K | L | M | O | P | T | Q | R | X | U | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 7 | 8 | A | C | E | E | I | K | L | M | O | P | T | Q | R | X | U | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 8 | | 8 | A | C | E | E | I | K | L | M | O | P | T | Q | R | X | U | S |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 10 | 13 | 15 | A | C | E | E | I | K | L | M | O | P | S | Q | R | T | U | X |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 10 | 12 | 12 | A | C | E | E | I | K | L | M | O | P | R | Q | S | T | U | X |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 10 | 11 | 11 | A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 10 | | 10 | A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 14 | 14 | 15 | A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

# Sorting

| | | | a[] | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 15 | | 15 | A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

| | | | a[] | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lo | j | hi | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | | | A | C | E | E | I | K | L | M | O | P | Q | R | S | T | U | X |

# Sorting

# Sorting

Quick.java

```
public class Quick {
    public static void sort(Comparable[] a) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    public static void sort(Object[] a, Comparator c) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1, c);
    }

    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) {
            return;
        }
        int j = partition(a, lo, hi);
        sort(a, lo, j - 1);
        sort(a, j + 1, hi);
    }

    private static void sort(Object[] a, int lo, int hi, Comparator c) {
        if (hi <= lo) {
            return;
        }
        int j = partition(a, lo, hi, c);
        sort(a, lo, j - 1, c);
        sort(a, j + 1, hi, c);
    }
}
```

# Sorting

Quick.java

```java
public class Quick {
    public static void sort(Comparable[] a) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    public static void sort(Object[] a, Comparator c) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1, c);
    }

    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) {
            return;
        }
        int j = partition(a, lo, hi);
        sort(a, lo, j - 1);
        sort(a, j + 1, hi);
    }

    private static void sort(Object[] a, int lo, int hi, Comparator c) {
        if (hi <= lo) {
            return;
        }
        int j = partition(a, lo, hi, c);
        sort(a, lo, j - 1, c);
        sort(a, j + 1, hi, c);
    }
}
```

$$T(n) = n \log n$$

# An Algorithmic Improvement

# An Algorithmic Improvement

When the input array has a large number of duplicates, 3-way quick sort can significantly improve performance

## An Algorithmic Improvement

When the input array has a large number of duplicates, 3-way quick sort can significantly improve performance

3-way quick sort partitions the array into three parts, one each for keys smaller than, equal to, and larger than the pivot

# An Algorithmic Improvement

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| | | | B | R | B | R | W | W | W | R | R | W | R | B |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| | | | R | B | W | W | R | W | B | R | R | W | B | R |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 0 | 11 | R | B | W | W | R | W | B | R | R | W | B | R |

# An Algorithmic Improvement

| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 11 | R | B | W | W | R | W | B | R | R | W | B | R |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 2 | 11 | B | R | W | W | R | W | B | R | R | W | B | R |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 2 | 10 | B | R | R | W | R | W | B | R | R | W | B | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 3 | 10 | B | R | R | W | R | W | B | R | R | W | B | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 3 | 9 | B | R | R | B | R | W | B | R | R | W | W | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 4 | 9 | B | B | R | R | R | W | B | R | R | W | W | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 5 | 9 | B | B | R | R | R | W | B | R | R | W | W | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 5 | 8 | B | B | R | R | R | W | B | R | R | W | W | W |

## An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 5 | 7 | B | B | R | R | R | R | B | R | W | W | W | W |

## An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2 | 6 | 7 | B | B | R | R | R | R | B | R | W | W | W | W |

## An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 3 | 7 | 7 | B | B | B | R | R | R | R | R | W | W | W | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 3 | 8 | 7 | B | B | B | R | R | R | R | R | W | W | W | W |

# An Algorithmic Improvement

| | | | a[] | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lt | i | gt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 3 | 8 | 7 | B | B | B | R | R | R | R | R | W | W | W | W |

# An Algorithmic Improvement

## An Algorithmic Improvement

Quick3way.java

```
public class Quick3way {
    public static void sort(Comparable[] a) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    public static void sort(Object[] a, Comparator c) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1, c);
    }

    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) {
            return;
        }
        int lt = lo, gt = hi;
        Comparable v = a[lo];
        int i = lo + 1;
        while (i <= gt) {
            int cmp = a[i].compareTo(v);
            if (cmp < 0) {
                exchange(a, lt++, i++);
            } else if (cmp > 0) {
                exchange(a, i, gt--);
            } else {
                i++;
            }
        }
        sort(a, lo, lt - 1);
        sort(a, gt + 1, hi);
    }

    private static void sort(Object[] a, int lo, int hi, Comparator c) {
        if (hi <= lo) {
            return;
```

# An Algorithmic Improvement

Quick3way.java

```
        }
        int lt = lo, gt = hi;
        Object v = a[lo];
        int i = lo + 1;
        while (i <= gt) {
            int cmp = c.compare(a[i], v);
            if (cmp < 0) {
                exchange(a, lt++, i++);
            } else if (cmp > 0) {
                exchange(a, i, gt--);
            } else {
                i++;
            }
        }
        sort(a, lo, lt - 1, c);
        sort(a, gt + 1, hi, c);
    }
}
```

# An Algorithmic Improvement

## An Algorithmic Improvement

Given an array of $n$ keys with $k$ of them distinct, let $f_i$ be the frequency of the $i$th key and $p_i = f_i/n$ the probability that the $i$th key is found when the array is sampled

## An Algorithmic Improvement

Given an array of $n$ keys with $k$ of them distinct, let $f_i$ be the frequency of the $i$th key and $p_i = f_i/n$ the probability that the $i$th key is found when the array is sampled

The Shannon entropy of the keys is defined as

$$\begin{aligned} H &= -(p_1 \lg p_1 + p_2 \lg p_2 + \cdots + p_k \lg p_k) \\ &= -\sum_{i=1}^{k} p_i \lg p_i \end{aligned}$$

# An Algorithmic Improvement

Given an array of $n$ keys with $k$ of them distinct, let $f_i$ be the frequency of the $i$th key and $p_i = f_i/n$ the probability that the $i$th key is found when the array is sampled

The Shannon entropy of the keys is defined as

$$
\begin{aligned}
H &= -(p_1 \lg p_1 + p_2 \lg p_2 + \cdots + p_k \lg p_k) \\
&= -\sum_{i=1}^{k} p_i \lg p_i
\end{aligned}
$$

Note that $0 \leq H \leq \ln n$

## An Algorithmic Improvement

Given an array of $n$ keys with $k$ of them distinct, let $f_i$ be the frequency of the $i$th key and $p_i = f_i/n$ the probability that the $i$th key is found when the array is sampled

The Shannon entropy of the keys is defined as

$$\begin{aligned} H &= -(p_1 \lg p_1 + p_2 \lg p_2 + \cdots + p_k \lg p_k) \\ &= -\sum_{i=1}^{k} p_i \lg p_i \end{aligned}$$

Note that $0 \leq H \leq \ln n$

Running time for 3-way quick sort in terms of Shannon entropy is $T(n) = nH$