Sorting Applications

Outline

Performance Characteristics of Sorting Algorithms

Performance Characteristics of Sorting Algorithms

Algorithm	Stable?	In-place?	$T(n)$	$S(n)$
Selection	no	yes	n^2	1
Insertion	yes	yes	n^2	1
Shell	no	yes	$n \log n$?	1
Merge	yes	no	$n \log n$	n
Quick	no	yes	$n \log n$	1
3-way quick	no	yes	nH	1
Heap	no	yes	$n \log n$	1

Performance Characteristics of Sorting Algorithms

Performance Characteristics of Sorting Algorithms

To sort an array `a` of numbers efficiently, we can replace `Comparable` with the primitive type name, replace calls to `less()` with `a[i] < a[j]`, and inline any calls to `exchange()`

Performance Characteristics of Sorting Algorithms

To sort an array `a` of numbers efficiently, we can replace `Comparable` with the primitive type name, replace calls to `less()` with `a[i] < a[j]`, and inline any calls to `exchange()`

Example

Insertion.java

```
public class Insertion {
    public static void sort(int[] a) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && a[j] < a[j - 1]; j--) {
                int swap = a[i];
                a[i] = a[j];
                a[j] = swap;
            }
        }
    }

    public static void sort(double[] a) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && a[j] < a[j - 1]; j--) {
                double swap = a[i];
                a[i] = a[j];
                a[j] = swap;
            }
        }
    }
}
```

Performance Characteristics of Sorting Algorithms

To sort an array `a` of numbers efficiently, we can replace `Comparable` with the primitive type name, replace calls to `less()` with `a[i] < a[j]`, and inline any calls to `exchange()`

Example

Insertion.java

```
public class Insertion {
    public static void sort(int[] a) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && a[j] < a[j - 1]; j--) {
                int swap = a[i];
                a[i] = a[j];
                a[j] = swap;
            }
        }
    }

    public static void sort(double[] a) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && a[j] < a[j - 1]; j--) {
                double swap = a[i];
                a[i] = a[j];
                a[j] = swap;
            }
        }
    }
}
```

`java.util.Arrays.sort()` uses 3-way quick sort for primitives and merge sort for objects

Applications

Applications

We can use sorting algorithms to solve other problems — a technique in algorithm design known as reduction

Applications

We can use sorting algorithms to solve other problems — a technique in algorithm design known as reduction

Example:

Applications

We can use sorting algorithms to solve other problems — a technique in algorithm design known as reduction

Example:

- Duplicates: finding number of unique keys in a collection of keys

Applications

We can use sorting algorithms to solve other problems — a technique in algorithm design known as reduction

Example:

- Duplicates: finding number of unique keys in a collection of keys
- Median: finding the median (the value with the property that half the keys are no larger and half the keys are no smaller) of a collection of keys

Applications

Applications

Program: `Rhymer.java`

Program: `Rhymer.java`

- Standard input: a sequence of words

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs
$ _
```

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs
$ java Rhymer
```

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs

$ java Rhymer
she sells sea shells on the sea shore
```

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs

$ java Rhymer
she sells sea shells on the sea shore
-
```

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs

$ java Rhymer
she sells sea shells on the sea shore
<ctrl-d>
```

Applications

Program: `Rhymer.java`

- Standard input: a sequence of words
- Standard output: the words such that rhyming words appear next to one another

```
>_ ~/workspace/dsaj/programs

$ java Rhymer
she sells sea shells on the sea shore
<ctrl-d>
sea
sea
she
the
shore
on
shells
sells
$ _
```

Applications

Applications

Rhymer.java

```
import java.util.Arrays;

import stdlib.StdIn;
import stdlib.StdOut;

public class Rhymer {
    public static void main(String[] args) {
        String[] strings = StdIn.readAllStrings();
        for (int i = 0; i < strings.length; i++) {
            strings[i] = reverse(strings[i]);
        }
        Arrays.sort(strings);
        for (int i = 0; i < strings.length; i++) {
            strings[i] = reverse(strings[i]);
        }
        for (String s : strings) {
            StdOut.println(s);
        }
    }

    private static String reverse(String s) {
        int n = s.length();
        if (n < 2) {
            return s;
        }
        return s.charAt(n - 1) + reverse(s.substring(0, n - 1));
    }
}
```

Applications

Applications

Given a collection a, the number of inversions in a is the number of unordered pairs (a_i, a_j) in a such that $i < j$ and $a_i > a_j$

Applications

Given a collection a, the number of inversions in a is the number of unordered pairs (a_i, a_j) in a such that $i < j$ and $a_i > a_j$

For example, if $a = \{1, 2, 3, 4, 6, 8, 5, 7\}$, the number of inversions is 3

Applications

Given a collection a, the number of inversions in a is the number of unordered pairs (a_i, a_j) in a such that $i < j$ and $a_i > a_j$

For example, if $a = \{1, 2, 3, 4, 6, 8, 5, 7\}$, the number of inversions is 3

Brute-force solution

```
private static long count(Comparable[] a) {
    long inversions = 0;
    int n = a.length;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (less(a[j], a[i])) {
                inversions++;
            }
        }
    }
    return inversions;
}
```

Applications

Applications

Program: `Inversions.java`

Program: `Inversions.java`

- Standard input: a sequence of integers

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs
$ _
```

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs
$ java dsa.Inversions
```

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs

$ java dsa.Inversions
1 2 3 4 6 8 5 7
```

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs

$ java dsa.Inversions
1 2 3 4 6 8 5 7
-
```

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs

$ java dsa.Inversions
1 2 3 4 6 8 5 7
<ctrl-d>
```

Applications

Program: `Inversions.java`

- Standard input: a sequence of integers
- Standard output: the number of inversions

```
>_ ~/workspace/dsaj/programs

$ java dsa.Inversions
1 2 3 4 6 8 5 7
<ctrl-d>
3
$ _
```

Applications

Applications

Inversions.java

```
package dsa;

import java.util.Comparator;

import stdlib.StdIn;
import stdlib.StdOut;

public class Inversions {
    public static long count(Comparable[] a) {
        Comparable[] b = a.clone();
        Comparable[] aux = a.clone();
        return count(b, aux, 0, a.length - 1);
    }

    public static long count(Object[] a, Comparator c) {
        Object[] b = a.clone();
        Object[] aux = a.clone();
        return count(b, aux, 0, a.length - 1, c);
    }

    public static long count(int[] a) {
        int[] b = a.clone();
        int[] aux = a.clone();
        return count(b, aux, 0, a.length - 1);
    }

    public static long count(double[] a) {
        double[] b = a.clone();
        double[] aux = a.clone();
        return count(b, aux, 0, a.length - 1);
    }

    private static long count(Comparable[] a, Comparable[] aux, int lo, int hi) {
        long inversions = 0;
        if (hi <= lo) {
```

Applications

Inversions.java

```
            return 0;
        }
        int mid = lo + (hi - lo) / 2;
        inversions += count(a, aux, lo, mid);
        inversions += count(a, aux, mid + 1, hi);
        inversions += merge(a, aux, lo, mid, hi);
        return inversions;
    }

    private static long count(Object[] a, Object[] aux, int lo, int hi, Comparator c) {
        long inversions = 0;
        if (hi <= lo) {
            return 0;
        }
        int mid = lo + (hi - lo) / 2;
        inversions += count(a, aux, lo, mid, c);
        inversions += count(a, aux, mid + 1, hi, c);
        inversions += merge(a, aux, lo, mid, hi, c);
        return inversions;
    }

    private static long count(int[] a, int[] aux, int lo, int hi) {
        long inversions = 0;
        if (hi <= lo) {
            return 0;
        }
        int mid = lo + (hi - lo) / 2;
        inversions += count(a, aux, lo, mid);
        inversions += count(a, aux, mid + 1, hi);
        inversions += merge(a, aux, lo, mid, hi);
        return inversions;
    }

    private static long count(double[] a, double[] aux, int lo, int hi) {
        long inversions = 0;
```

Applications

Inversions.java

```
        if (hi <= lo) {
            return 0;
        }
        int mid = lo + (hi - lo) / 2;
        inversions += count(a, aux, lo, mid);
        inversions += count(a, aux, mid + 1, hi);
        inversions += merge(a, aux, lo, mid, hi);
        return inversions;
    }

    private static long merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
        long inversions = 0;
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
        }
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid) {
                a[k] = aux[j++];
            } else if (j > hi) {
                a[k] = aux[i++];
            } else if (less(aux[j], aux[i])) {
                a[k] = aux[j++];
                inversions += (mid - i + 1);
            } else {
                a[k] = aux[i++];
            }
        }
        return inversions;
    }

    private static long merge(Object[] a, Object[] aux, int lo, int mid, int hi, Comparator c) {
        long inversions = 0;
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
```

Applications

Inversions.java

```
        }
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid) {
                a[k] = aux[j++];
            } else if (j > hi) {
                a[k] = aux[i++];
            } else if (less(aux[j], aux[i], c)) {
                a[k] = aux[j++];
                inversions += (mid - i + 1);
            } else {
                a[k] = aux[i++];
            }
        }
        return inversions;
    }

    private static long merge(int[] a, int[] aux, int lo, int mid, int hi) {
        long inversions = 0;
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
        }
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid) {
                a[k] = aux[j++];
            } else if (j > hi) {
                a[k] = aux[i++];
            } else if (aux[j] < aux[i]) {
                a[k] = aux[j++];
                inversions += (mid - i + 1);
            } else {
                a[k] = aux[i++];
            }
        }
```

Applications

Inversions.java

```
        return inversions;
    }

    private static long merge(double[] a, double[] aux, int lo, int mid, int hi) {
        long inversions = 0;
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
        }
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid) {
                a[k] = aux[j++];
            } else if (j > hi) {
                a[k] = aux[i++];
            } else if (aux[j] < aux[i]) {
                a[k] = aux[j++];
                inversions += (mid - i + 1);
            } else {
                a[k] = aux[i++];
            }
        }
        return inversions;
    }

    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

    private static boolean less(Object v, Object w, Comparator c) {
        return c.compare(v, w) < 0;
    }

    public static void main(String[] args) {
        int[] a = StdIn.readAllInts();
        StdOut.println(Inversions.count(a));
```

Applications

Inversions.java

```
    }
}
```