Sorting Applications
Outline

1 Performance Characteristics of Sorting Algorithms

2 Applications
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Stable</th>
<th>In-place</th>
<th>$T(n)$</th>
<th>$S(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>no</td>
<td>yes</td>
<td>$n^2$</td>
<td>$1$</td>
</tr>
<tr>
<td>Insertion</td>
<td>yes</td>
<td>yes</td>
<td>$n^2$</td>
<td>$1$</td>
</tr>
<tr>
<td>Shell</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$</td>
<td>$1$</td>
</tr>
<tr>
<td>Merge</td>
<td>yes</td>
<td>no</td>
<td>$n \log n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Quick</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$</td>
<td>$1$</td>
</tr>
<tr>
<td>3-way quick</td>
<td>no</td>
<td>yes</td>
<td>$nH$</td>
<td>$1$</td>
</tr>
<tr>
<td>Heap</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$</td>
<td>$1$</td>
</tr>
</tbody>
</table>
# Performance Characteristics of Sorting Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Stable?</th>
<th>In-place?</th>
<th>$T(n)$</th>
<th>$S(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>no</td>
<td>yes</td>
<td>$n^2$</td>
<td>1</td>
</tr>
<tr>
<td>Insertion</td>
<td>yes</td>
<td>yes</td>
<td>$n^2$</td>
<td>1</td>
</tr>
<tr>
<td>Shell</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$?</td>
<td>1</td>
</tr>
<tr>
<td>Merge</td>
<td>yes</td>
<td>no</td>
<td>$n \log n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Quick</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$</td>
<td>1</td>
</tr>
<tr>
<td>3-way quick</td>
<td>no</td>
<td>yes</td>
<td>$nH$</td>
<td>1</td>
</tr>
<tr>
<td>Heap</td>
<td>no</td>
<td>yes</td>
<td>$n \log n$</td>
<td>1</td>
</tr>
</tbody>
</table>
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

```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.
To sort an array 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

```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

```java
import java.util.Arrays;

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

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
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:
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
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

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
Applications

Program: Rhymer.java

- Standard input: a sequence of words
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
Applications

Program: Rhymer.java

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

```bash
~/workspace/dsaj/programs

$ java Rhymer
```
Program:  Rhymer.java

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

```bash
> ~/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

-`
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>
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
sells
$ _
```
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));
    }
}
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

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

```java
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;
}
```
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.
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

```java
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

Program: Inversions.java

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

Program: Inversions.java

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

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

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

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

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

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

Program:  Inversions.java

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

```bash
~/workspace/dsaj/programs
$ java dsa.Inversions
1 2 3 4 6 8 5 7
<ctrl-d>
```
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

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) {
            // Further code...
        } else {
            int mid = lo + (hi - lo) / 2;
            inversions += count(a, aux, lo, mid);
            inversions += count(a, aux, mid + 1, hi);
            inversions += count(a, aux, lo, mid);
            inversions += merge(a, aux, lo, mid, hi);
        }
        return inversions;
    }
    private static long merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
        int i = lo, j = mid + 1, k = lo;
        while (i <= mid && j <= hi) {
            if (aux[i] <= aux[j]) {
                a[k++] = aux[i++];
            } else {
                inversions += mid - i + 1;
                a[k++] = aux[j++];
            }
        }
        while (i <= mid) {
            a[k++] = aux[i++];
        }
        while (j <= hi) {
            a[k++] = aux[j++];
        }
        return inversions;
    }
}

Applications
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) {
        }
    }
}
```java
public class Inversions {
    public 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;
    }

    public 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;
    }

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

Applications
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];
    }
    return inversions;
}
Applications

```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++];
        }
    }
    return inversions;
}
```
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

```java
}

)`
```