Elementary Sorts

Outline

Prologue

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name	Date	Amount
Turing	6/17/1990	644.08
vonNeumann	3/26/2002	4121.85
Dijkstra	8/22/2007	2678.40
vonNeumann	1/11/1999	4409.74
Dijkstra	11/18/1995	837.42
Hoare	5/10/1993	3229.27
vonNeumann	2/12/1994	4732.35
Hoare	8/18/1992	4381.21
Turing	1/11/2002	66.10
Thompson	2/27/2000	4747.08
Turing	2/11/1991	2156.86
Hoare	8/12/2003	1025.70
vonNeumann	10/13/1993	2520.97
Dijkstra	9/10/2000	708.95
Turing	10/12/1993	3532.36
Hoare	2/10/2005	4050.20

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name ↑	Date	Amount
Dijkstra	8/22/2007	2678.40
Dijkstra	9/10/2000	708.95
Dijkstra	11/18/1995	837.42
Hoare	2/10/2005	4050.20
Hoare	8/18/1992	4381.21
Hoare	5/10/1993	3229.27
Hoare	8/12/2003	1025.70
Thompson	2/27/2000	4747.08
Turing	6/17/1990	644.08
Turing	1/11/2002	66.10
Turing	10/12/1993	3532.36
Turing	2/11/1991	2156.86
vonNeumann	3/26/2002	4121.85
vonNeumann	1/11/1999	4409.74
vonNeumann	2/12/1994	4732.35
vonNeumann	10/13/1993	2520.97

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name ↓	Date	Amount
vonNeumann	10/13/1993	2520.97
vonNeumann	1/11/1999	4409.74
vonNeumann	3/26/2002	4121.85
vonNeumann	2/12/1994	4732.35
Turing	6/17/1990	644.08
Turing	10/12/1993	3532.36
Turing	2/11/1991	2156.86
Turing	1/11/2002	66.10
Thompson	2/27/2000	4747.08
Hoare	8/12/2003	1025.70
Hoare	5/10/1993	3229.27
Hoare	8/18/1992	4381.21
Hoare	2/10/2005	4050.20
Dijkstra	11/18/1995	837.42
Dijkstra	8/22/2007	2678.40
Dijkstra	9/10/2000	708.95

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name	Date ↑	Amount
Turing	6/17/1990	644.08
Turing	2/11/1991	2156.86
Hoare	8/18/1992	4381.21
Hoare	5/10/1993	3229.27
Turing	10/12/1993	3532.36
vonNeumann	10/13/1993	2520.97
vonNeumann	2/12/1994	4732.35
Dijkstra	11/18/1995	837.42
vonNeumann	1/11/1999	4409.74
Thompson	2/27/2000	4747.08
Dijkstra	9/10/2000	708.95
Turing	1/11/2002	66.10
vonNeumann	3/26/2002	4121.85
Hoare	8/12/2003	1025.70
Hoare	2/10/2005	4050.20
Dijkstra	8/22/2007	2678.40

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name	Date ↓	Amount
Dijkstra	8/22/2007	2678.40
Hoare	2/10/2005	4050.20
Hoare	8/12/2003	1025.70
vonNeumann	3/26/2002	4121.85
Turing	1/11/2002	66.10
Dijkstra	9/10/2000	708.95
Thompson	2/27/2000	4747.08
vonNeumann	1/11/1999	4409.74
Dijkstra	11/18/1995	837.42
vonNeumann	2/12/1994	4732.35
vonNeumann	10/13/1993	2520.97
Turing	10/12/1993	3532.36
Hoare	5/10/1993	3229.27
Hoare	8/18/1992	4381.21
Turing	2/11/1991	2156.86
Turing	6/17/1990	644.08

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name	Date	Amount ↑
Turing	1/11/2002	66.10
Turing	6/17/1990	644.08
Dijkstra	9/10/2000	708.95
Dijkstra	11/18/1995	837.42
Hoare	8/12/2003	1025.70
Turing	2/11/1991	2156.86
vonNeumann	10/13/1993	2520.97
Dijkstra	8/22/2007	2678.40
Hoare	5/10/1993	3229.27
Turing	10/12/1993	3532.36
Hoare	2/10/2005	4050.20
vonNeumann	3/26/2002	4121.85
Hoare	8/18/1992	4381.21
vonNeumann	1/11/1999	4409.74
vonNeumann	2/12/1994	4732.35
Thompson	2/27/2000	4747.08

Prologue

Sorting is the process of arranging a sequence of objects in some logical order

Example

Name	Date	Amount ↓
Thompson	2/27/2000	4747.08
vonNeumann	2/12/1994	4732.35
vonNeumann	1/11/1999	4409.74
Hoare	8/18/1992	4381.21
vonNeumann	3/26/2002	4121.85
Hoare	2/10/2005	4050.20
Turing	10/12/1993	3532.36
Hoare	5/10/1993	3229.27
Dijkstra	8/22/2007	2678.40
vonNeumann	10/13/1993	2520.97
Turing	2/11/1991	2156.86
Hoare	8/12/2003	1025.70
Dijkstra	11/18/1995	837.42
Dijkstra	9/10/2000	708.95
Turing	6/17/1990	644.08
Turing	1/11/2002	66.10

Prologue

Prologue

☰ `dsa.Selection, dsa.Insertion, dsa.Shell, dsa.Merge, dsa.Quick, dsa.Quick3way, dsa.Heap`	
`static void sort(Comparable[] a)`	sorts the array `a` according to the natural order of its objects
`static void sort(Object[] a, Comparator c)`	sorts the array `a` according to the order induced by the comparator `c`
`static void sort(int[] a)`	sorts the array `a`
`static void sort(double[] a)`	sorts the array `a`

Prologue

Prologue

A library `L` that implements the sort API can sort (in ascending order) an array `a` of objects of type `T`, according to the objects' natural order, provided

Prologue

A library `L` that implements the sort API can sort (in ascending order) an array `a` of objects of type `T`, according to the objects' natural order, provided

- `T` implements the `Comparable` interface

Prologue

A library `L` that implements the sort API can sort (in ascending order) an array `a` of objects of type `T`, according to the objects' natural order, provided

- `T` implements the `Comparable` interface
- If `v` and `w` are objects of type `T`, then `v.compareTo(w)` returns an integer that is negative, zero, or positive when `v < w`, `v = w`, or `v > w`, respectively

Prologue

A library `L` that implements the sort API can sort (in ascending order) an array `a` of objects of type `T`, according to the objects' natural order, provided

- `T` implements the `Comparable` interface
- If `v` and `w` are objects of type `T`, then `v.compareTo(w)` returns an integer that is negative, zero, or positive when `v < w`, `v = w`, or `v > w`, respectively

To sort `a`, we write

```
L.sort(a);
```

Prologue

Prologue

A library `L` that implements the sort API can also sort (in ascending order) an array `a` of objects of type `T`, according to the order induced by a comparator `C`, provided

Prologue

A library `L` that implements the sort API can also sort (in ascending order) an array `a` of objects of type `T`, according to the order induced by a comparator `C`, provided

- `C` implements the `Comparator` interface

Prologue

A library `L` that implements the sort API can also sort (in ascending order) an array `a` of objects of type `T`, according to the order induced by a comparator `c`, provided

- `c` implements the `Comparator` interface
- If `v` and `w` are objects of type `T` and `c` is an object of type `C`, then `c.compare(v, w)` returns an integer that is negative, zero, or positive when `v < w`, `v = w`, or `v > w`, respectively

Prologue

A library `L` that implements the sort API can also sort (in ascending order) an array `a` of objects of type `T`, according to the order induced by a comparator `c`, provided

- `c` implements the `Comparator` interface
- If `v` and `w` are objects of type `T` and `c` is an object of type `C`, then `c.compare(v, w)` returns an integer that is negative, zero, or positive when `v < w`, `v = w`, or `v > w`, respectively

To sort `a` using a comparator object `c`, we write

```
L.sort(a, c);
```

Prologue

Prologue

The sorting algorithms we consider refer to the objects they sort only through two operations: `less()` that compares two objects and `exchange()` that exchanges them

Prologue

The sorting algorithms we consider refer to the objects they sort only through two operations: `less()` that compares two objects and `exchange()` that exchanges them

The running time $T(n)$ of a sorting algorithm is determined by counting the number of comparisons and exchanges performed, where n is the size of the input

Prologue

The sorting algorithms we consider refer to the objects they sort only through two operations: `less()` that compares two objects and `exchange()` that exchanges them

The running time $T(n)$ of a sorting algorithm is determined by counting the number of comparisons and exchanges performed, where n is the size of the input

A sorting algorithm is adaptive if $T(n) = n$ when the input is nearly sorted or has few unique objects

Prologue

The sorting algorithms we consider refer to the objects they sort only through two operations: `less()` that compares two objects and `exchange()` that exchanges them

The running time $T(n)$ of a sorting algorithm is determined by counting the number of comparisons and exchanges performed, where n is the size of the input

A sorting algorithm is adaptive if $T(n) = n$ when the input is nearly sorted or has few unique objects

A sorting algorithm is "in place" if it does not require any extra memory besides what is needed for storing the input and perhaps a function-call stack, ie, running space $S(n) = 1$

Prologue

The sorting algorithms we consider refer to the objects they sort only through two operations: `less()` that compares two objects and `exchange()` that exchanges them

The running time $T(n)$ of a sorting algorithm is determined by counting the number of comparisons and exchanges performed, where n is the size of the input

A sorting algorithm is adaptive if $T(n) = n$ when the input is nearly sorted or has few unique objects

A sorting algorithm is "in place" if it does not require any extra memory besides what is needed for storing the input and perhaps a function-call stack, ie, running space $S(n) = 1$

A sorting algorithm is stable if it preserves the relative order of equal objects, ie, if $i < j$ and $a[i] \equiv a[j]$, then $\pi(i) < \pi(j)$, where $\pi(x)$ is the position of $a[x]$ after the sort

Prologue

Prologue

Example (transactions sorted by amount)

Name	Date	Amount ↑
Turing	1/11/2002	66.10
Turing	6/17/1990	644.08
Dijkstra	9/10/2000	708.95
Dijkstra	11/18/1995	837.42
Hoare	8/12/2003	1025.70
Turing	2/11/1991	2156.86
vonNeumann	10/13/1993	2520.97
Dijkstra	8/22/2007	2678.40
Hoare	5/10/1993	3229.27
Turing	10/12/1993	3532.36
Hoare	2/10/2005	4050.20
vonNeumann	3/26/2002	4121.85
Hoare	8/18/1992	4381.21
vonNeumann	1/11/1999	4409.74
vonNeumann	2/12/1994	4732.35
Thompson	2/27/2000	4747.08

Prologue

Example (transactions sorted by amount and then by name (unstable))

Name ↑	Date	Amount ↑
Dijkstra	9/10/2000	708.95
Dijkstra	11/18/1995	837.42
Dijkstra	8/22/2007	2678.40
Hoare	8/12/2003	1025.70
Hoare	5/10/1993	3229.27
Hoare	2/10/2005	4050.20
Hoare	8/18/1992	4381.21
Thompson	2/27/2000	4747.08
Turing	6/17/1990	644.08
Turing	10/12/1993	3532.36
Turing	2/11/1991	2156.86
Turing	1/11/2002	66.10
vonNeumann	10/13/1993	2520.97
vonNeumann	1/11/1999	4409.74
vonNeumann	2/12/1994	4732.35
vonNeumann	3/26/2002	4121.85

Prologue

Example (transactions sorted by amount and then by name (stable))

Name ↑	Date	Amount ↑
Dijkstra	9/10/2000	708.95
Dijkstra	11/18/1995	837.42
Dijkstra	8/22/2007	2678.40
Hoare	8/12/2003	1025.70
Hoare	5/10/1993	3229.27
Hoare	2/10/2005	4050.20
Hoare	8/18/1992	4381.21
Thompson	2/27/2000	4747.08
Turing	1/11/2002	66.10
Turing	6/17/1990	644.08
Turing	2/11/1991	2156.86
Turing	10/12/1993	3532.36
vonNeumann	10/13/1993	2520.97
vonNeumann	3/26/2002	4121.85
vonNeumann	1/11/1999	4409.74
vonNeumann	2/12/1994	4732.35

Prologue

Prologue

An ideal sorting algorithm is one that:

Prologue

An ideal sorting algorithm is one that:

- Performs $T(n) = n \lg n$ comparisons and $T(n) = n$ exchanges in the worst case

Prologue

An ideal sorting algorithm is one that:

- Performs $T(n) = n \lg n$ comparisons and $T(n) = n$ exchanges in the worst case
- Is adaptive

Prologue

An ideal sorting algorithm is one that:

- Performs $T(n) = n \lg n$ comparisons and $T(n) = n$ exchanges in the worst case
- Is adaptive
- Is in place

Prologue

An ideal sorting algorithm is one that:

- Performs $T(n) = n \lg n$ comparisons and $T(n) = n$ exchanges in the worst case
- Is adaptive
- Is in place
- Is stable

Prologue

Prologue

Program: `XYZSort.java`

Prologue

Program: `XYZSort.java`

- Command-line input: “-” (for case-insensitive comparison) or “+” (for case-sensitive comparison) as command-line argument

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings

Prologue

Program: `XYZSort.java`

- Command-line input: “-” (for case-insensitive comparison) or “+” (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ _
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
-
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ _
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
_
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

```
$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
S o r t E x a m p l e
```

Prologue

Program: `XYZSort.java`

- Command-line input: “-” (for case-insensitive comparison) or “+” (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
S o r t E x a m p l e
_
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

```
$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
S o r t E x a m p l e
<ctrl-d>
```

Prologue

Program: `XYZSort.java`

- Command-line input: "-" (for case-insensitive comparison) or "+" (for case-sensitive comparison) as command-line argument
- Standard input: a sequence of strings
- Standard output: the strings in sorted order

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

$ java dsa.XYZSort -
S o r t E x a m p l e
<ctrl-d>
a E e l m o p r S t x
$ java dsa.XYZSort +
S o r t E x a m p l e
<ctrl-d>
E S a e l m o p r t x
$ _
```

Prologue

Prologue

XYZSort.java

```java
package dsa;

import java.util.Comparator;

import stdlib.StdIn;
import stdlib.StdOut;

public class XYZSort {
    public static void sort(Comparable[] a) {
        ...
    }

    public static void sort(Object[] a, Comparator c) {
        ...
    }

    public static void sort(int[] a) {
        ...
    }

    public static void sort(double[] a) {
        ...
    }

    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;
    }

    private static void exchange(Object[] a, int i, int j) {
        Object swap = a[i];
        a[i] = a[j];
```

Prologue

XYZSort.java

```
        a[j] = swap;
    }

    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        if (args[0].equals("-")) {
            sort(a, String.CASE_INSENSITIVE_ORDER);
        } else if (args[0].equals("+")) {
            sort(a);
        } else {
            throw new IllegalArgumentException("Illegal command line argument");
        }
        for (String s : a) {
            StdOut.print(s + " ");
        }
        StdOut.println();
    }
}
```

Selection Sort

Selection Sort

Find the smallest item in the array and exchange it with the first entry, then find the next smallest item and exchange it with the second entry, and so on

Selection Sort

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
0	6	S	O	R	T	E	X	A	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
0	6	A	O	R	T	E	X	S	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
1	4	A	O	R	T	E	X	S	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
1	4	A	E	R	T	O	X	S	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
2	10	A	E	R	T	O	X	S	M	P	L	E

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
2	10	A	E	E	T	O	X	S	M	P	L	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
3	9	A	E	E	T	O	X	S	M	P	L	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
3	9	A	E	E	L	O	X	S	M	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
4	7	A	E	E	L	O	X	S	M	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
4	7	A	E	E	L	M	X	S	O	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
5	7	A	E	E	L	M	X	S	O	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
5	7	A	E	E	L	M	O	S	X	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
6	8	A	E	E	L	M	O	S	X	P	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
6	8	A	E	E	L	M	O	P	X	S	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
7	10	A	E	E	L	M	O	P	X	S	T	R

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
7	10	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
8	8	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
8	8	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
9	9	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
9	9	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
10	10	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
10	10	A	E	E	L	M	O	P	R	S	T	X

Selection Sort

Selection Sort

Selection.java

```java
public class Selection {
    public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (less(a[j], a[min])) {
                    min = j;
                }
            }
            exchange(a, i, min);
        }
    }

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (less(a[j], a[min], c)) {
                    min = j;
                }
            }
            exchange(a, i, min);
        }
    }
}
```

Selection Sort

Selection.java

```java
public class Selection {
    public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (less(a[j], a[min])) {
                    min = j;
                }
            }
            exchange(a, i, min);
        }
    }

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (less(a[j], a[min], c)) {
                    min = j;
                }
            }
            exchange(a, i, min);
        }
    }
}
```

$T(n) = n^2$

Insertion Sort

Insertion Sort

Consider the items one at a time, inserting each into its proper place among those already considered (ie, sorted)

Insertion Sort

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
1	0	S	O	R	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
1	0	O	S	R	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
2	1	O	S	R	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
2	1	O	R	S	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
3	3	O	R	S	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
3	3	O	R	S	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
4	0	O	R	S	T	E	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
4	0	E	O	R	S	T	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
5	5	E	O	R	S	T	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
5	5	E	O	R	S	T	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
6	0	E	O	R	S	T	X	A	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
6	0	A	E	O	R	S	T	X	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
7	2	A	E	O	R	S	T	X	M	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
7	2	A	E	M	O	R	S	T	X	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
8	4	A	E	M	O	R	S	T	X	P	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
8	4	A	E	M	O	P	R	S	T	X	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
9	2	A	E	M	O	P	R	S	T	X	L	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
9	2	A	E	L	M	O	P	R	S	T	X	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
10	2	A	E	L	M	O	P	R	S	T	X	E

Insertion Sort

		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
10	2	A	E	E	L	M	O	P	R	S	T	X

Insertion Sort

Insertion Sort

Insertion.java

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

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && less(a[j], a[j - 1], c); j--) {
                exchange(a, j, j - 1);
            }
        }
    }
}
```

Insertion Sort

Insertion.java

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

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && less(a[j], a[j - 1], c); j--) {
                exchange(a, j, j - 1);
            }
        }
    }
}
```

$T(n) = n^2$

Shell Sort

Shell Sort

Rearrange the array using insertion sort such that taking every kth entry (starting anywhere) yields a k-sorted subsequence

Shell Sort

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	H	E	L	P	S	O	R	T	E	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	L	P	H	O	R	T	S	X	A	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	P	H	O	L	T	S	X	R	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	P	H	O	L	T	S	X	R	M	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	O	L	P	S	X	R	T	S	L	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	L	L	P	S	O	R	T	S	X	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	L	L	P	S	O	R	T	S	X	E

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R

Shell Sort

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

Shell Sort

Shell Sort

Shell.java

```java
public class Shell {
    public static void sort(Comparable[] a) {
        int n = a.length;
        int k = 1;
        while (k < n / 3) {
            k = 3 * k + 1;
        }
        while (k >= 1) {
            for (int i = k; i < n; i++) {
                for (int j = i; j >= k && less(a[j], a[j - k]); j -= k) {
                    exchange(a, j, j - k);
                }
            }
            k /= 3;
        }
    }

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        int k = 1;
        while (k < n / 3) {
            k = 3 * k + 1;
        }
        while (k >= 1) {
            for (int i = k; i < n; i++) {
                for (int j = i; j >= k && less(a[j], a[j - k], c); j -= k) {
                    exchange(a, j, j - k);
                }
            }
            k /= 3;
        }
    }
}
```

Shell Sort

Shell.java

```java
public class Shell {
    public static void sort(Comparable[] a) {
        int n = a.length;
        int k = 1;
        while (k < n / 3) {
            k = 3 * k + 1;
        }
        while (k >= 1) {
            for (int i = k; i < n; i++) {
                for (int j = i; j >= k && less(a[j], a[j - k]); j -= k) {
                    exchange(a, j, j - k);
                }
            }
            k /= 3;
        }
    }

    public static void sort(Object[] a, Comparator c) {
        int n = a.length;
        int k = 1;
        while (k < n / 3) {
            k = 3 * k + 1;
        }
        while (k >= 1) {
            for (int i = k; i < n; i++) {
                for (int j = i; j >= k && less(a[j], a[j - k], c); j -= k) {
                    exchange(a, j, j - k);
                }
            }
            k /= 3;
        }
    }
}
```

$T(n)$ not known (comparable to $n \log n$)