

# Outline1 Prologue2 Selection Sort

3 Insertion Sort

4 Shell Sort



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

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

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

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

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

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

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

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

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

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

Name	$Date \downarrow$	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

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

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

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

Name	Date	Amount $\downarrow$
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



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



A library L that implements the sort API can sort (in ascending order) an array a of objects of type  $\tau$ , according to the objects' natural order, provided

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

• T implements the Comparable interface

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 τ, then v.compareTo(w) returns an integer that is negative, zero, or positive when v < w, v = w, or v > w, respectively

A library  ${\tt L}$  that implements the sort API can sort (in ascending order) an array  ${\tt a}$  of objects of type  ${\tt T}$ , 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);



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

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

• c implements the comparator interface

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.COMPATE(V, W) returns an integer that is negative, zero, or positive when V < W, V = W, Or V > W, respectively

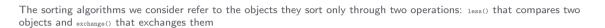
A library  $_L$  that implements the sort API can also sort (in ascending order) an array  $_a$  of objects of type  $_T$ , according to the order induced by a comparator  $_G$ , provided

- c implements the Comparator interface
- If v and w are objects of type  $\tau$  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);
```





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

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

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

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



Name Turing

Hoare

vonNeumann

vonNeumann

Thompson

# Example (transactions sorted by amount)

	-//	
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

Date

1/11/2002

8/18/1992

1/11/1999

2/12/1994

2/27/2000

Amount ↑

66.10

4381.21

4409.74

4732.35

4747.08

vonNeumann

vonNeumann

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

2/12/1994

3/26/2002

4732.35

4121.85

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





An ideal sorting algorithm is one that:

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

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

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

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



Program: XYZSort.java

Program: xyzSort.java

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

## 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

- 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

# 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

\$ \_

# 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

# \$ java dsa.XYZSort -

- 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

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

- 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

- 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

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

- 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

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

- 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 -
```

```
$ java dsa.XIZSOTT -
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 +
```

- 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

```
$ 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 +
-
```

- 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

```
$ 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
```

- 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

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

- 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

```
$ java dsa.XYZSort -
S o r t E x a m p l e
<tr1-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
<ctr1-d>
```

- 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

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

<td
```

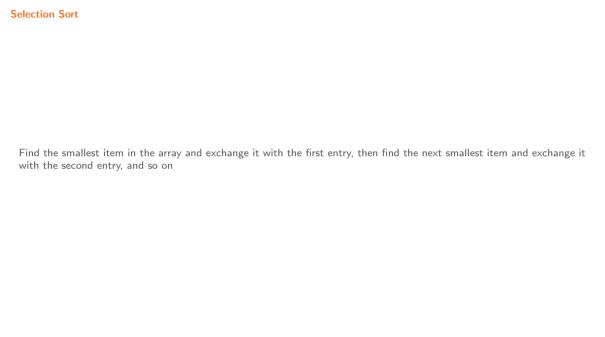


```
☑ XYZSort.java
package dsa;
import java.util.Comparator;
import stdlib.StdIn;
import stdlib.StdOut:
public class XYZSort {
    public static void sort(Comparable[] a) {
    public static void sort(Object[] a, Comparator c) {
    public static void sort(int[] a) {
    public static void sort(double[] 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[i]:
```

```
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();
}
```







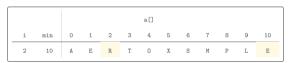
						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	0	R	Т	Е	Х	A	М	P	L	Е



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
0	6	A	0	R	Т	Е	Х	S	M	P	L	Е



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
1	4	A	Е	R	Т	0	Х	S	М	Р	L	Е



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
2	10	A	Е	Е	Т	0	Х	S	М	P	L	R



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
3	9	A	Е	Е	L	0	Х	S	М	P	Т	R



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
4	7	A	Е	Е	L	M	Х	S	0	P	Т	R



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
5	7	A	Е	Е	L	М	0	S	Х	P	Т	R



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
6	8	A	Е	Е	L	M	0	Р	Х	S	T	R



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
7	10	A	Е	Е	L	M	0	P	R	S	Т	Х

						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
8	8	A	Е	Е	L	М	0	P	R	S	Т	Х

						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
8	8	A	Е	Е	L	M	0	P	R	S	Т	Х



						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
9	9	A	Е	Е	L	M	0	P	R	S	Т	Х

						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
10	10	A	Е	Е	L	М	0	P	R	S	Т	х

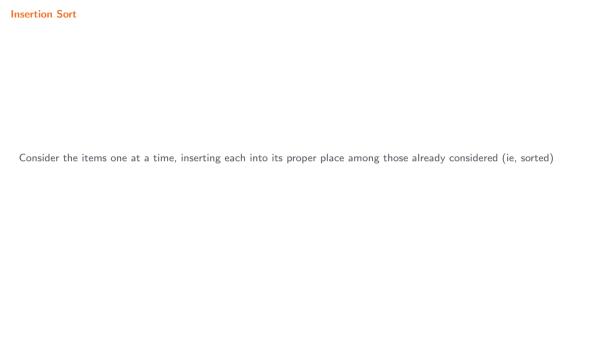
						a[]						
i	min	0	1	2	3	4	5	6	7	8	9	10
10	10	A	Е	Е	L	М	0	P	R	S	Т	х



```
☑ Selection.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.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 i = i + 1: i < n: i++) {
                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);
```







						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	0	R	Т	E	Х	A	M	P	L	Е



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
1	0	0	S	R	Т	E	х	A	M	Р	L	E

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

						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
2	1	0	R	S	Т	E	х	Α	М	Р	L	E

						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
3	3	0	R	S	Т	E	Х	A	М	P	L	E

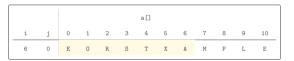
						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
3	3	0	R	S	Т	E	х	Α	M	Р	L	E



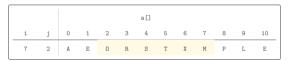
						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
4	0	Е	0	R	S	Т	Х	Α	М	Р	L	Е



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
5	5	Е	0	R	S	Т	Х	A	М	P	L	E



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
6	0	A	Е	0	R	S	Т	Х	M	Р	L	Е



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
7	2	A	Е	М	0	R	S	Т	Х	P	L	Е



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
8	4	A	Е	М	0	Р	R	S	Т	Х	L	E



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
9	2	A	E	L	М	0	Р	R	S	Т	Х	Е



						a[]						
i	j	0	1	2	3	4	5	6	7	8	9	10
10	2	A	Е	Е	L	М	0	P	R	S	Т	х



```
☑ 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):
```

```
☑ 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);
```

$$T(n) = n^2$$







k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S	Н	E	L	L	S	0	R	Т	E	Х	A	M	P	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	S	Н	Е	L	L	S	0	R	Т	E	Х	A	М	Р	L	Е

	k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	13	S	Н	Е	L	L	S	0	R	Т	E	Х	A	М	P	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	Н	Е	L	L	S	0	R	Т	E	Х	A	М	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	Н	Е	L	L	S	0	R	Т	E	Х	A	M	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	Н	Е	L	L	S	0	R	Т	E	Х	A	М	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	Н	Е	L	L	S	0	R	Т	E	Х	A	M	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13	P	Н	Е	L	L	S	0	R	Т	E	Х	A	М	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	P	Н	Е	L	L	S	0	R	Т	E	х	A	М	S	L	E

										9	10	11	12	13	14	15
4	P	Н	Е	L	L	S	0	R	Т	Е	Х	A	M	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	Р	S	0	R	Т	E	Х	A	М	S	L	E

k				6	7	8	9	10	11	12	13	14	15
				0	R	Т	Е	Х	A	M	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	Р	S	0	R	Т	E	Х	A	М	S	L	E

	0				7	8	9	10	11	12	13	14	15
					R	Т	Е	Х	Α	М	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	Р	S	0	R	Т	E	Х	A	М	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	P	S	0	R	Т	E	Х	A	М	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	Р	S	0	R	Т	E	Х	A	М	S	L	E



k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Н	Е	L	Р	S	0	R	Т	E	Х	A	М	S	L	E

						9	10	11	12	13	14	15
		L		0		Е	х	A	М	S	L	E

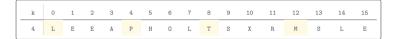
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	L	Р	Н	0	R	Т	S	Х	A	M	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	L	Р	Н	0	R	Т	S	х	A	М	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	L	Р	Н	0	R	Т	S	Х	A	M	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	L	P	Н	0	R	Т	S	Х	A	M	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	Α	Р	Н	0	L	Т	S	Х	R	М	S	L	E



k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	A	М	Н	0	L	P	S	Х	R	Т	S	L	Е

k				6	7	8	9	10	11	12	13	14	15
4	L			0	L	P	S	Х	R	Т	S	L	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	A	М	Н	0	L	P	S	Х	R	Т	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	A	M	Н	0	L	P	S	Х	R	Т	S	L	Е

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	A	M	Н	L	L	P	S	0	R	Т	S	х	Е

	k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ľ	4	L	Е	Е	A	М	Н	L	L	P	S	0	R	Т	S	Х	E

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	L	Е	Е	A	M	Н	L	E	P	S	0	L	Т	S	х	R

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	L	Е	Е	A	М	Н	L	Е	P	S	0	L	Т	S	Х	R

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	A	E	E	Е	Н	L	L	L	М	0	Р	R	s	s	Т	Х



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

```
☑ Shell.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, i, i - 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  $n \log n$ )