DSA

Module 14

Sorting

Author
Srinivas Dande

14. Sorting

- Sorting Algorithm is used to arrange elements of an array/list in an order specified.
- Following are the list of Sorting Algorithms

1) Bubble Sort
2) Selection Sort
3) Insertion Sort
4) Merge Sort
5) Quicksort
6) Heap Sort
7) Counting Sort
8) Radix Sort
9) Bucket Sort
10) Shell Sort

14.1. Complexity of Sorting Algorithms

Sorting Algorithm	Time Complexity Best	Time Complexity Worst	Time Complexity Average	Space Complexity
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(nlog n)	O(nlog n)	O(nlog n)	O(n)
Quicksort	O(nlog n)	$O(n^2)$	O(nlog n)	O(log n)
Heap Sort	O(nlog n)	O(nlog n)	O(nlog n)	O(1)
Counting Sort	O(n+k)	O(n+k)	O(n+k)	Max
Radix Sort	O(n+k)	O(n+k)	O(n+k)	Max
Bucket Sort	O(n+k)	$O(n^2)$	O(n)	O(n+k)
Shell Sort	O(nlog n)	$O(n^2)$	O(nlog n)	O(1)

14.2. Stability of Sorting Algorithms

a) Stable Sorting Algorithms
b) Unstable Sorting Algorithms

a) Stable Sorting Algorithms:

- If equal elements are not reordered as a result of the sort then that sorting algorithm is Stable.

Ex:

Bubble Sort
Insertion Sort
Merge Sort
Counting Sort
Radix Sort
Bucket Sort

b) Unstable Sorting Algorithms:

- If equal elements are reordered as a result of the sort then that sorting algorithm is Unstable.

Ex:

Selection Sort
Quicksort
Heap Sort
Shell Sort

14.3. Sorting in Java

a) Arrays.sort()
b) Collections.sort()

a) Arrays.sort():

- Arrays.sort() can used in two ways.

    1) For Array of Primitives(int,char,double etc)
    2) For Array of Objects(Integer, Student, Customer etc)

1) For Array of Primitives(int,char,double etc):

- Sorts the specified array into ascending numerical order.
- Sorting algorithm is a Dual-Pivot Quicksort
- This algorithm offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to O(n2)performance

2) For Array of Objects(Integer, Student, Customer etc)

- Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.
- All elements in the array must implement the Comparable interface.
- This sort is guaranteed to be stable: equal elements will not be reordered as a result of the sort.
- Sorting algorithm is a MergeSort
- This algorithm offers O(n log(n)) performance

- ✓ **Arrays.sort() uses QuickSort for Array of Primities**
- ✓ **Arrays.sort() uses MergeSort for Array of Objects**

b) Collections.sort()

1) For Lists (ArrayList, LinkedList etc)

- Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.
- All elements in the array must implement the Comparable interface.
- This sort is guaranteed to be stable: equal elements will not be reordered as a result of the sort.
- Sorting algorithm is a MergeSort
- This algorithm offers O(n log(n)) performance

✓ **Collections.sort() uses MergeSort for Collection of Objects**

14.4. Labs on Arrays.sort()

14.4.1. Sort the Array

Lab1.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab1 {

	public static void main(String[] args) {

		int arr1[] = {10,5,20,15,30,25};
		char arr2[] = {'C','D','E','B','A'};

		Arrays.sort(arr1);
		System.out.println(Arrays.toString(arr1));

		Arrays.sort(arr2);
		System.out.println(Arrays.toString(arr2));

	}
}
```

14.4.2. Sort the Sub-Array

Lab2.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab2 {

	public static void main(String[] args) {

		int arr1[] = {10,5,20,15,30,25};
		char arr2[] = {'C','D','E','B','A'};

		Arrays.sort(arr1,2,6);
		System.out.println(Arrays.toString(arr1));

		Arrays.sort(arr2,3,5);
		System.out.println(Arrays.toString(arr2));

	}
}
```

14.4.3. Sort Array in ASC and DESC Order

Lab3.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;
import java.util.Collections;

/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab3 {

	public static void main(String[] args) {

		Integer arr[] = {10,5,20,15,30,25};

		Arrays.sort(arr);
		System.out.println(Arrays.toString(arr));

		Arrays.sort(arr,Collections.reverseOrder());
		System.out.println(Arrays.toString(arr));
	}
}
```

14.4.4. Sort Array in ASC and DESC Order

Lab4.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;
import java.util.Comparator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

class MyIntegerComparator implements Comparator<Integer>{

	@Override
	public int compare(Integer a, Integer b) {
		return b-a;
	}
}

public class Lab4{

	public static void main(String[] args) {

		Integer arr[] = {10,5,20,15,30,25};

		Arrays.sort(arr);
		System.out.println(Arrays.toString(arr));

		Arrays.sort(arr,new MyIntegerComparator());
		System.out.println(Arrays.toString(arr));
	}
}
```

14.4.5. Arrange as Odd First and Even Next

Lab5.java

```
package com.jlcindia.arrays.sort;
import java.util.Arrays;
import java.util.Comparator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class MyEvenComparator implements Comparator<Integer>{
	@Override
	public int compare(Integer a, Integer b) {
		return a%2-b%2;
	}
}

class MyOddComparator implements Comparator<Integer>{
	@Override
	public int compare(Integer a, Integer b) {
		return b%2-a%2;
	}
}

public class Lab5{
	public static void main(String[] args) {

		Integer arr[] = {10,5,20,15,30,25};

		Arrays.sort(arr,new MyEvenComparator());
		System.out.println(Arrays.toString(arr));

		Arrays.sort(arr,new MyOddComparator());
		System.out.println(Arrays.toString(arr));
	}
}
```

14.4.6. Using Comparable

Lab6.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class Customer implements Comparable<Customer>{

	int cid;
	String cname;
	long phone;

	public Customer() {}

	public Customer(int cid, String cname, long phone) {
		super();
		this.cid = cid;
		this.cname = cname;
		this.phone = phone;
	}

	@Override
	public String toString() {
		return "[" + cid + ", " + cname + ", " + phone + "]";
	}

	@Override
	public int compareTo(Customer cust) {
		return this.cid-cust.cid;
	}
}
```

```
public class Lab6{

	public static void main(String[] args) {

		Customer cust1 = new Customer(102,"sd",222);
		Customer cust2 = new Customer(103,"ds",333);
		Customer cust3 = new Customer(101,"sri",111);

		Customer customers[] = {cust1,cust2,cust3};

		Arrays.sort(customers);

		for(Customer cust:customers) {
			System.out.println(cust);
		}

	}
}
```

14.4.7. Using Comparator

Lab7.java

```
package com.jlcindia.arrays.sort;

import java.util.Arrays;
import java.util.Comparator;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

class MyMarksComparator implements Comparator<Student>{

	@Override
	public int compare(Student stu1, Student stu2) {
		return (int) (stu1.marks-stu2.marks);
	}
}

class Student implements Comparable<Student>{

	int sid;
	String sname;
	double marks;

	public Student() {}

	public Student(int sid, String sname, double marks) {
		super();
		this.sid = sid;
		this.sname = sname;
		this.marks = marks;
	}
```

```
	@Override
	public String toString() {
		return "[" + sid + ", " + sname + ", " + marks + "]";
	}

	@Override
	public int compareTo(Student stu) {
		return this.sid-stu.sid;
	}

}
public class Lab7{

	public static void main(String[] args) {

		Student stu1 = new Student(102,"sd",70);
		Student stu2 = new Student(103,"ds",60);
		Student stu3 = new Student(101,"sri",50);

		Student students[] = {stu1,stu2,stu3};

		Arrays.sort(students);

		for(Student stu:students) {
			System.out.println(stu);
		}

		Arrays.sort(students,new MyMarksComparator());

		for(Student stu:students) {
		  System.out.println(stu);
		 }

	}
}
```

14.5. Labs on Collections.sort()

14.5.1. Sort the List in ASC and DESC

Lab8.java

```
package com.jlcindia.collections.sort;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab8 {
	public static void main(String[] args) {

		List<Integer> mylist = new ArrayList<>();
		mylist.add(10);
		mylist.add(5);
		mylist.add(20);
		mylist.add(25);
		mylist.add(15);

		Collections.sort(mylist);
		System.out.println(mylist);

		Collections.sort(mylist,Collections.reverseOrder());
		System.out.println(mylist);

	}
}
```

14.5.2. Using Comparable

Lab9.java

```
package com.jlcindia.collections.sort;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class Customer implements Comparable<Customer>{
	int cid;
	String cname;
	long phone;

	public Customer() {}
	public Customer(int cid, String cname, long phone) {
		super();
		this.cid = cid;
		this.cname = cname;
		this.phone = phone;
	}

	@Override
	public String toString() {
		return "[" + cid + ", " + cname + ", " + phone + "]";
	}

	@Override
	public int compareTo(Customer cust) {
		return this.cid-cust.cid;
	}
}
```

```
public class Lab9{

	public static void main(String[] args) {

		Customer cust1 = new Customer(102,"sd",222);
		Customer cust2 = new Customer(103,"ds",333);
		Customer cust3 = new Customer(101,"sri",111);

		List<Customer> customers = new ArrayList<>();
		customers.add(cust1);
		customers.add(cust2);
		customers.add(cust3);

		Collections.sort(customers);

		for(Customer cust:customers) {
			System.out.println(cust);
		}

	}
}
```

14.5.3. Using Comparator

Lab10.java

```
package com.jlcindia.collections.sort;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
class MyMarksComparator implements Comparator<Student> {

	@Override
	public int compare(Student stu1, Student stu2) {
		return (int) (stu1.marks - stu2.marks);
	}
}

class Student implements Comparable<Student> {

	int sid;
	String sname;
	double marks;

	public Student() {   }
	public Student(int sid, String sname, double marks) {
		super();
		this.sid = sid;
		this.sname = sname;
		this.marks = marks;
	}
```

```
	@Override
	public String toString() {
		return "[" + sid + ", " + sname + ", " + marks + "]";
	}

	@Override
	public int compareTo(Student stu) {
		return this.sid - stu.sid;
	}
}
public class Lab10 {
	public static void main(String[] args) {

		Student stu1 = new Student(102, "sd", 70);
		Student stu2 = new Student(103, "ds", 60);
		Student stu3 = new Student(101, "sri", 50);

		List<Student> students = new ArrayList<>();
		students.add(stu1);
		students.add(stu2);
		students.add(stu3);

		Collections.sort(students);

		for (Student stu : students) {
			System.out.println(stu);
		}
		System.out.println("----------------------");

		Collections.sort(students, new MyMarksComparator());

		for (Student stu : students) {
			System.out.println(stu);
		}
	}
}
```

14.6. Exploring Bubble Sort

- Bubble sort is the simplest sorting algorithm.
- It works by iterating the input array from the first element to the last, comparing each pair of elements and swapping them if needed.
- Bubble sort continues its iterations until no more swaps are needed.
- **The only significant advantage that bubble sort has over other implementations is that it can detect whether the input list is already sorted or not.**

Complexity of Bubble Sort:

Time Complexity - Best	O(n)
Time Complexity - Worst	$O(n^2)$
Time Complexity - Average	$O(n^2)$
Space Complexity	O(1)

14.6.1. Bubble Sort Implementation

Lab11.java

```
package com.jlcindia.sorting;

import java.util.Arrays;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab11 {
        public static void bubbleSort(int arr[]) {

                int n= arr.length;
```

```
		for(int i=0;i<n;i++) {
			for(int j=0;j<n-i-1;j++) {
				if(arr[j]>arr[j+1]) {
					int temp = arr[j];
					arr[j]=arr[j+1];
					arr[j+1]=temp;
				}
			}
		}

	public static void main(String[] args) {
		int arr[] = {5,9,15,8,20,7,10};
		System.out.println(Arrays.toString(arr));

		bubbleSort(arr);
		System.out.println(Arrays.toString(arr));
	}
}
```

14.6.2. Optimized Bubble Sort

Lab12.java

```
package com.jlcindia.sorting;

import java.util.Arrays;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

public class Lab12 {

	public static void bubbleSort(int arr[]) {

		int n= arr.length;
```

```
        for(int i=0;i<n;i++) {

            boolean swaped=false;

            for(int j=0;j<n-i-1;j++) {
                if(arr[j]>arr[j+1]) {
                    int temp = arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                    swaped=true;
                }
            }

            if(swaped==false) {
                break;
            }
        }
    }

    public static void main(String[] args) {

        int arr[] = {5,9,15,8,20,7,10};
        System.out.println(Arrays.toString(arr));

        bubbleSort(arr);
        System.out.println(Arrays.toString(arr));

    }
}
```