Chapter 1

R-1.11:

```
/*  Java Class: Flower
    Author: Zhenyu Jia
    Class: CSCI 240
    Date: 9/21/2022
    Description: R-1.11 from homework1
    I certify that the code below is my own work.
Exception(s): N/A
*/
public class Flower{  //class name: Flower
    private String name;
    private int number;
    private float price;

    Flower(String s, int n, float p){
        name = s;
        number = n;
        price = p;
    }

    public void setName(String s) { name = s; }
    public void setNumber(int n) { number = n; }
    public void setPrice(float p) { price = p; }

    public String getName() { return name; }
    public int getNumber() { return number; }
    public float getPrice() { return price; }
}
```

R-1.20:

```
public static int sum(int n){
        if(n == 1)
            return 1;
        return n + sum(n-1);
    }
```

C-1.5:

Pseudocode:

```
        int[] arr = new int[52];
        for(int i=0; i<52; i++){
                random r = random(52);          //random number from 1-52
                if(arr[r] == 0)
                        arr[r] = set[i];
                else
                        repeat the random process until arr[r] == 0;
```

```
        }
        return arr;
```

C-1.6:

```
/*  Java Class: Exercise
   Author: Zhenyu Jia
   Class: CSCI 240
   Date: 9/23/2022
   Description: C-1.6 from homework1
   I certify that the code below is my own work.
Exception(s): N/A
*/
import java.util.*;

public class Exercise {
   final String target = "abcdef";
   List<String> list = new ArrayList<String>();

   public void allPossibleString() {
      for (int i = 0; i < target.length(); i++) {
         String s = Character.toString(target.charAt(i));
         String str = target;
         helper(s, str, "");
      }
      list.stream().forEach(System.out::println);
   }

   public void helper(String s, String str, String res) {
      res += s;
      if (res.length() == target.length()) {
         list.add(res);
         return;
      }
      String sstr = str.replace(s, "");
      for (int i = 0; i < sstr.length(); i++) {
         String ss = Character.toString(sstr.charAt(i));
         helper(ss, sstr, res);
      }
   }

}
```

Chapter 2:

R-2.1: The compiler will take longer to determine the methods of each subclass; Also it takes longer every time the constructors use super method to initialize itself.

R-2.2: When the father class is changed or updated, it takes longer time to also update the child classes if there are many of them.

C-2.4:

```
/*  Java Class: Line
   Author: Zhenyu Jia
   Class: CSCI 240
   Date: 9/23/2022
   Description: C-2.4 from homework1
   I certify that the code below is my own work.
Exception(s): N/A
*/
public class Line {
   private double a;
   private double b;

   Line(int a1, int b1){
      a = a1;
      b = b1;
   }
   public double getA(){ return a; }
   public double getB() { return b; }

   public double intersect(Line line) throws Exception {
      if(a == line.getA())
         throw new Exception("Parallel");
      else{
         return (line.getB()-b)/(a-line.getA());
      }
   }
}
```

Chapter 3:

R-3.6: After the list is formed, the variable size(int) will be created. The size will be plus or minus with the function add() or remove(). The function size() will return the variable size at the complexity of O(1).

R-3.11:

```
int n=0, int max = Integer.MinValue;
public int max(int[] array, int n, int max){
        if(n == array.length)
                return max;
        if(array[n] > max)
                max = array[0]
        return max(array, ++n, max);
```

time complexity: O(n)

space complexity: O(1)

C-3.5:
We can create an array where each element is an Integer list with a size n.
List<List<Integer>>array = new ArrayList<>(n);
everytime calls meet(i,j), we set array[i].add(j) and array[j].add(i)
when array[x].size() == n, it indicates that x has won.

C-3.10:
If the two themselves are adjacent, a total of three edges (that is, three next relationships) need to be modified {a node} -> {v = v1} -> {v = v2} -> {a node}
If the two are not adjacent, a total of four edges {a node} -> {v = v1} -> {some nodes} -> {v = v2} -> {a node} need to be modified (assuming v1 is in v2 forward)
It needs to use two while loops to individually find the two precious nodes of x and y in a singly linked list, after knowing the two precious nodes and two after nodes, it is able to swap the two nodes.
What's more, it doesn't need any loop to find the previous nodes in a doubly linked list, therefore it takes less time to swap two nodes in a doubly linked list.

Chapter 4:
R-4.7:
A=8nlogn
B=2n^2
Simplify: 4logn = n
When n=16, 8nlogn = 2n^2
Therefore, when n0 = 16, A is faster than B for n >= n0

R-4.13:
4n nlogn 3n+100logn 4nlogn + 2n n^2+10n n^3 2^10 2^logn

R-4.20:
O(n^3)

C-4.6:
```
 public void sortByK(int[] array, int k) {
       sortHelper(0, array.length - 1, array, k);
       System.out.println(Arrays.toString(array));
   }

   public void sortHelper(int low, int high, int[] array, int k) {
       while (low < high) {
           while (low < high && array[high] > k) --high;
           while (low < high && array[low] <= k) ++low;
           swap(array, low, high);
       }
   }
```

```
public void swap(int[] array, int x, int y) {
      int temp = array[x];
      array[x] = array[y];
      array[y] = temp;
   }
```
The running time would be O(n)

Chapter 5:
R-5.5: 3, 8, 2, 1, 6, 7, 4, 9, 5

R-5.9: 5, 3, 2, 8, 9, 1, 7, 6, 4

R-5.11
D:1,2,3,4,5,6,7,8 Q:null

1.Q.add:(D.pollFirst(), D.pollFirst(), D.pollFirst(), D.pollLast(), D.pollLast(), D.pollLast(), D.pollLast(), D.pollFirst())

D:null Q:4,5,6,7,8,3,2,1

2.D.offerFirst(Q.poll()), D,offerLast(Q.poll()), D.offerLast(Q.poll()), D.offerFirst(Q.poll()), D.offerFirst(Q.poll()), D.offerFirst(Q.poll()), D.offerLast(Q.poll()), D.offerLast(Q.poll())

D:6,7,8,1,2,3,5,4 Q:null

3.Q.add: D.pollFirst(), D.pollFirst(), D.pollFirst()

D: 1,2,3,5,4 Q:8,7,6

4.D.pollLast(), D.pollLast(), D.pollLast()

D:1,2,3,5,4,6,7,8 Q:null

C-5.2:
```
 public boolean SandQ(Stack S, Queue Q, int E) {
      boolean flag = false;
      while (!S.isEmpty()) {
         if (S.peek().equals(E))
            flag = true;
         Q.offer(S.pop());
      }
      while (!Q.isEmpty())
         S.push(Q.poll());
      while (!S.isEmpty())
         Q.offer(S.pop());
```

```
        while (!Q.isEmpty())
            S.push(Q.poll());
        return flag;
    }
```

C-5.5:
Given two queues, Q1 and Q2
Step 1: push the value into Q1
Step 2: push the second value into Q2, then pop the value in Q1 to Q2 in order until Q1 is empty
Step 3: push the third value into Q1, then pop the value in Q2 to Q1 in order until Q2 is empty
Step 4: Repeat step 2 and step 3
The running time of push() would be O(n), and pop() would O(1)

Chapter: 6:
R-6.3:

```
public void RotateArray(int[] arr, int d) {
        int temp[] = new int[arr.length];
        for (int i = d, k = 0; i < arr.length; i++, k++)
            temp[k] = arr[i];
        for (int i = 0; i < arr.length; i++)
            arr[i] = temp[i];
    }
```

R-6.9:
R-6.16:
The minimum number would be zero. Because in this situation every element is accessed k times, therefore no elements have been accessed less than k times.
The maximum number would be n-1. In this situation only one element have been accessed kn times, therefore the rest of elements have been accessed 0 times.

C-6.10:
a: 1,3,5,7,9,8,6,4,2,0
b: 1, 3, 5, 7, …,n-5, n-3, n-1, n-2, n-4, n-6, …, 4, 2, 0

C.6,13:

```
public void shuffle(int[] arr){
        for(int i=arr.length-1;i>=0;i--){
            int randomIndex = randomInteger(i); //randomInteger(n) return random int < n
            int temp = arr[randomIndex];
            arr[randomIndex] = arr[i];
            arr[i] = temp;
        }
    }
```