Notice

- You are not allowed to modify the signatures of the methods you are asked to implement. You are free to add new methods and variables to the class.
- For each question, write the following information as a comment to the method:
  1) all the Internet resources you've used for solving the problem (even though you didn't explicitly copy from them);
  2) all the students with whom you have discussed this question; and
  3) running time of your method, in the Big-Oh notation.[1]
- Make your algorithms as efficient as possible. Your scores depend on their efficiency.
- No marks shall be awarded if your submission does not compile.[2]
- Submission procedure (each deviation will result in deduction of 10 points):
  1) Name the .java file as `<class_name>_<your_id>_<your_name>.java`, e.g., `Sorting_12345678d_TangTszkei.java` if your name is Tang Ketosis and your ID is 12345678d. (You may find Alt-Shift-R helpful if you're using Eclipse.)
  2) Put all your files into a folder with name `A1_<your_id>_<your_name>`.
  3) Create a .zip or .jar file to contain this folder, similar as the distributed file. WARNING: ONLY ZIP AND JAR ARE ACCEPTED.

Plagiarism checker: Ke Yuping (yuping.ke@connect.polyu.hk)

[1] A frequent question is "Whether I can use methods from Java library?". Most library methods are complicated and difficult to analyze. If you use them, you have to count their steps, which is almost impossible from my experience.

[2] If you've problems, this and this web pages may help, and you're welcome to seek help on the discussion forum (DON'T SHARE YOUR CODES).

1. (35 points) Grader: Dai Xuelong (20102284r@connect.polyu.hk)

   In the ith major iteration of insertion sort, we need to puts element $a[i]$ to its correct position. For this we need to find from the sorted part the position of the first element that is greater than $a[i]$, or i if there does exist such an element. Since $a[0], a[1], \ldots, a[i-1]$ have been sorted, it makes sense to use the idea of binary search for this purpose.

   The search step is however not exactly the same as binary search. First, we don't know the specific "key" to search. Second, if the value of $a[i]$ appears in $a[0], a[1], \ldots, a[i-1]$, then we have to return the last of them. For example, you can consider the last iteration of the following examples:

   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{0};$$
   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{1};$$
   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{2};$$
   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{4};$$
   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{5}.$$
   $$1, 2, 2, 4, 4, 4, 7, 8, 8, \mathbf{9}.$$

   In particular, the position should be 6 when the last element is 4. A naïve idea is to search one by one after finding a 4, but then the search step takes $O(n)$ time in the worst case (e.g., when all the elements have the same value).

   Implement insertion sort in `Sorting.java` such that the search step uses $O(\log n)$ time. A class *Element* is provided to facilitate your testing. See `Sorting.java` for details.

   ```
   public static void insertionSort(Element[] a) {}
   ```

2. (35 points) Grader: Liu Qijiong (21037042r@connect.polyu.hk)

   We are playing cards with a standard 52-card deck, where A is the largest of a suit, and 2 is the smallest. You have been delivered a hand of n cards, and you have sorted them in the suit-first order: spades, hearts, clubs, and diamonds, each suit in decreasing order.

   Write an algorithm to reorder your hand of cards into rank-first order: for cards of the same rank, you follow the order of spade, heart, club, and then diamond. An example can be found in `CardGame.java`

   ```
   public static void reorder(Card[] hand) {}
   ```

   Five bonus points for also implementing `reorderBack`, from rank-first to suit-first.

   ```
   public static void reorderBack(Card[] hand) {
   ```

3. (30 points) Grader: Mu Feiteng (20033801r@connect.polyu.hk)

   The input is a circularly and doubly linked list L of $n = 3k$ nodes. The task is to split L into three circularly and doubly linked lists of equal length, containing the first k nodes, the middle k nodes, and the last k nodes of L. Implement the following method in `CDList.java`.

   ```
   public CDList[] split() {}
   ```

   For example, if the list is

   $$11 \to 12 \to 13 \to 55 \to 52 \to 58 \to 29 \to 26 \to 20 \uparrow,$$

   then the three resulting lists are

   $$11 \to 12 \to 13 \uparrow; \quad 55 \to 52 \to 58 \uparrow; \quad 29 \to 26 \to 20 \uparrow .$$

   Five bonus points for handling the general case where n may or may not be a multiple of 3. The length of each resulting list is either $\lfloor \frac{n}{3} \rfloor$ or $\lceil \frac{n}{3} \rceil$.

   Three bonus points for handling the case where instead of three, the number d of parts is a parameter.

   ```
   public CDList[] split(int d) {}
   ```