MATH 182: Homework #3

Professor Wotao Yin

Eric Chuu
Assignment: 2.1 # 4, 9, 13, 14, 21
UID: 604406828

February 3, 2017

Problem 2.1.4

We use insertion sort to sort the array: E A S Y Q U E S T I O N
The results are shown in the table below. Elements to the right of the index are not accessed during that iteration of insertion sort, so we leave those entries blank. The sorted answer is in the last row.

i	**j**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
		E	A	S	Y	Q	U	E	S	T	I	O	N
1	**0**	A	E										
2	**2**	A	E	S									
3	**3**	A	E	S	Y								
4	**2**	A	E	Q	S	Y							
5	**4**	A	E	Q	S	U	Y						
6	**2**	A	E	E	Q	S	U	Y					
7	**5**	A	E	E	Q	S	S	U	Y				
8	**6**	A	E	E	Q	S	S	T	U	Y			
9	**3**	A	E	E	I	Q	S	S	T	U	Y		
10	4	A	E	E	I	O	Q	S	S	T	U	Y	
11	4	A	E	E	I	N	O	Q	S	S	T	U	Y

□

Problem 2.1.9

We use shellsort to sort the following array: E A S Y S H E L L S O R T Q U E S T I O N
The results at the end of each h-sort are shown below are shown in the table below. The sorted answer is in the last row.

Input	E A S Y S H E L L S O R T Q U E S T I O N
13 - sort	E A S Y S H E L L S O R T Q U E S T I O N
4 - sort	E A E E L H I L M Q O O S S S R S T U Y T
1 - sort	A E E E H I L L N O O Q R S S S S T T U Y

□

Problem 2.1.13

Deck Sort. Explain how you would put a deck of cards in order by suit (spades, hearts, clubs, diamonds) and by rank within each suit (A, 2, ..., K), with the restriction that the cards must be laid out face down in a row, and the only allowed operations are to check the values of two cards and to exchange two cards (keeping them face down).

Solution
Since we are allowed to check the values of two cards, we can develop an ordering of the cards. In the context of `Java`, we would implement the `Comparable` interface. The ordering/hierarchy would define all spades cards to be less than all hearts cards, which are less than clubs, which are less than diamonds. Thus the following inequality would hold:

$$2\diamondsuit > Q\clubsuit > 10\clubsuit > A\heartsuit > 4\spadesuit > A\spadesuit$$

With this order defined, we can then apply the shellsort algorithm which only requires seeing the values of two cards at a time and exchanging two cards to get them in the correct position. Iterating this process would eventually give us a sorted deck in the order required. □

Problem 2.1.14

Dequeue Sort. Explain how you would sort a deck of cards, with the restriction that the only allowed operations are to look at values of the top two cards, to exchange the top two cards, and to move the top card to the bottom of the deck.

Solution
With the above restrictions, we consider the following algorithm to sort the deck. Let the deck have n cards. Then we look at the top two cards and swap them so that the larger one is on top. The top card is then moved to the bottom of the desk. If we repeat this process of swapping and moving to the bottom $n-1$ times, then the smallest card of the deck will be on top. We then move this card to the bottom of the deck. We can then repeat the same process $n-2$ times and find the second smallest card in the deck at the top of the deck, followed directly by the smallest card. From here, we swap the two to get the smallest card on top, and we move the smallest card to the bottom of the deck. Then we move the second to smallest card to the bottom of the deck. We continue this process until the final card on top is the largest card, followed by the remaining cards in ascending order. Then we can just move the top card to the back to get the deck in ascending order. A four card example is given below. Consider the left side to be top of the stack and the right side to be the bottom of the stack. Each column corresponds to the k-iteration in which the k-th smallest card is found.

9564	6549	6459	**4569**
5649	5496	4659	
6549	4596	6594	
5496	5964	5694	
4965	96**45**	6945	
9654		9**456**	

□

Problem 2.1.21

The code for the implementation of the `Transaction` class is given below. The `main` routine performs a simple test that compares two transaction amounts. The output is shown below:

```
 >> java Transaction
Implementation of Transaction class
Transaction 1 was larger with:  100.43
```

```
import java.util.*;

public class Transaction implements Comparable<Transaction> {

   private final double amount;

   public Transaction(double amt) {
      this.amount = amt;
   }

   public double getAmount() {
      return this.amount;
   }

   public int compareTo(Transaction other) {
      if (this.amount < other.amount) {
         return -1;
      }
      if (this.amount > other.amount) {
         return 1;
      }
      return 0;
   }

   public static void main(String[] args) {

      System.out.println("Implementation of Transaction class");

      double amt1 = 100.43;
      double amt2 = 99.733;

      Transaction t1 = new Transaction(amt1);
      Transaction t2 = new Transaction(amt2);

      if (t1.compareTo(t2) > 0) {
         System.out.println("Transaction 1 was larger with: " + t1.amount);
      } else if (t1.compareTo(t2) < 0) {
         System.out.println("Transaction 2 was larger with: " + t2.amount);
      } else {
         System.out.println("The transactions were equal.");
      }
   }
}
```
