COMP 1020
Lab 11

MATERIAL COVERED

- Simple sorting – the Selection Sort

Notes:

- The three exercises in this lab are independent – they can be done in any order. However, the Gold exercise is very similar to the Bronze exercise, and it would be best to do the Bronze exercise first.
- Only one of the three exercises is required, but try to do as many as you can.

Selection Sort to print an ArrayList

1. Start with the file **`TemplateLab11Bronze.java`**.
2. Complete the **`printInOrder`** method at the end of the file. This method will accept an **`ArrayList<Integer> data`** as its parameter. It will *not* sort this list. Instead, it will use a modified selection sort algorithm to simply *print* the list in ascending order. The list should be printed on a single line. As each number is printed, it can be deleted from **`data`**, so that when the method is finished, the list **`data`** will be an empty list.
3. The output from the program should look like this. (The data will be random.)

```
Original list:
[27, 44, 22, 20, 41, 29, 10, 2, 33, 7, 16, 26, 13, 44, 15, 15, 23, 11, 42, 13]
Print the list in ascending order:
2 7 10 11 13 13 15 15 16 20 22 23 26 27 29 33 41 42 44 44
Original list (should be empty):
[]
```

Selection Sort on a Linked List

1. Start with the files **`TestLab11Silver.java`**, **`IntLL-template.java`**, and **`Node.java`**. Immediately rename the second file **`IntLL.java`**. This file implements a linked list of **`int`** data. A constructor and a few other suitable methods are already defined for you.
2. Complete the **`void selSort()`** method in the **`IntLL`** class, so that it will perform a selection sort algorithm to re-arrange the **`Node`**s in the linked list so that the data is in ascending order.
3. Half of the algorithm has been done for you – the **`Node largestPrev()`** method will find the **`Node`** in the list that contains the largest data item. (Since it's easiest to add new nodes to the front of a linked list, you need the largest one each time, not the smallest one, if you want to get it into ascending order.) It will not return a reference to that **`Node`** – it will return a reference to the *previous* **`Node`**, since that is what you will need to remove it from the original unsorted list.

4. You should not create any new nodes at all. The existing nodes must be re-linked into a new order, but nothing should be created or destroyed.
5. When you run **`TestLab11Silver`**, the output should be as shown below. (The data is random.)

```
Original list:
<< 41  34  29  40  30  33  17  16  30  16  37  49  3  8  34  28  49  38  7  44 >>
Sorted list:
<< 3  7  8  16  16  17  28  29  30  30  33  34  34  37  38  40  41  44  49  49 >>
```

Non-destructive version of the Bronze exercise

1. Begin with the file **`TemplateLab11Gold.java`**.
2. This is exactly the same as the Bronze exercise: Complete the **`printInOrder`** method at the end of the file. This method will accept an **`ArrayList<Integer> data`** as its parameter. It will *not* sort this list. Instead, it will use a modified selection sort algorithm to simply *print* the list in ascending order. The list should be printed on a single line.
3. Here's the tricky part. This time, the list *may not* be changed at any time. You must leave the original list **`data`** completely untouched (you can use **`.get()`** and **`.size()`** but nothing else). You also must not use any other array or list of any type (no arrays, no linked lists, no **`ArrayList`**s – nothing but simple **`int`** variables). Good luck.
4. The output from the program should look like this. (The data will be random – there might be duplicate values in the list.)

```
Original list:
[29, 8, 25, 19, 3, 23, 38, 15, 36, 19, 4, 14, 3, 20, 23, 10, 4, 8, 27, 41]
Print the list in ascending order:
3 3 4 4 8 8 10 14 15 19 19 20 23 23 25 27 29 36 38 41
Original list (should be unchanged):
[29, 8, 25, 19, 3, 23, 38, 15, 36, 19, 4, 14, 3, 20, 23, 10, 4, 8, 27, 41]
```