Shuni Li
Comp221 HW0
Professor Shilad Sen
Feb 3, 2016

1. (15 pts) Design an algorithm *SecondLargest* that, given an array of n numbers, A, finds the second largest number in the array. Write out the algorithm in clear pseudocode like the book's style, and give a step-by-step example that illustrates how your algorithm works on a small input. You can assume that there are no duplicate numbers in the array and it has a size of at least two.

```
Algorithm SecondLargest(A[0..n - 1])
//Input: Array A[0..n − 1] of n non-duplicate numbers
//Output: result, the second largest number in the array

1.    if A[0] > A[1] then
2.        Largest = A[0]
3.        secondLargest = A[1]
4.    else
5.        Largest = A[1]
6.        secondLargest = A[0]
7.    for i=2 to n - 1 do
8.        if A[i] > Largest then
9.            secondLargest = Largest
10.           Largest = A[i]
11.       else if A[i] > secondLargest && A[i] < Largest then
12.           secondLargest = A[i]
13.   return secondLargest
```

Let the array, A, be [2, 4, 6, 3, 5].

First, we compare A[0], which is 2, with A[1], which 4. Since 4 is greater than 2 (A[1] > A[0]), we set the variable, Largest, to 4, and set the variable, secondLargest, to 2.

Then we enter the loop, comparing each element in the array with these two variables. The next number A[2] is 6, which is greater than the current Largest variable, 4. So we set secondLargest to the current Largest variable, 4, and set the current Largest variable to 6.

A[3] equals 3, which is smaller than both the Largest and secondLargest varibles. It doesn't satisfy any conditionals, so we move to the next element.

A[4] is 5, which is greater than 4 and smaller than 6. Thus we set secondLargest to 5. We have looped over all elements in the range, so we exit the loop. And 5 is returned as the second largest number.

2. (10 pts) Complete exercise 6 in section 1.2. Use pseucode and not English. Describe the algorithm used by your favorite ATM machine in dispensing cash.

```
Algorithm DispenseCash(Account acc)
//Input: Account object including account number, correct pin and balance
        data

1.  ask for pin
2.  if pinEntered == acc.getCorrectPin() then
4.          while true
5.                  show withdraw screen
6.                  ask for the amount needed
7.                  if amountEntered <= acc.getBalance() then
8.                          acc.balance = acc.getBalance() - amountEntered
9.                          dispense money
10.                         print “Do you want to withdraw more money?”
11.                         if input is no then
12.                                 break
13.                 else
14.                         print “You do not have enough money.”
15.                         print “Do you want to try again?”
16.                         if input is no then
17.                                 break
18.  else
19.         print “Wrong pin!”
20. print “Bye!”
```

3. (10 pts) Look at the algorithm attached to question 1 in section 1.3, and rewritten below (Note that the form below is an acceptable way of giving pseudocode in a Google Doc). Show the result of running this algorithm on the list: **[35, 32, 11, 45, 20]** Give a series of diagrams that show the Count array and the output array S, and show the changes to each array step by step.

```
Algorithm ComparisonCountingSort(A[0..n - 1])
1.      for i = 0 to n - 1 do
2.          Count[i] = 0
3.      for i = 0 to n - 2 do
4.          for j = i + 1 to n - 1 do
5.              if A[i] < A[j] then
6.                  Count[j] = Count[j] + 1
7.              else
8.                  Count[i] = Count[i] + 1
9.      for i = 0 to n - 1 do
10.         pos = Count[i]
11.         S[pos] = A[i]
12.     return S
```

The resulted array, S, is [11, 20, 32, 35, 45].

The following diagram is for the change of array *Count*:

The original Count array:

0	0	0	0	0

→ Since 35>32, Count[0] increases

1	0	0	0	0

→ Since 35>11, Count[0] increases

2	0	0	0	0

→Since 35<45, Count[3] increases

2	0	0	1	0

→Since 35>20, Count[0] increases

3	0	0	1	0

→Since 32>11, Count[1] increases

3	1	0	1	0

→Since 32<45, Count[3] increases

3	1	0	2	0

→Since 32>20, Count[1] increases

3	2	0	2	0

→Since 11<45, Count[3] increases

3	2	0	3	0

→Since 11>20, Count[4] increases

3	2	0	3	1

→Since 45>20, Count[3] increases

3	2	0	4	1

The following diagram is for the change of array *S*:

S[3] = A[0]

			35	

→
S[2] = A[1]

		32	35	

→
S[0] = A[2]

11		32	35	

→
S[4] = A[3]

11		32	35	45

→
S[1] = A[4]

11	20	32	35	45

4. (15 pts) Describe in English a simple method for determining if a piece of text contains a word unit palindrome (e.g. "You should ***mind your own business: Own your mind.*** Curiosity killed the cat."). Would your design change if a human or a computer were to perform the method? Explain why or why not. Note that your method should return true even if the word unit palindrome is embedded within a larger piece of text (i.e. even if there is extra text before and / or after it: ").

   1. First, take out all punctuations and convert all words into lowercase letters.

   2. Assume the palindrome has odd number of words. Start from the first word. Consider every word as the middle point of a palindrome and check if the two words beside it are the same by using substring method. if yes, return true.

   3. Then assume the palindrome has even number of words and check if any two consecutive words are the same using substring method. If yes, return true.

   4. Otherwise, return false.

   My design won't change if I were to determine the existence of palindrome. This method is very efficient because it starts from a single word and moves towards both sides of the text. For human, it might be quicker because we can do step 2 and 3 at the same time in mind.