Homework 6

Due date: Feb 27, 2020, 9:30am

Objective

- Perform a best and worst case analysis of an algorithm.
- Determine the growth rates of various functions.

Exercises

1. (18 points)
   Describe the worst and best case of the following method. Determine the time complexity of the worst and best case using the big-O notation.
   a) The following method determines whether the given phrase is a palindrome. Space characters are ignored in the phrase.

```
boolean isPalindrome (Deque<character> phrase)
   character charHead, charTail;
   while (not phrase.isEmpty()) {
      charHead = " ";
      while (not phrase.isEmpty() AND charHead == " ")
         charHead = phrase.removeHead();

      charTail = " ";
      while (not phrase.isEmpty() AND charTail == " ")
         charTail = phrase.removeTail();

      if (charHead != " " && charTail != " " && charHead != letterTail)
         return false;

   return true;
```

   Description of the worst case:

   Time complexity of the worst case:

   Description of the best case:

   Time complexity of the best case:

b) In the following method, assume that a random value is generated in constant time.

```
int[] generateSortedValues (int values[0,...,n-1])
   for (i = 0,...,n-1)
      values[i] = generate a random value;
      values.insertionSort();

   return values;
```

Description of the worst case:

Time complexity of the worst case:

Description of the best case:

Time complexity of the best case:

c) When determining the time complexity of the following method, watch out for the input size!

```
double getMedianOfDiagonal (int matrix[0,...,n-1][0,...,n-1])
   int i = 0;
   boolean sorted = true;
   for (i = 0,...,n-1)
      diagonal[i] = matrix[i][i];
      if (i > 0 && diagonal[i-1] > diagonal[i])
         sorted = false;

   if (!sorted)
      diagonal.selectionSort();

   int mid1 = n/2;
   int mid2 = (n-1)/2;
   return (diagonal[mid1] + diagonal[mid2])/2;
```

Description of the worst case:

Time complexity of the worst case:

Description of the best case:

Time complexity of the best case:

2. (12 points)
   For each of the following three recursive methods, write a non-recursive method that accomplishes exactly the same task.

a)
```
void printQueue (Queue queue)
    if (not queue.isEmpty())
       Object obj = queue.dequeue();
       obj.print();
       printQueue(queue);
```

b)
```
void printQueue (Queue queue)
    if (not queue.isEmpty())
       Object obj = queue.dequeue();
       printQueue(queue);
       obj.print();
```

c)
```
void runSlotMachine(int availableCash)
    if (availableCash <= 1000)
       return;
    sum = 0;
    while (sum < 100)
       wait until coin has been entered by the player;
       coin = get value of entered coin;
       sum += coin;
    userwins = playRoundOfLuckySeven();
    if (userwins)
       pay out 1000 in coins to the player;
       availableCash -= 1000;
    runSlotMachine(availableCash);
```

d) What is the disadvantage of the recursive algorithm ofExercise 2(c) above compared to a nonrecursive algorithm?

3. (6 points)
Write pseudocode for a method that reverses the order of the elements in a stack using one additional stack and possibly some variables. Do not use (explicitly) any other additional collection. Your method has to use recursion. The only valid stack operations are isEmpty, push, pop, and peek. You are not allowed to use a pointer or reference for a stack, that is, an operation of the form "stack1 = stack2;" is not valid where stack1 and stack2 are stacks. (See also HW2, Exercise 4)

   You will likely find that the recursive implementation is much simpler than the non-recursive implementation. Consider why that is the case. (You do not need to submit your considerations.)

4. (6 / 3 / 5 points)
The algorithm maximum determines the greatest element in the array A[p. . . q], where p is the lowest index in the array A and q is the highest index in A.

```
int maximum (int[] A, int p, int q)

   if (p == q)
      return A[p];

   int k, l, max1, max2, max3;
   k = p + roundDown((q-p+2)/3);
   l = k + roundDown((q-p+2)/3);
   max1 = maximum(A, p, k-1);
   max2 = maximum(A, k, l-1);
   max3 = maximum(A, l, q);
   if (max1 >= max2 && max1 >= max3)
      return max1;
   else if (max2 >= max1 && max2 >= max3)
      return max2;
   else
      return max3;
```

a) The function uses the divide-and-conquer approach to determine the minimum element. Describe what the function does in each of the three steps of divide-and-conquer.

   Divide:

   Conquer:

   Combine:

b) Mark the base case of the recursive algorithm `maximum`.

c) Assume the array `myData` has the elements [6, 3, 8, 2, 9, 5, 7, 4, 1] where the first element is stored at index 0 and the last element is stored at index 8. How many recursive calls to maximum are made when the function call `maximum(myData,0,8)` is made? Include the originating call `maximum(myData,0,8)` in your count.

Submission

Turn in a hard-copy with your solutions at the beginning of the class meeting on Feb 27. The solutions can be hand-written or typed.