Week 2 Problem Set
Analysis of Algorithms

[Show with no answers] [Show with all answers]

1. (Counting primitive operations)

   The following algorithm

   - takes a sorted array A[1..*n*] of characters
   - and outputs, in reverse order, all 2-letter words vω such that v≤ω.

   ```
   for all i=n down to 1 do
       for all j=n down to i do
           print "A[i]A[j]"
       end for
   end for
   ```

   Count the number of primitive operations (evaluating an expression, indexing into an array). What is the time complexity of this algorithm in big-Oh notation?

   [show answer]

2. (Big-Oh Notation)

   a. Show that $\sum_{i=1}^{n} i^2 \in O(n^3)$

   b. Show that $\sum_{i=1}^{n} \log i \in O(n \log n)$

   c. Show that $\sum_{i=1}^{n} \frac{i}{2^i} \in O(1)$

   [show answer]

3. (Algorithms and complexity)

   Let $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ be a polynomial of degree n. Design an O(*n*)-time algorithm for computing the function *p(x)*.

   Hint: Assume that the coefficients a_i are stored in an array A[0..n].

   [show answer]

4. (Ordered linked lists)

   A particularly useful kind of linked list is one that is sorted. Give an algorithm for inserting an element at the right place into a linked list whose elements are sorted in ascending order.

   Example: Given the linked list

   ```
   L = 17 26 54 77 93
   ```

   the function `insertOrderedLL(L,31)` should return the list

   ```
   L = 17 26 31 54 77 93
   ```

   Hint: Develop the algorithm with the help of a diagram that illustrates how to use pointers in order to
   - find the right place for the new element and
   - link the new element to its predecessor and its successor.

   [show answer]

5. (Advanced linked list processing)

   Describe an algorithm to split a linked list in two halves and output the result. If the list has an odd number of elements, then the first list should contain one more element than the second.

   Note that:
   - your algorithm should be 'in-place' (so you are not permitted to create a second linked list or use some other data structure such as an array);
   - you should not traverse the list more than once (e.g. to count the number of elements and then restart from the beginning).

   An example of the result of the algorihtm could be

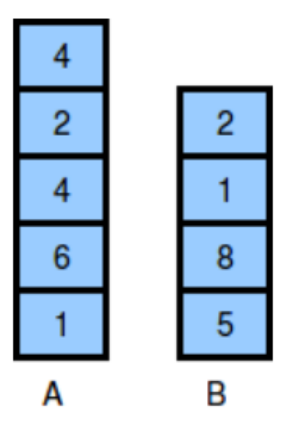
   ```
   Linked list: 17 26 31 54 77 93 98
   First half:  17 26 31 54
   Second half: 77 93 98
   ```

   [show answer]

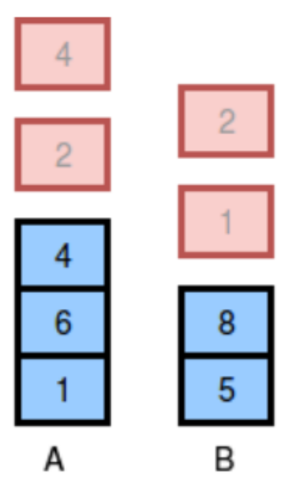
6. **Challenge Exercise**

   Suppose that you are given two stacks of non-negative integers A and B and a target threshold $k \geq 0$. Your task is to determine the maximum number of elements that you can pop from A and B so that the sum of these elements does not exceed *k*.

   Example:



   Maximum number of elements that can be popped without exceeding *k* = 10 is 4:



   If k = 7, then the answer would be 3 (the top element of A and the top two elements of B).

   a. Write an algorithm (in pseudocode) to determine this maximum for any given stacks A and B and threshold *k*. As usual, the only operations you can perform on the stacks are pop() and push(). You *are* permitted to use a third "helper" stack but no other aggregate data structure.

   b. Determine the time complexity of your algorithm depending on the sizes *m* and *n* of input stacks A and B.

   Hints:
   - A so-called greedy algorithm would simply take the smaller of the two elements currently on top of the stacks and continue to do so as long as you haven't exceeded the threshold. This won't work in general for this problem.
   - Your algorithm only needs to determine the number of elements that can maximally be popped without exceeding the given *k*. You do not have to return the numbers themselves nor their sum. Also you do not need to restore the contents of the two stacks; they can be left in any state you wish.

   [show answer]