American University of Armenia, CSE
CS121 Data Structures A, C
Fall 2021

Homework Assignment 2

Due Date: Monday, September 27 by 23:59 electronically on Moodle

Please solve the programming tasks either in Java or C++, following good coding practices (details are on Moodle).

1. **(16 points)** Implement *recursive* variants of *selection sort* and *insertion sort* for an array of strings using lexicographical order. Test both methods in a program that inputs the length and the string elements of a sequence and outputs the sorted sequence. The two methods for the sorting algorithms are not allowed to use loops.

2. **(10 points)** Assume we have a sequence S of integers. Among the first few elements, the step between two adjacent integers, i.e. their difference, is odd. However, starting at a certain index k and to the end of S, the step between any pair of adjacent integers is even.

   Write an **efficient** method/function that, given such an array, determines the first index k where the step becomes even. Write a program that inputs the length of the array and its elements, uses your method to output the first index k.

   Give big-O estimates for the running times of both the method and the whole program. Briefly justify your answers.

sample input	sample output
7 5 8 -3 -5 1 -3 -9	2

3. **(5 points)** Give an example input of length 10 on which merge sort runs in $O(n \log n)$ time, insertion sort runs in $O(n^2)$ time, and quick sort (where the pivot is the first element) runs in $O(n^2)$ time to sort in non-decreasing order of elements. Illustrate all three sorting algorithms for that example. Specify the running times of all three algorithms on the reverse of your example.

4. **(25 points)** The task is to implement *merge sort* for a singly linked list of integers. To achieve that:

   (a) Implement a generic method/function that, given the head node of a singly linked list L, splits it into two singly linked lists L_{odd} and L_{even} and returns the head nodes of L_{odd} and L_{even} in a length-2 array. All the elements at odd positions in the original list need to go into L_{odd} and, similarly, all the elements at even positions in the original list need to go into L_{even}.

   Your method may traverse the original list **only once**. You are **not allowed** to create any nodes or list objects. While we refer to lists L, L_{odd} and L_{even} here, the method/function should **not** use any list objects; only node sequences.

(b) Implement a method/function that, given the head nodes of two sorted singly linked lists of integers, merges them together into one sorted singly linked list and returns its head node.

Your method may traverse the original lists **only once**. You are **not allowed** to create any nodes or list objects.

(c) Implement a method/function that, given the head node of a singly linked list of integers, applies merge sort on it and returns the head node of the resulting sequence. Your implementation should rely on the previous methods/functions as subroutines.

Test the resulting sorting algorithm in a program. Discuss the worst-case execution times of each of the implemented methods.

5. (24 points) Consider an array of n non-negative integers in the range $0..(10^8 - 1)$. One way to sort it is to use the following hybrid algorithm:

(1) divide the sequence into $N = 10^4$ buckets with singly linked lists representing the elements in the ranges $0..(10^4 - 1)$, $10^4..(2 \times 10^4 - 1)$, $\ldots$, $((10^4 - 1) \times 10^4)..(10^8 - 1)$;

(2) sort the contents of each bucket using *bubble sort*;

(3) combine the resulting sorted subsequences together to get the final result.

Implement the algorithm described. Note that you will need to implement bucket sort for a singly linked list as a subroutine. Express the best- and worst-case execution times of the hybrid sorting algorithm in terms of n and N. Is this sorting algorithm an optimal choice for such a sequence? If not, which algorithm(s) would be better suited for the task. Briefly justify your answers.

6. (20 points) Consider a sequence of non-negative integers of maximum six digits. Implement the *radix sort* algorithm for such a sequence. You can use an array to represent the sequence to be sorted, but should use singly linked lists for the intermediate steps of the algorithm, i.e. the buckets should be singly linked lists. Illustrate the use of your implementation in a program.