EL9343 Homework 3

Due: Sept. 28st 8:00 a.m.

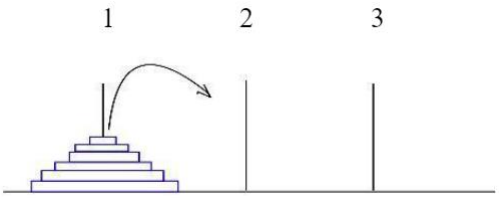
1. True or False questions:
   (a) One can find the maximum sub-array of an array with n elements within $O(n \log n)$ time.
   (b) In the maximum sub-array problem, combining solutions to the sub-problems is more complex than dividing the problem into sub-problems.
   (c) Bubble sort is stable.
   (d) When input size is very large, a divide-and-conquer algorithm is always faster than an iterative algorithm that solves the same problem.
   (e) It takes $\Theta(n)$ time to check if an array of length n is sorted or not.
   (f) Insertion sort is NOT in-place.
   (g) The running time of merge-sort in worst-case is $O(n \log n)$.

   Answer:
   (a) T
   (b) T
   (c) T
   (d) F, it depends on the running time function (also other constraints like space occupancy)
   (e) T
   (f) F, insertion sort is in-place
   (g) T
2. Consider sorting n numbers stored in array A, indexed from 1 to n. First find the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n-1$ elements of A.
   (a) Write pseudo-code for this algorithm, which is known as *selection sort*.
   (b) What loop invariant does this algorithm maintain?
   (c) Give the best-case and worst-case running times of selection sort in Θ-notation.

   Answer:
   (a)
   SELECTION-SORT(A)
   $n \leftarrow A$.length
   for $j \leftarrow 1$ to $n-1$ **do**
   $smallest = j$
   for $i \leftarrow j+1$ to n **do**
   if $A[i] < A[smallest]$ **then**
   $smallest \leftarrow i$
   end if
   end for
   exchange $A[j]$ with $A[smallest]$
   end for
   (b) At the start of each iteration of the outer **for** loop, the sub-array $A[1, 2, \ldots, j-1]$ consists of the smallest j - 1 elements in array $A[1, 2, \ldots, n]$, and this sub-array is in sorted order.
   (c) The running time is $\Theta(n^2)$ for all cases.
3. A mathematical game (or puzzle) consists of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with **n** disks stacked on a **start** rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the **end** rod, obeying the following rules:
   i Only one disk may be moved at a time;

ii Each move consists of taking the top disk from one of the rods and placing it on top of another rod or on an empty rod;

iii No disk may be placed on top of a disk that is smaller than it.



Please design a MOVE(n, start, end) function to illustrate the minimum number of steps of moving n disks from start rod to the end rod.
You **MUST** use the following functions and format, otherwise you will not get points of part (a) and (b):

```
def PRINT(origin, destination):
    print("Move the top disk from rod", origin, "to rod", destination)

def MOVE(n, start, end): # TODO: you need to design this function
    pass
```

For example, the output of MOVE(2, 1, 3) should be:

```
Move the top disk from rod 1 to rod 2
Move the top disk from rod 1 to rod 3
Move the top disk from rod 2 to rod 3
```

(a) Give the output of MOVE(4, 1, 3).

(b) Fill in the function MOVE(n, start, end) shown above. You can use Python, C/C++ or pseudo-code form, as you want.

(c) What's the minimum number of moves of MOVE(5, 1, 3), and MOVE(n, 1, 3)?

Answer:

(a)

```
Move the top disk from rod 1 to rod 2
Move the top disk from rod 1 to rod 3
Move the top disk from rod 2 to rod 3
Move the top disk from rod 1 to rod 2
Move the top disk from rod 3 to rod 1
Move the top disk from rod 3 to rod 2
Move the top disk from rod 1 to rod 2
Move the top disk from rod 1 to rod 3
Move the top disk from rod 2 to rod 3
Move the top disk from rod 2 to rod 1
Move the top disk from rod 3 to rod 1
Move the top disk from rod 2 to rod 3
Move the top disk from rod 1 to rod 2
Move the top disk from rod 1 to rod 3
Move the top disk from rod 2 to rod 3
```

(b) In Python,

```
def MOVE(n, start, end):
    mid_rod = 6 - start - end # mid_rod is the remaining rod
    if n == 1:
        PRINT(start, end)
    else:
```

```
MOVE(n - 1, start, mid_rod)
MOVE(1, start, end)
MOVE(n - 1, mid_rod, end)
```

Answers in C/C++ or pseudo-code is also OK as long as they are correct.

(c) Let $T(n)$ denote the minimum number of moves of MOVE(n, 1, 3), then $T(5) = 31$, $T(n) = 2^n - 1$. It is easy to prove that by having,

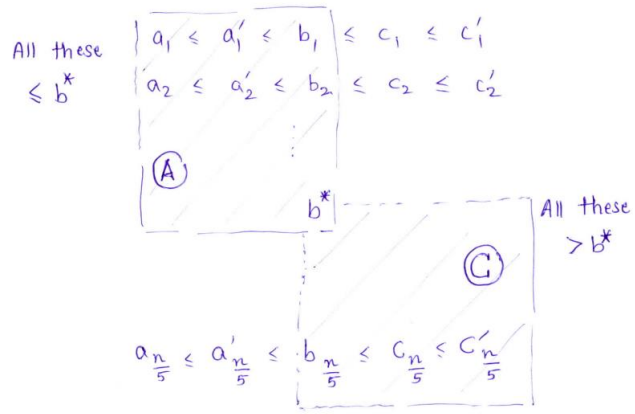
$$T(1) = 1$$
$$T(n) = T(n-1) + T(1) + T(n-1)$$

4. Finding the median of an unordered array in $O(n)$ (Part I). Let's consider the algorithm 1,

Algorithm 1 FindMedian(L)

Input: L as input list containing n real numbers
Output: The median of L as m

1: **if** $n \leq 10$ **then**
2: Sort L and return the median m
3: **end if**
4:
5: Divide L into $\frac{n}{5}$ lists of size 5 each
6: (Making up the corner case that n is not at multiple of 5 is skipped. An easy way is to add minimum and maximum of the array at the head and the tail)
7:
8: Sort each list, let i^{th} list be $a_i \leq a'_i \leq b_i \leq c_i \leq c'_i, i = 1, 2, \ldots, \frac{n}{5}$
9: Recursively find median of $b_1, b_2, \ldots, b_{\frac{n}{5}}$, call it b^*
10: Reorder indices so that $b_1, b_2, \ldots, b_{\frac{n}{10}} \leq b^* < b_{\frac{n}{10}+1}, \ldots, b_{\frac{n}{5}}$
11: Define A and C as shown in the figure



12: Make sure both A and C have equal number of elements, approximately $\frac{3}{10}n$, when n is large
13:
14: Drop A and C from the original list L, to get a new list L', with $n - \frac{3}{10}n - \frac{3}{10}n = \frac{2}{5}n$ elements
15: Find median of remaining L' recursively and return it as m.

(a) Let $T(n)$ be the running time of this algorithm, find the recurrence formula, and then solve it.

(b) Is this algorithm correct? If yes, try to prove it; otherwise, find a counter case (something like the true median is in A or C and is dropped).

Answer:

(a) For $T(n)$ in the recurrence formula, we start by analyzing each component of $T(n)$.
At line 8, sorting a list of 5 elements is $O(1)$, while there are $\frac{n}{5}$ lists to sort, making it $O(n)$ in total.
At line 9, the time of finding median of a list of $\frac{n}{5}$ elements is $T(\frac{n}{5})$.
At line 10, reordering takes $O(n)$ time. It is different from sorting that we just keep $b_i \leq b^*, i = 1, 2, \ldots, \frac{n}{10}$, but not require that $\forall i < j, b_i < b_j$.

At line 14, dropping A and C takes $O(n)$ time, for there are $\frac{3}{10}n + \frac{3}{10}n = \frac{3}{5}n$ elements to drop and to drop an element is $O(1)$.
At line 15, the time of finding median of a list of $\frac{2}{5}n$ elements is $T(\frac{2}{5}n)$.
For other operations in the algorithm, like dividing the list, defining sets and sorting a list of less than 10 elements, they are all $O(1)$.
Putting these all together, we have,

$$T(n) = T(\frac{n}{5}) + T(\frac{2}{5}n) + O(n)$$

To solve the recurrence, we use substitution method.

$$\exists c > 0, T(n) \leq T(\frac{n}{5}) + T(\frac{2}{5}n) + cn$$

$$\text{Assume that } \forall k \leq n, T(n) \leq dn, \text{ for some } d \geq \frac{5}{2}c,$$

$$\text{Then, } T(n+1) \leq d\frac{1}{5}(n+1) + d\frac{2}{5}(n+1) + c(n+1) \leq d(n+1)$$

$$\text{Also we have } T(n) \leq C' \text{ when } n \leq 10,$$

$$\text{and we can choose } d \text{ sufficiently large to cover the initial conditions}$$

$$\therefore T(n) = O(n)$$

(b) The algorithm is wrong (but we are gonna fix it in Part II the next HW). Consider the counter case as below,

$$1,\ 2,\ 3,\ 4, 14,$$
$$5,\ 6, 13, 15, 16,$$
$$7,\ 8, 17, 18, 19,$$
$$9, 10, 23, 24, 25,$$
$$11, 12, 20, 21, 22$$

If this is the input to the algorithm, it remains the same after line 10. We can see that the true median 13 is in the set A and will be dropped next.
To prove an algorithm is wrong, we only need one counter case. This is only an illustration, and as long as the answer makes sense you should receive full points.