EL-GY 9343 Homework 3

Yihao Wang
yw7486@nyu.edu

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. True

(b) In the maximum sub-array problem, combining solutions to the sub-problems is more complex than dividing the problem into sub-problems. False

(c) Bubble sort is stable. True

(d) When input size is very large, a divide-and-conquer algorithm is always faster than an iterative algorithm that solves the same problem. False

(e) It takes $O(n)$ time to check if an array of length n is sorted or not. True

(f) Insertion sort is NOT in-place. False

(g) The running time of merge-sort in worst-case is $O(n \log n)$. True

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.

Algorithm 1 Selection Sort(A)

Input: A as input list to be sorted, containing n real numbers
Output: A sorted version of A

for $i = 1, 2, \ldots, n-1$ **do**
    $min\_idx \leftarrow i$
    for $j = i+1, i+2, \ldots, n$ **do**
        if $A[j] < A[min\_idx]$ **then**
            $min\_idx \leftarrow j$
        end if
    end for
    SWAP$(A[i], A[min_idx])$
end for
return A

(b) What loop invariant does this algorithm maintain?

At the start of each iteration, the sub-array $A[1, \ldots, i-1]$ is sorted.

(c) Give the best-case and worst-case running times of selection sort in Θ-notation.

Best case: $\Theta(n^2)$.

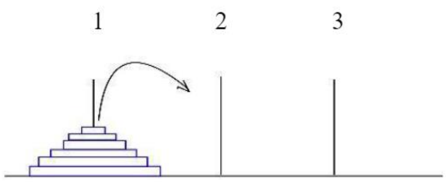
Worst case: $\Theta(n^2)$.

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).

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

```
def MOVE(n, start, end): # Python code
    if n == 1:
        PRINT(start, end)
    else:
        temp_rod = 6 - start - end
        MOVE(n-1, start, temp_rod)
        MOVE(1, start, end)
        MOVE(n-1, temp_rod, end)
```

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

Minimum moves of MOVE(5, 1, 3): 31

Minimum moves of MOVE(n, 1, 3): $2^n - 1$

4. Finding the median of an unordered array in O(n) (Part I). Let's consider the algorithm 2,

Algorithm 2 FindMedian(L)

Input: L as input list containing n real numbers

Output: The median of L as m

if $n < 10$ **then**

Sort L and return the median m

end if

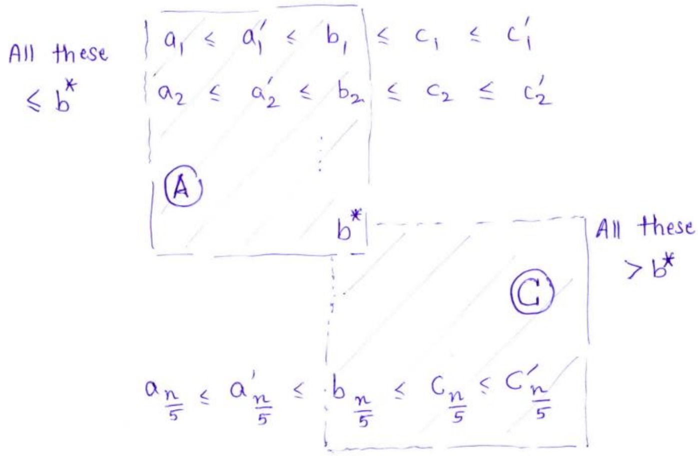
Divide L into $\frac{n}{5}$ lists of size 5 each

Sort each list, let i^{th} list be $a_i \le a'_i \le b_i \le c_i \le c'_i, i = 1, 2, \dots, \frac{n}{5}$

Recursively find median of $b_1, b_2, \dots, b_{\frac{n}{5}}$, call it $b*$

Reorder indices so that $b_1, b_2, \dots, b_{\frac{n}{10}} \le b* < b_{\frac{n}{10}+1}, \dots, b_{\frac{n}{5}}$

Define A and C as shown in the figure (both A and C have approximately $\frac{3}{10}n$ elements)



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

Find median of remaining L' recursively and return it as m.

Note that there could be some corner cases to consider, like n is not a multiple of 5 (can be fixed by padding), or the number of elements in A or C is inaccurate. However, as long as n is large enough, this little inaccuracy will not infect the analysis of complexity.

(a) Let $T(n)$ be the running time of Algo.2, find the recurrence formula, and then solve it.

The Algo.2 recursively calls itself twice, with input size of $\frac{1}{5}n$ and $\frac{2}{5}n$ respectively. Moreover, it calls a sorting algorithm on $\frac{1}{5}n$ lists, each of size 5, which contributes a time complexity of $\frac{1}{5}n \cdot C \sim O(n)$. The rest part of the Algo.2 has a constant time complexity. Therefore, the recurrence formula is

$$T(n) \simeq T(\frac{n}{5}) + T(\frac{2n}{5}) + O(n), \tag{1}$$

which we can analyse with recursion tree as in **Figure 1**. We can see that the time complexity of $T(n)$ is almost the same as that of $S(n)$:

$$S(n) = \frac{3}{5}S(n) + O(n), \tag{2}$$

except for the total recursion level number on each branch. Moreover, the total recursion level of $T(n)$ must be less than that of $S(n)$ since $\max(\log_{\frac{5}{2}} n, \log_5 n) < \log_{\frac{5}{3}} n$. Thus we have $T(n) = O(S(n))$. To solve $S(n)$, we can apply Master's method where $a = 1$, $b = \frac{5}{3}$ and $f(n) = n$ which indicates that $S(n) = O(n)$. Or we can simply sum up all the cost of $S(n)$:

$$\begin{aligned} S(n) &\le \left[1 + \frac{3}{5} + (\frac{3}{5})^2 + \cdots + (\frac{3}{5})^{+\infty}\right] \cdot n \\ &= \frac{1}{1 - \frac{3}{5}} \cdot n = \frac{5}{2}n \implies S(n) = O(n) \end{aligned}$$

Combined with $T(n) = O(S(n))$, we conclude that $T(n) = O(n)$.

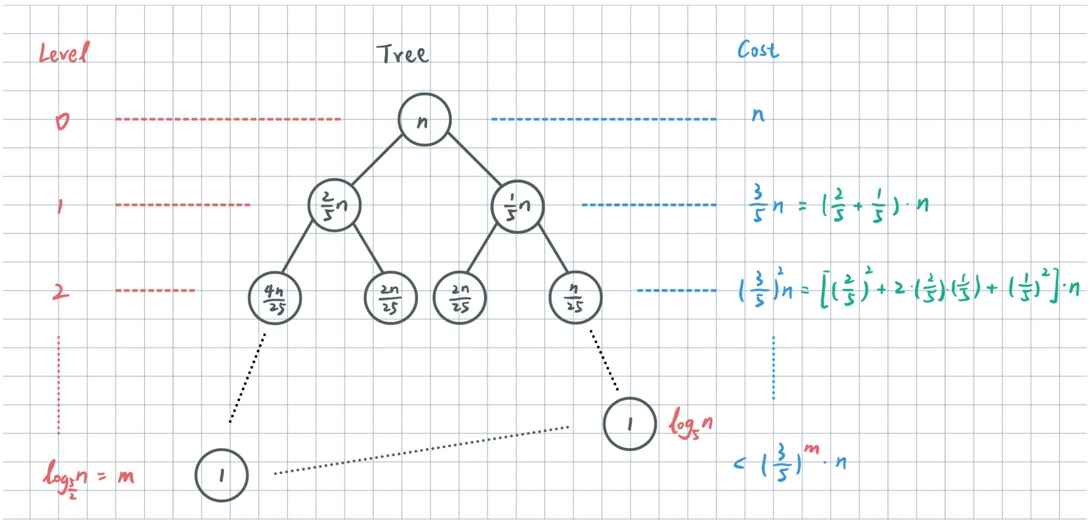


Figure 1: Recursion tree of $T(n)$ in Eq.(1)

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

No, it is not correct. Consider a simple case that the original list L is $[1, 2, 3, \ldots, 15]$. According to Algo.2, we first divide L into $\frac{15}{5} = 3$ lists, which are $[1, 2, \ldots, 5]$, $[6, 7, \ldots, 10]$, and $[11, 12, \ldots, 15]$. Since they are all in sort, we don't need to bother finding the median of $[b_1, b_2, b_3]$ or re-indexing these sub-lists. Next, according to the definition of A and C, the true median, 8 , is discard, as shown in **Figure 2**. Therefore, we have described a counter case.

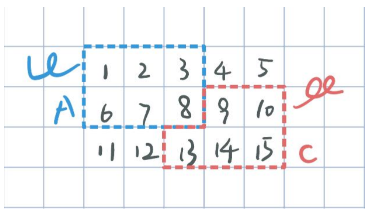


Figure 2