Cmpe 300: Homework 2 — Due: November 21st 17:00

Solve the following questions in LaTeX or using a word processor. Bring a hard copy of your homework to the midterm and deliver it to us. You do not have to print the questions. Your answers need not be on separate pages.

The purpose of this homework is to familiarize you with algorithm analysis. This is an individual assignment, so work on your own. Please do not submit just an answer, and rather show all your reasoning. For any further questions, contact the assistant at utkan.gezer@boun.edu.tr.

1. **(40 pts)** Answer the following questions considering algorithm 1.

Algorithm 1 Stack Sort (all rights reserved)

```
Input: L[0 : n − 1] (a list of size n)
Output: L[0 : n − 1] (sorted in increasing order)
 1: function StackSort(L[0 : n − 1])
 2:     // S[0 : n − 1] is initialized as an array of n stacks. Stacks are initially empty.
 3:     call Push(S[0], L[0])
 4:     s ← 0                                          ▷ Index of the last non-empty stack
 5:     for i ← 1 to (n − 1) do                ▷ Prepare ascending stacks with ascending tops.
 6:         call FindStack(S[0 : s], L[i], k)
 7:         call Push(S[k], L[i])
 8:         s ← max(s, k)
 9:     end for
10:     b ← 0                                          ▷ Index of the first non-empty stack
11:     for i ← 0 to (n − 1) do
12:         call Pop(S[b], L[i])                                ▷ Pop the smallest stack.
13:         if Empty(S[b]) then
14:             b ← b + 1
15:         else
16:             k ← b
17:             while k < s .and. Top(S[k]) > Top(S[k + 1]) do      ▷ Ensure ascending tops.
18:                 call Interchange(S[k], S[k + 1])
19:                 k ← k + 1
20:             end while
21:         end if
22:     end for
23: end function
```

Some notes on the modules used in the algorithm:

- Push, Pop, and Empty are the same stack operations you have seen in the lecture.
- Top accesses the top element of the stack, without removing it.
- FindStack module has two input parameters; a list of $s + 1$ stacks, where the top element of each subsequent stack is larger than the top element of the previous stack, and a value ($L[i]$) to search. The output parameter (k) is set to the index of first stack with the top element greater than or equal to the searched value, or to $s + 1$ if there is no such stack. It is an extension of the binary search algorithm, and makes just as many comparisons as a binary search would.

Consider the comparisons as the basic operations. Note that Empty and FindStack modules also make comparisons. A logical, concise, and gap-free explanation with reasons is necessary to get the full credit.

(a) **(15 pts)** Describe a best-case input for this algorithm. Then, analyze the best-case for the algorithm exactly. Finally, state the Θ-class for the best-case.

(b) **(25 pts)** Do the same for the worst-case.

2. **(30 pts)** In the following algorithm, consider the comparisons as the basic operations. Then, give an exact formula for its worst-case time complexity as a recurrence relation. You do not have to convert it into the closed-form.

   It is assumed that the input list is of size $2^{(2^k)}$ for some non-negative integer k. State your assumptions if you have any other.

Algorithm 2 Root Search

```
Input: $L[0 : n-1]$ (a sorted list of size $n = 2^{(2^k)}$), $y$ (searched value)
Output: $x$ (index of searched value in the list, $\infty$ if searched value is not in the list)
function RootSearch($L[0 : n-1], y, x$)
    if $n = 2$ then
        if $L[0] = y$ then
            $x \leftarrow 0$
        else if $L[1] = y$ then
            $x \leftarrow 1$
        else
            $x \leftarrow \infty$
        end if
    else
        $b \leftarrow n - \sqrt{n}$
        while $b > 0$ .and. $L[b] > y$ do
            $b \leftarrow b - \sqrt{n}$
        end while
        call RootSearch($L[b : b + \sqrt{n} - 1], y, x$)
        $x \leftarrow b + x$
    end if
end function
```

3. **(30 pts)** Find a closed-form formula for each of the following recurrence relations. Justify your answer.

(a) $T(n) = \begin{cases} 4 \cdot T(n-1) - 3 \cdot T(n-2) & \text{for } n > 1, \\ 17 & \text{for } n = 1, \\ 9 & \text{for } n = 0. \end{cases}$

(b) $T(n) = \begin{cases} \frac{T(n-1)}{2} + 1 & \text{for } n > 0, \\ 10 & \text{for } n = 0. \end{cases}$

(c) $T(n) = \begin{cases} n \cdot T(n/2) & \text{for } n > 1, \\ 1 & \text{for } n = 1. \end{cases}$ Assume that n is a non-negative power of 2.