CmpE 300
Analysis of Algorithms
Fall 2019
Homework 2

Emilcan ARICAN - 2016400231

Solution 1

1.a

Best-case input is a non-increasing series. During creating stacks, only one stack is enough since the when an element is placed next placed element is less than or equal to placed one. Therefore, FindStack function takes one basic operation to find the proper stack. Also max function takes one basic operation. This for loop has n-1 iterations. So, it makes 2n-2 basic operations. There are n comparisons for the for loop when we take it into account, number of basic operations until now is 3n-2.

It will make empty checks and the controls at the while at an iteration. However since there is only one stack is used, there can't be the stack interchange. So it won't enter the while loop. Since "k < s" control always false for this input, second comparison of the while won't be executed. Therefore, there is only 1+1=2 basic operations at every iteration. Total number of iterations are n. So the number of basic operations of this part becomes 2n. However at last iteration Empty(S[b]) is true, since while comparison won't be performed it should be one less. So it makes 2n-1.

There are n+1 comparisons for the for loop.

So number of basic iterations until here is:

$$\begin{aligned} 3n - 2 + 2n - 1 + n + 1 &= 6n - 2 \\ B(n) &= 6n - 2 \\ B(n) &\in \Theta(n) \end{aligned}$$

1.b

$$[\dots L_1 \dots][\dots L_2 \dots]$$

Worst-case input is two series L_1, L_2 forming a series like above mentioned manner. Every element of L_1 is greater than every element of L_2. Also L_1, L_2 are increasing series with same number of elements. And, L_1 comes before L_2. Assume n is even.

This input leads to n/2 stacks with 2 element in each. While creating stacks, at first there are only one stack and adds new stacks until there are n/2 stacks. Because first n/2 elements are increasing. Rest of them places one to each since these are increasing either, but since these are less than L_1's elements they got placed into already existing stacks. At first iteration makes log(1) basic operations and goes like log(2), log(3), ... , log((n/2) - 1).

After (n/2)th stack is created, log(n/2) basic operations are made, for the rest n/2 elements.

For loop has n-1 iterations so n-1 basic operations are made by max function, there are total of n controls for the for loop when we take it it into account number of basic operations are log(1) + log(2) + ... + log((n(/2)) - 1 - 1) + (n/2).log(n/2) + 2n - 1.

At every iteration, first pops an element and moves it to end of the stack list because newly emerged number is greater than every element of the L_2 and the every element of L_1 emerged before. Therefore it has to be go to the end of the stack list, it takes (n/2)-1 iterations. So for the first n/2 of them ((n/2)-1)*2 + 1 (there are two comparisons at while, plus one at the end is last "k<s" control to exit from the while loop) operations are made. Also there are n/2 empty checks. So in total $(n/2)(((n/2)-1)*2+1)+n/2 = (n-1).(n/2)+n/2$. Finally, $\frac{n^2}{2}$.

Remaining n/2 elements are only elements in their stack. While these elements are being removed first pops then controls if stack is empty. So it makes in total n/2 comparisons for remaining.At this part, there are $\frac{n^2}{2} + \frac{n}{2}$ basic operations. So it makes $\frac{n^2+n}{2}$.

Last for takes n iterations therefore there are n+1 controls for the for loop.

So basic operations in total are sum of the all of these:

$$W(n) = \frac{n^2+n}{2} + log(1) + log(2) + ... + log((n/2)-1) + (n/2).log(n/2) + 2n - 1 + n + 1$$

$$W(n) = \frac{n^2}{2} + log(1) + log(2) + ... + log((n/2)-1) + log(n/2) + ((n/2)-1).log(n/2) + \frac{7n}{2}$$

$$W(n) = \frac{n^2}{2} + log((n/2)!) + ((n/2)-1).log(n/2) + \frac{7n}{2}$$

Dominant characteristic of W(n) is n^2 while n goes to infinity. Therefore, $W(n) \in \Theta(n^2)$.

Solution 2

Worst-case of this algorithm is that searched value is at the 1st index or less than every element at the list. Since it performs maximum amounts of while loop operations at this case, it leads to maximum numbers of comparisons.

Firstly, there is a "n = 2" control which is our first basic operation. Basic operations until here is 1.

Then, it looks to value at (n-$\sqrt{n}$)th index, since searched value is less looks to index with (n-$2\sqrt{n}$). It takes $\sqrt{n}$ iterations of while so with two comparisons except by the last one (since when b becomes 0, it wont make second comparison of the while); the last one there is only one comparison, so there are $2\sqrt{n}$ - 1. So, basic operations until here is $2\sqrt{n}$ - 1 + 1 = $2\sqrt{n}$

Then calls the RootSearch function with parameter $\sqrt{n}$. And it leads to W($\sqrt{n}$) basic operations. So final number of basic operations is W($\sqrt{n}$) + $2\sqrt{n}$.

$$W(n) = W(\sqrt{n}) + 2\sqrt{n}$$

Base case of this algorithm is X(2) since it ends the recursion. At this call, there are three controls which are "L[0] = y", "L[1] = y" and "n = 2". Therefore,

$$W(2) = 3$$

Solution 3

3.a

characteristic equation method:

$$T(n) = 4T(n-1) - 3T(n-2)$$

characteristic eqn.

$$a^2 = 4a - 3$$
$$a^2 - 4a + 3 = 0$$
$$a_1 = 3$$
$$a_2 = 1$$

$$T(n) = c_1.a_1^n + c_2.a_2^n$$
$$T(n) = c_1.3^n + c_2.1^n$$
$$T(n) = c_1.3^n + c_2$$
$$T(1) = c_1.3 + c_2$$
$$17 = c_1.3 + c_2$$
$$T(0) = c_1.3^0 + c_2.1^0$$
$$9 = c_1 + c_2$$
$$c_1 = 4$$
$$c_2 = 5$$

$$T(n) = 4.3^n + 5$$

3.b

back substitution method:

$$T(n) = \frac{T(n-1)}{2} + 1$$
$$T(n) = \frac{T(n-2)}{4} + \frac{1}{2} + 1$$
$$T(n) = \frac{T(n-3)}{8} + \frac{1}{4} + \frac{1}{2} + 1$$
$$\vdots$$
$$T(n) = \frac{T(0)}{2^n} + \frac{1}{2^n} + \frac{1}{2^{n-1}} + ... + \frac{1}{2^0}$$
$$T(n) = \frac{T(0)}{2^n} + \frac{1 - (\frac{1}{2})^n}{1 - \frac{1}{2}}$$
$$T(n) = \frac{10}{2^n} + 2(1 - \frac{1}{2^n})$$
$$T(n) = \frac{10}{2^n} + 2 - \frac{2}{2^n}$$
$$T(n) = \frac{8}{2^n} + 2$$

3.c

$$
\begin{aligned}
T(n) &= n.T(n/2) \\
n &= 2^k \\
T(2^k) &= 2^k.T(2^{k-1}) \\
T(2^k) &= 2^k.2^{k-1}.T(2^{k-2}) \\
T(2^k) &= 2^k.2^{k-1}.2^{k-2}.T(2^{k-3}) \\
&\vdots \\
T(2^k) &= 2^k.2^{k-1}.2^{k-2}...2^1.T(2^0) \\
T(2^k) &= 2^{\frac{k.(k+1)}{2}}.T(1) \\
T(2^k) &= 2^{\frac{k.(k+1)}{2}} \\
2^k &= n \\
T(n) &= n^{\frac{(k+1)}{2}} \\
k &= log_2(n) \\
T(n) &= n^{\frac{(log_2(n)+1)}{2}}
\end{aligned}
$$