CSc 226: Operating Systems (Spring 2022)

Written Assignment 1 (W1)

Alex Holland V00

Question 1

(a)

A naive method for computing $p(x)$ for a particular value of x could be done by using a nested loop. The outer loop would calculate the summation from $i = 0$ *to* n. The inner loop would be used to calculate x^i via a method call. Each of these two loops would run in $O(n)$. Thus the total run-time of the naive method would be $O(n^2)$. A pseudo implementation of this naive method can be seen:

```
\\Input: Array (A), value x
\\Ouput: y = P(x)
naiveMethod(A, x)
y <- 0;
for i <- to A.length do:
    k <- 1;
    for j <- 1 to i - 1 do:
        k *= x
    y <- y + A[i] * k
    end
end
```

(b)

Since there is n multiplications and n additions, for each case the multiplications run in $O(n)$ and addition in $O(n)$. The run-time can be described as $O(n+n) = O(2n)$. Thus by big-Oh notation we can characterize the number of multiplications and additions this method evaluation uses as $O(n)$.

Question 2

We have an array A of n distinct integers, to find if there are three integers in A that sum to 0 we can use the following algorithm:

```
3SUM(A)
quickSort(A);                       \\Sort the array of n distinct elements
for i <- 0 to n - 2 do:             \\Loop through array
    left <- i + 1;
    right <- n - 1;
    a <- A[i];
    while (right > left) do:
        b <- A[left];
        c <- A[right];
        if (a + b + c = 0) then:    \\Print out the output if a, b, and c -
            right -= 1;             \\sum to 0
            left += 1;
            print a, b, c;
        else if (a + b + c < 0) then:   \\If the three integers are less then
```

```
            left += 1;                    \\ 0, increment the left
        else:                             \\ If the three integers are greater
            right -= 1;                   \\ then 0, increment the right
    end
end
```

In this algorithm we sort the array and then check all possible 3-way pairs. There are two nested loops; a for and while loop. Thus the running time of this 3SUM algorithm is $O(n^2)$.

Question 3
We must first determine a good pivot. After diving a sequence of numbers into equal-sized groups of 3 elements in $O(1)$ time, LinearSelect sorts each group of size 3 in $\lceil n/3 \rceil * \binom{3}{2} = n$ time. To gather all the medians of each group takes n time. If the running time of LinearSelect is $\lceil n/3 \rceil$, then to compute the median of $\lceil n/3 \rceil$ takes roughly $T(n/3)$ time. So the time Complexity of pivot selection is :

$$T(n) = 2n + T(n/3)$$

Partition in LinearSelect runs in n time.

From out clever pivot selection we are guaranteed that $2 \times \lceil n/\frac{3}{2} \rceil$. Thus in the worst case $n/3$ elements at partitioning are in the lower sequence and $n - n/3 = 2n/3$ are in the greater sequence, or vice versa.

$$\begin{aligned}\text{Clever pivot selection} &= 2n + T(n/3) \\ \text{Partition} &= n \\ \text{onquer recursive call} &= T(2n/3)\end{aligned}$$

Thus the recurrence equation is $T(n) = 3n + T(n/3) + T(2n/3)$.

To prove that a LinearSelect algorithm does not run in O(n) if it uses groups of size 3 and not 7, we can solve a reccurence equation by guessing:

$$\begin{aligned}&\text{Guess } T(n) \le cn \\ &T(n) = 3n + T(n/3) + T(2n/3) \\ &\le 3n + cn/3 + 2cn/3 \\ \\ &3n + cn/3 + 2cn/3 \le cn \\ &\frac{9n + cn + 2cn}{3} \le cn \\ &9n + 3cn \le 3cn \\ &9 \le 0\end{aligned}$$

Since 9 is not less then or equal to 0 (it is not solvable), $T(n)$ does not run in $O(n)$ time.

Question 4

(a)

$$T(n) = 16T(n/4) + n^4$$
$$a = 16, b = 4, c = 4 < log_b a = 2$$
$$\text{Since } c > log_b a \text{, then } T(n) \text{ is } \Theta(n^c)$$
$$T(n) = \Theta(n^4)$$

(b)

$$T(n) = 125T(n/5) + n^2$$
$$a = 125, b = 5, c = 2 < log_b a = 3$$
$$\text{Since } c < log_b a \text{, then } T(n) \text{ is } \Theta(n^{log_b a})$$
$$T(n) = \Theta(n^{log_5 125}) = \Theta(n^3)$$

(c)

$$T(n) = 64T(n/8) + n^2$$
$$a = 64, b = 8, c = 2 < log_b a = 2$$
$$\text{Since } c = log_b a \text{, then } T(n) \text{ is } \Theta(n^2 logn)$$
$$T(n) = \Theta(n^2 logn)$$

Question 5
We want to create a new algorithm for matrix multiplication whose running time is better than Strassen's $O(n^{2:807})$. We can achieve this by creating a method of multiplying two 3×3 matrices using as few multiplications as possible. To do this we must use 21 multiplications or less. We get $T(n/3)$ since the matrices are being partitioned into 1/3 blocks. We would have $21T(n/3)$ recursive calls. Like Strassen's algorithm there is $\Theta(n^2)$ additions and subtractions for two matrices. From this we can create the following reccurence equation:

$$T(n) = 21T(n/3) + \Theta(n^2)$$
$$a = 21, b = 3, c = 2$$
$$\text{Since } c < log_b a \text{, then } T(n) \text{ is } \Theta(n^{log_b a})$$
$$T(n) = \Theta(n^{log_3 21}) = \Theta(n^{2.77})$$

This method requires $O(n^{2.77})$ arithmetic operations to multiply two 3×3 matrices. Thus this new method is faster is better then Strassen's matrix multiplication algorithm.