W1-S22

Parv Bhardwaj

February 2022

1 Question 1

a) Write a recursive method double power(double a, int b) which computes a to the bth power. Your method should work for both positive and negative inputs. You are not allowed to use a power operator – you can only use +, -, *, and /. Also, no loops are allowed. Your solution may be pseudocode or in the general style of Java. 3 P. b) What is the tightest Big-O complexity of your method as a function of a and b? Explain your answer. 2 P. c) What is the maximum depth of the call stack as a function of a and b? 2 P.

1.1 Answer part a: code

```
public static double power(double a, int b) {

      if (b != 0 && b > 0) {
         return (a * power(a, b - 1));

      }
      else if (b != 0 && b < 0) {
         return (1 / power(a, -b));


        }
      else {
         return 1;
      }

   }
```

1.2 Answer part b: tightest big-o complexity

The tightest Big-O complexity for this function is O(b) since it the recursive function will be called b times i.e it will be repeated four times. This is because b is reduced by one for every call.

1.3 Answer part c: maximum depth of the call stack functions

The maximum call depth of the function will again b. As it's going to repeat itself b times.

2 Question 2

Consider the following code for Euclid's algorithm (see lecture 4 or §2.4 of the textbook):

```
  int gcd(int m, int n) {
while(n != 0) {
int rem = m % n; m = n;
n = rem;
}
return m; }
```

Write the code using recursion

2.1 Recursive code

```
  public static int gcd(int m , int n){
  if(n == 0){
    return m;
  }
  else{
    return gcd(n , m%n);

  }
  }
```

3 Question 3

Order the following functions by growth rate: N,N, N1.5, N2, NlogN, NloglogN, Nlog2 N, Nlog(N2), 2/N, 2N , 2N/2, 37, N2 log N, N3. Indicate which functions grow at the same rate. Show your work. (15 P.)

3.1 Answer to question 3

The order is as follows from the lowest growth ratio to the highest growth ratio:

$$\frac{2}{N} < 37 < \sqrt{N} < N < NNloglogN < NlogN = Nlog(N^2)) < Nlog^2(N) < N^{1.5} < N^2logN < N^3 < 2^{\frac{N}{2}} < 2^N$$

The reason why $N\log N$ will have the same growth rate as $N\log N^2$ is because:

$$N\log(N^2) = 2N\log N = O(N\log N)$$

Whereas the other functions don't grow at the same rather.

1. $2^{\frac{N}{2}} \neq 2^N$**.**

$$\lim_{N\to\infty}\frac{2^{\frac{N}{2}}}{2^N} = \lim_{N\to\infty}\frac{2^{\frac{N}{2}}}{2^{\frac{N}{2}} * 2^{\frac{N}{2}}} = \lim_{N\to\infty}\frac{2^{\frac{N}{2}}}{2^N} = \lim_{N\to\infty}\frac{1}{2^{\frac{N}{2}}} = 0$$

2. $N * (\log N)^2 * N = N[\log N^2] \neq N\log N\log N$

3. $N^{1.5} \neq N(\log N^2)$ because as:

$$\lim_{N\to\infty}\frac{N^{1.5}}{Nlog^2N} = \lim_{N\to\infty}\frac{N^{0.5}}{\log^2 N} = \lim_{N\to\infty}\frac{0.5N^{-0.5}}{2 * logN\frac{1}{N}} = \lim_{N\to\infty}\frac{0.25N^{0.5}}{logN} = \infty$$

For the cases mentioned abbve we needed limits to calculate/compare there growth rate but for the other cases it's fairly simple to compare the growth rates. Growth rate is nothing but the derivative of the function given and then we compare their derivatives.

4 Question 4

A core part of many pattern recognition algorithms involves comparisons between pairs of points. For example, in optical character recognition, a pattern such as a alpha-numeric character might be compared to templates to identify that pattern. For the following questions Suppose we have K templates and N patterns. a) What is the running time complexity for an algorithm that finds the closest template for every pattern by comparing every template? (2 P.) b) Suppose each comparison (with appropriate pre-processing) takes 0.05 ms and we have K = 5000 tem- plates in our character database (to cover all the different fonts). How long would it take to recognize a typical page of text assuming N=1000 characters per page? (2 P.) c) Now suppose we implement a new algorithm that using a data structure that allows us to narrow down the possible set of closest templates by half at each iteration. What is the running time of this new algorithm? (1 P.) d) What is the new average time per page? (2 P.) e) Suppose we increase the size of our character database to 10,000. In terms of percentage of the original running time, how much do each of the two algorithms increase? (4 P.)

4.1 Running time complexity...

Let's say K templates and then N are the patterns. We compare eack K to N then the time complexity is given by $O(KN)$

4.2 Time to recoginze a page

We have 5000 templates, 1000 characters and it takes 0.05ms to compress it. So the time to recognize a typical page of text is

$$5000 * 1000 * 0.05 = 2.5 * 10^5 ms = 250s$$

4.3 Time when algorithm narrows by half every iteration

Since it's cut down by half every iteration the time complxity for this particular algorithm is given by $O(N \log K)$

4.4 New time to recognize a page

$$1000 * \log_2 5000 * 0.05 = 614ms$$

4.5 Percentage of running time increase

Run-time for $O(KN)$ increases by 100% and for $O(N)logK$ it is 8.14.

5 Question 5

Q5. An algorithm takes 2 ms for input size 100. How large a problem can be solved in 1 min if the running time is the following (assume low-order terms are negligible): a) linear(2P.) b) O(NlogN)(4P.) c) quadratic(3P.) d) cubic(3P.)

5.1 a: Linear time complexity O(N)

Input size is 100 and time taken for 100 is 2ms. Since, the algorithm takes 2ms for input size of 100. It will take 1ms for input size half of 100 which is 50. Maximum size. for 1 min processing time with the linear time complexity(O(N)) is

$$\frac{100}{2} * 60000 = 3000000$$

5.2 b: Max size for $O(N \log N)$ complexity

We get the maz size using the graph

$$\frac{N \log N}{100 \log N} = \frac{60000}{2} \to N = 1000000$$

5.3 c: time complexity is quadratic $O(N^2)$

In 2 secondsd a size of 100 is processed for 1 it's 50. Let the size of the problem to be solved in 1min be n.

$$\frac{n^2}{100^2} = \frac{60000}{2ms} \rightarrow n = 17320.50$$

5.4 d: cubic time complexity $O(N^3)$

Similar to c we let the max size be n.

$$\frac{n^3}{100^3} = \frac{60000}{2ms} \rightarrow n = 3107.23$$

6 Question 6

. For each of the following program fragments, provide the big-O bound of its running time. (What is a sensible measure of the problem size here)? For each problem, provide a brief justification of your answer.

6.1 a: O(n)

loop is going to run till n so as n increase so does the running time. That's why the running time complexity has to be $O(N)$

6.2 b: $O(N^2)$

There are tow loops which each run for length n so the total time to run this function varies with $n * n$ which is n^2

6.3 c: $O(N^3)$

There are a two loops the inner one runs for the length of n*n and the outer loop runs till length n. So all-combined the run loops till length n*n*n.This makes the time complexity as n^3

7 Question 7

Again consider the following program fragments and provide the big-O bound of its running time. For each problem, provide a clear justification of your answer, and include any formulas you use. Part of the chal- lenge in these problems is to systematically and concisely represent the number of times each loop executes. Problems with the correct answer but without clear (and correct) justification will only be given partial credit.

7.1 a: time-complexity for function a

To figure out the time complexity we first look at the code peice by piece `sum = 0;` This executes at an independent time let's say.

Now looking at the outerloop:
`for(int i = 0; i < n; i++)` takes time which varies with n.

Looking at the innerloop.
`for(int j; j< 1; j++)` this line of code depends on iteration value of i so this will execute with respect to n/2.

`sum++;` executed c times The total running time is: $C + \frac{C}{2} * n^2$

7.2 b: time-complexity for function b

Now the Big-O bound of it's running time $= O(n^2)$

Very similar to part a we calculate the time needed for each line to run The first line is run at a constant time. The outerloop runs n times. The second outerloop runs i*i times. i itself depends on n so the it executes this loop with respect to $n^2/2$times. The innermost loop runs with respect to j and j depends on $n^2/2$ so this loop is run $n^2/4$ times. The last line is run at constant time c just like in part a. So the total run time for this code is:

$$c + n * \frac{n^2}{2} * \frac{n^2}{4} * c \rightarrow c + \frac{c}{8} * n^5$$

The Big-O bound of its running time is $O(n^5)$

7.3 c: time-complexity for function c

Doing the same process for c now `sum =0;` this takes a constant c time.

the outermost loop `for(int i =0; i < n; i++)`. This loop runs n times.

The middle-loop `for(int j=0; j< i*i; j++)`
j depends on i^2 i depends on n. This particular block has a running time $\frac{n^2}{2}$ times on average.

the if statement also runs at constant time c.

The last loop only runs if the if statement is true. For the worst case scenario it's run all n times.

the last statement also runs at a constant rate a constant time c.
Total time is:

$$c + n * \frac{n^2}{2} * \frac{n}{2} * c = c + \frac{c}{4} * n^4$$

Big-O bound of its running time = $O(n^4)$

——————————————-END OF ASSIGNMENT——————————