Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Section:\_\_\_\_\_\_\_\_

CSE 3318 - Homework 2

Total: 100 points Topics: time complexity of loops, good information delivery, run code to see behavior.

Write all your answers in a pdf or docx called 3318\_H2.pdf and submit it in Canvas.

**Put your NAME and section at the top of the first page.**

You can handwrite the answer on paper and scan the paper as pdf, but make sure your handwriting is neat and can be read. If the scan is not legible, or did not capture all of your answers on a page, or it is missing a page, you will NOT have a chance to resubmit after the deadline. DOWNLOAD it back from CANVAS and CHECK that the SUBMITTED PDF is good.

You can write your answers on white paper, but they must match the required format (e.g. the tables and the components for the time complexity of loops calculation).

You do not need to ‘fit’ your answer in the given spacing. You can use as much space as your need.

You can run the code from this homework on your machine. You can also run it on omega or the VM, but that is optional, not mandatory. Do NOT run it on an online coding server (that provides and IDE) as that may not produce the expected clock time.

**Q1.** (**8** points)

a) (3pts) What is wrong in the statement (something is wrong with its time complexity):

“Function void helper(int X, double T); has time complexity O(N)”

The time complexity is not in terms of X or T, which are the provided parameters

b) (3pts) Given the loop: for(k = 1; k<T; k++) {….} where TC1iter(k) = O(k2) is this dependent or independent?

It is dependent since the loop variable is part of the time complexity of one iteration

c) (2pts) True / False

If TC1iter(k) of a loop is independent of the loop variable, k, then it is correct to calculate the time complexity of that loop by multiplying the number of loop repetitions with TC1iter(k).

**Q2.** (**6** points) Find the dominant terms and write Θ for each of the functions below. (Pay close attention.)

NM + M3+ 500M2+ NM2+ 106 = O(\_\_\_\_M3+NM2\_\_\_\_\_\_)

100N3+20N2+15lgM +5N = O(\_\_\_\_\_\_N3+lgM\_\_\_\_\_\_\_\_\_)

**Q3.** (65 points) Show your work as done in class. Whenever a summation is needed, clearly write the summation and the closed form for it.

(See cheat sheet for summations. E.g.  = , has closed form:.)

a) (5 points)

Assume that void mistery(int X); has time complexity O(X)

Fill in the time complexity of the function call: mistery(8);

TC**mistery(8)** = O( \_\_\_\_\_\_\_\_ )

­­

b) (10 points)

for(k = 1; k <= N; k=6\*k)

for(t = 0; t < S; t++)

printf("D");

for-t: TC1iter(\_t\_\_) = \_\_\_\_O(S)\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

for-k: TC1iter(\_k\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

c) (10 points)

for(k = N; k >= 0; k=k-1)

for(t = 0; t <= k; t = t+9)

printf("C");

for-t: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

for-k: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

d) (10 points)

if ( check(T)) { // O(T)

doThis(M) // O(lgM)

}

else {

doThat(T) // O(T)

}

Final answer: worst O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) best O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) in general O(\_\_\_\_\_\_\_\_)

Show your work as done in class.

e) (10 points)

Assume that void some\_fct(int X); has time complexity Θ(X)

for(k = 0; k <= N; k = k+1)

some\_fct(k);

TC**some\_fct(k)** = Θ( \_\_\_\_\_\_\_\_ ) (time complexity of the function call)

for-k: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

f) (10 points) Assume that void some\_fct2(int N, int k); has time complexity Θ(N2)

for(k = 1; k <= M; k++) {

some\_fct2(k,N);

}

TC**some\_fct2(k,N)** = O( \_\_\_\_\_\_\_\_ ) (time complexity of the function call)

for-k: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

g) (10 points)

Assume that void some\_fct3(int N); has time complexity Θ(N3)

for(k = 1; k <= M; k=k+1) { ­­

some\_fct3(S);

for(t = 0; t < M; t = t+1)

printf("X");

}

TC**some\_fct3(S)** = Θ( \_\_\_\_\_\_\_\_ ) (time complexity of the function call)

for-t: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

for-k: TC1iter(\_\_\_) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependent / independent of loop variable

Change of var: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/ repetitions \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Closed form: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**Q4.** (**21** points)

1. (16 points) This problem shows how code that looks similar can have significant differences in the time it takes to run for large values of the data input. The code provided below has main() and 4 other functions. You will need to

* Calculate O() for each function (except main()) and fill it in the tables below. You do NOT need to show your derivations for computing O, give just the final answer.
* Run the code and observe the actual time (seconds/minutes/hours). You can simply look at your watch to see the how long it takes. To do this, make changes to main() to run each one of those functions one at a time with the values of N (from the tables), and see the actual time (e.g. seconds) it takes and report it in the table by category: **“less than 1 second”, “less than 1 minute”, “1 to 15 minutes”, “between 16 minutes and 1 hour”, “1 to 2 hours”, “more than 2 hours”. When you run the code use a desktop or laptop. DO NOT use an online complier as that may not give the expected time.**
* If you stop the execution of a program before it terminates, you can calculate an estimate of how long it would have taken if you print the value of the outer loop variable as progress report. If only 100 out of 1000 values are printed after 1 hour, it did 10% and so it would take another 9 hours to finish.

#include <stdio.h>

void runtime\_increment(int N);

void runtime\_print(int N);

void runtime\_print\_long(int N, char\* long\_str);

int main(void){

int L = 1000, i;

char str[L+1];

/\* even if str is not used in runtime\_increment and runtime\_print, keep it to make the comparison for runtime\_print\_long fair. \*/

for(i=0; i<L; i++){

str[i] = 'A';

}

str[i] = '\0';

// call the functions here and record the time they take.

runtime\_increment(100);

//runtime\_print(100);

//runtime\_print\_long(100, str);

}

void runtime\_increment(int N){

int i, k, t, res = 0;

for(i = 1; i <= N; i=i+1)

for(k = 1; k <= N; k++)

for(t = 1; t <= N; t++)

res = res + 1;

}

void runtime\_print(int N){

int i, k, t;

for(i = 1; i <= N; i=i+1)

for(k = 1; k <= N; k++)

for(t = 1; t <= N; t++)

printf("A");

}

void runtime\_print\_long(int N, char\* long\_str){

int i, k, t;

for(i = 1; i <= N; i=i+1)

for(k = 1; k <= N; k++)

for(t = 1; t <= N; t++)

printf("%s\n", long\_str);

}

Continue to next page

Hint: If the work a function does depends on the size of the data it works with, the TC should reflect that. (You can assume printf prints a string by printing one letter at a time.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Values of N | | | |
|  | 10 | 100 | 300 | 1000 |
| runtime\_increment  O(\_\_\_\_\_\_\_\_\_\_\_\_\_) |  |  |  |  |
| runtime\_print  O(\_\_\_\_\_\_\_\_\_\_\_\_\_) |  |  |  |  |
| runtime\_print\_long  O(\_\_\_\_\_\_\_\_\_\_\_\_\_) |  |  |  |  |

// When compiling this function you need to link the math library, lm, needed for pow(). Eg: **gcc main.c –lm**

void runtime\_pow(int N){

int i, res = 0;

for(i = 1; i <= pow(2.0, (double)N); i=i+1)

res = res + 1;

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Values of N | | | | |
|  | 10 | 15 | 20 | 25 | 30 |
| runtime\_pow  O(\_\_\_\_\_\_\_\_\_\_\_\_\_) |  |  |  |  |  |

Answer the questions below based on what you learn from this experiment.

1. runtime\_print has the same code as runtime\_increment except that instruction ‘res = res+1’ was replaced with ‘print…’ instruction. Why does runtime\_print take more time to run? (Purpose: see that not all Θ(1) instructions are the same. Note the big slowdown that comes from function call and accessing the screen versus a simple increment.)

**Answer: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. runtime\_print\_long has the same code as runtime\_print except that it prints the string given as argument instead of the single letter ‘A’. Why does runtime\_print\_long take more time to run?

(Purpose: see the actual time cost of printing one letter versus a long string. Use that to help you estimate the correct time complexity for printf when printing a string.)

**Answer: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. After you record the time for runtime\_increment, pay attention to how the performance gets worse as N gets larger. Do you think a program with such time complexity would be feasible for an application where N is a million?

**Answer: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. After you record the time for runtime\_pow, notice how much faster the performance deteriorates (i.e. it takes too long to run even for small values of N such as 20). Compare that with runtime\_increment and runtime\_print (compare both the actual time, and the time complexity).

This is for your own reflection on this topic. No written answer needed here.

**b)** (5 points) Look at the program below.

**Which of the three functions above (runtime\_increment, runtime\_print and runtime\_pow) has time performance ‘closer’ (or more similar) to that of the runtime\_rec in the code below?**

Note that you do not need to compute the time complexity for runtime\_rec. We did not cover that yet. Use other methods (e.g. look at the actual time it takes to execute for different values of N and see to which function from above).

(Purpose: estimate the time complexity of the recursive function runtime\_rec. Later we will learn how to mathematically derive the time complexity for recursive functions as well.)

**ANSWER:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

void runtime\_rec(int N, char \* str){

if (N==0) return;

str[N-1] = 'L';

runtime\_rec(N-1, str);

str[N-1] = 'R';

runtime\_rec(N-1, str);

}

int main(int argc, char\*\* argv) {

int N = 0;

char ch;

char str[100];

printf("run for: N = ");

scanf("%d", &N);

str[N] = '\0'; //to use it as a string of length N.

printf("runtime\_rec(%d)\n", N);

runtime\_rec(N, str);

}