Name_____Section:_____

CSE 3318 - Homework 1

Total: 100 points Topics: time complexity of loops, C review: pointers, dynamic memory allocation (with malloc() or calloc()), 1D arrays, pass-by-reference, Valgrind and debugging demo.

What to submit

Submit 3 items: a video file, the updated a C program (scores.c), and a pdf with written answers.
Submit demo video under Homework1-Video in Canvas.
Submit TC.pdf and scores.c in Homework1-Code-TC in Canvas

Video (11 pts)

File: demo_sum.c
Record and submit in Canvas a short video that shows you using a Unix system (omega, VM, Ubuntu) with Valgrind and debugging the program demo_sum.c .

a) (5pts) On omega or VM or Ubuntu
    a. compile with -g
    b. run with Valgrind
b) (6pts) show the debugging steps listed below. Use either an IDE (VSCode/Code::Blocks/onlinegdb/etc) or the command line debugger from Unix. Set a break point, start running the program in debug mode:
    a. examine value of some variables.
    b. Step line-by-line a couple of times,
    c. set a break point later on (e.g. outside of a loop) and use “continue” to run until it meets that break point.

Check that the video is clear, not blurry. We have to be able to read the text on the screen (commands, value of variables being examined, lines of code, etc).
See a sample video in Canvas. You can use the Studio tool in Canvas (in left menu) to record a screen capture video.
I used Code::Blocks for debugging. It does not have to be Code::Blocks for you. Another IDE (even online) or gdb in command line are also good. The important thing is to show that you can start debugging.

Program(15 pts)

Purpose: 1D array practice, dynamic memory allocation (i.e. using malloc()/calloc()), test functions.

In scores.c, fill in the code for function:

```
int* get_scores_below(int thresh, int sz_arr, int * arr, int* sz_res)
```

It should return a dynamically allocated 1D array containing values from arr that are strictly less than the given thresh value. This array must be allocated on the heap (malloc/calloc), and its size, sz_res, is updated using pass-by-reference.

If the new array is empty (there are no elements that are less than the threshold) the function returns NULL and sets sz_res to 0.

Examples:

- For threshold 60 and array {92, 57, 100, 95, 38, 10, 85, 91, 20} it should return the array {57, 38, 10, 20}
- For threshold 50 and array {22, 45, 30, 49, 38} it should return the array {22, 45, 30, 49, 38}
- For threshold 22 and array {22, 45, 30, 49, 38} it should return NULL (and not allocate any space). Here there is no value strictly smaller than the threshold.

The function comments in the scores.c file describe the function behavior. Read them.

Hints:

- File scores.c runs a few hardcoded tests for this function. The elements in the result array should appear in the same order they were in the original array. If not, the tests for correctness will fail.
- It is ok to iterate over the input array 2 times:
    - iterate over it once to count how many values are smaller than the threshold, so that you know the size of the result array.
    - allocate the space for the result array
    - iterate again to copy the needed values into the result array
- Pay attention to the indexes in the input and result array. Trace the data on paper! If needed use a debugger to see the values of the indexes (or print them) and compare those with what you expect on paper.

Grading criteria subtask 1 (15 pts)

3 pts – allocate dynamic memory for the exact number of elements smaller than threshold
3 pts – if result array is empty sets sz_res to 0 and returns NULL
3 pts – correctly copy elements in order in which they are in the array
3 pts – sz_res is correctly updated (when the result array is not empty)
3 pts – no Valgring errors

TC questions (74 pts)

Write all your answers for this part in a pdf or docx called TC.pdf and submit it in Canvas.

Fill in NAME and section at the top. Make your answers visible: use a different color or highlight them.

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.

Q1. (**9** points)

a) (3pts) What is wrong with the time complexity in the statement below?

"Function `void helper(int X, double T);` has time complexity O(V)"

Find the dominant term(s) and write O for each of the math functions below.

b) (3pts) $NS + N^3 + 500N^2 + SN^2 + 10^6 = O(\_\_\_\_\_)$

c) (3 pts) $100S^3+20S^2+15\lg N +5S = O(\_\_\_\_\_)$

Q2. (65 points) Show your work as done in class. Clearly write the summation and its closed form.

(See cheat sheet for summations. E.g. $1+2+3+\ldots+(n-1)+n = \sum_{t=1}^{n} t$, has closed form: $\frac{n(n+1)}{2}$.)

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);`

$TC_{\texttt{mistery(8)}} = O(\quad\_\_\_\_\_\quad)$

b) (10 points)

```
for(k = 1; k <= N; k=6*k)
   for(j = 0; j < S; j++)
      printf("D");
```

Iter (e)	j	$TC_{1iter}(j)$ =
0		
1		
2		
3		
…		
e	j =	
…		
p		

change of variable: j = _____

j_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Iter (e)	k	$TC_{1iter}(k)$ =
0		
1		
2		
3		
…		
e	k =	
…		
p		

change of variable: k = _____

k_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Final answer: O( )

c) (10 points)

```
for(j = N; j >= 0; j-- )
   for(u = 0; u <= j; u = u+9)
      printf("C");
```

Iter (e)	u	$TC_{1iter}(u)$ =
0		
1		
2		
3		
…		
e	u =	
…		
p		

change of variable: u = _____

u_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Iter (e)	j	$TC_{1iter}(j)$ =
0		
1		
2		
3		
…		
e	j =	
…		
p		

change of variable: j = _____

j_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Final answer: O( )

d) (10 points)

```
if ( check(T)) {  // O(T)
    doThis(M, T)  // O(T^2).
}
else {
    doThat(T)  // O(1)
}
```

Final answer: worst O(_____) best O(_____) in general O(_____)

Show your work as done in class.

Extra, self practice (not part of homework. Do not write the answer for it here.) What would your answer be if doThis(M, T) had time complexity O(lgM)?

e) (10 points)

Assume that `void some_fct(int X);` has time complexity Θ(X) . Solve

```
for(v = 0; v <= N; v = k+1)
   some_fct(v);
```

Fill in: $TC_{\texttt{some\_fct(k)}}$ = Θ(_____)

Iter (e)	V	$TC_{1iter}(v)$ =
0		
1		
2		
3		
…		
e	v =	
…		
p		

change of variable: v = _____

v_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Final answer: O( )

f) (10 points)

Assume that `void some_fct2(int N, int k);` has time complexity $\Theta(N^2)$

Solve:

```
for(k = 1; k <= M; k++) {
   some_fct2(k,N);
}
```

Fill in:

$TC_{\texttt{some\_fct2(k,N)}}$ = O(_____)

Iter (e)	k	$TC_{1iter}(k)$ =
0		
1		
2		
3		
…		
e	k =	
…		
p		

change of variable: j = _____

j_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Final answer: O( )

g) (10 points) Assume that `void some_fct3(int N);` has TC(N) = Θ(N^3) Find TC for:

```
for(k = 1; k <= M; k=k+1) {
   some_fct3(S);
   for(t = 0; t < M; t = t+1)
      printf("X");
}
```

Fill in: $TC_{\texttt{some\_fct3(S)}}$ = Θ(_____)

Iter (e)	t	TC_{1iter}(k) =
0		
1		
2		
3		
…		
e	t =	
…		
p		

change of variable: t = _____

t_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Iter (e)	k	TC_{1iter}(k) =
0		
1		
2		
3		
…		
e	k =	
…		
p		

change of variable: k = _____

k_last = _____ p = _____

Sum:_____

_____

_____

_____

TC of entire loop is O(_____)

Final answer: O(_____)

Q4. (**21** points)

a) (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.

You are given 2 programs:

- runtime_a.c
- runtime_rec.c

You will need to:

- Calculate O() for each function (except for main() and runtime_rec) 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). See the actual time (e.g. seconds) it takes and report it in the table.
- To record the time you can use your watch, but I recommend to use the "time" command in shell. E.g.:
  `time ./a.out 2 100`

You need to link the math library at compilation (with -lm). To compile:

```
gcc -o runtime_a runtime_a.c -lm
```

To run the runtime_a executable:

```
./runtime_a option N
```

Where `option` can be 1,2,3 or 4, corresponding to the 4 functions: `runtime_increment, runtime_print, runtime_print_long, runtime_pow.`

Sample run commands:

```
./runtime_a 1 100
```

Will call runtime_increment(100)

```
./runtime_a 2 300
```

Will call runtime_print(300)

```
./runtime_a 3 30
```

Will call runtime_print_long(30)

```
./runtime_a 4 10
```

Will call runtime_pow(10)

To display the time use the "time" command on a Unix system (omega/VM). Report the "real" time. E.g. when running:

```
time ./a.out 1 100
```

the "time" command prints:

```
real    0m0.012s
user    0m0.006s
sys     0m0.000s
```

and I would report ***0.012s*** or ***less than 1 second***.

Hint1: The TC of a function definition should depend on the size of the data it works with.

Hint2: 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(_____)				

Fill in information for the runtime_pow(). This function will be slower, so we will only try small values of N. If it becomes too slow with N=25, you do not have to try N = 30.

Function	Values of N				
	10	15	20	25	30
`runtime_pow` O(_____)					

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

1. `runtime_print` has the same code as `runtime_increment` except that instruction '`res = res+1`' was replaced with '`printf("A")`' 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: _____
   _____
2. `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: _____
   _____
3. 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: _____
4. 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 runtime_rec.c program.

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?

You do not need to compute the formula for time complexity for runtime_rec. We did not cover that yet.

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.

Compare the time it takes each of these functions for the same value of N.

Compile `runtime_rec.c` :

```
gcc -o runtime_rec runtime_rec.c
```

E.g compare the time for:

```
time ./runtime_rec 10
```

with the time for each:

```
time ./runtime_a 1 10
time ./runtime_a 2 10
time ./runtime_a 4 10
```

and see to which one it is most similar.

Repeat that for other values of N (e.g. for N = 20).

ANSWER:_____