Name Apar Pokhrel Section: 002

CSE 2320 - Homework 1

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

Q1. (6 points) Bob and Alice are solving practice problems for CSE 2320. They look at this code:

```
for(i = 1; i <= N; i = (i*2)+17 )
     for(k = i+1; k <= i+N; k = k+1) // notice i in i+1 and i+N
          printf("B");
```

Alice says the loops are dependent. Bob says they are not dependent. Who is correct? Bob

What do you think? Are they dependent or not dependent? They are Not dependent.

Justify your answer: The iteration of the inner for loop doesn't depend on the value of i. Regardless, the inner for loop runs $(i+N-i-1) = N-1$ time

Q2. (12 points) Answer the following questions. **Give the EXACT answer as a function of N as done in class** (i.e. show if it is rounded down or up, and/or if it has a + or – a constant).

```
for(i=0; i<=N; i=i+4) {
    if (i%3==0)
        printf("Y");
    else
        printf("Z");
}
```

(b) and c) are the type of my so-called interesting questions that build upon what we did in class.)

Show your calculations for each answer. Add more space if you need to.

Hint: you can answer c) if you solved a) and b).

a) How many times does the loop iterate? $1+\lfloor N/4 \rfloor$

Values of i : 0, 4, 8, 12 ..... $i_{last} \le N$

$i = 4e$; where $e = 0, 1, 2, 3 \ldots\ldots p_{last}$.

$i_{last} \le N$

$4p \le N$

$\Rightarrow p \le \frac{N}{4}$

$\therefore p = \lfloor N/4 \rfloor$

b) How many Y are printed? $1+\lfloor N/12 \rfloor$

No. of Y's printed $= 1 + \lfloor N/(4\times 3) \rfloor = 1 + \lfloor N/12 \rfloor$

c) How many Z are printed? $\lfloor N/4 \rfloor - \lfloor N/12 \rfloor$

No. of Z's printed = Total iterations − No. of Y's printed

$= 1 + \lfloor N/4 \rfloor - 1 - \lfloor N/12 \rfloor$

Q3. (30 points) See the end of this document for an example of how to do and show your work for the time complexity of nested loops. Show your summations to the right of the code (as shown in the example on the last page), or write them separately.

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

a) (15 points)

i) `for(i = 1; i <= N; i=i+1)` $\rightarrow \sum_{i=1}^{N} Si = S\sum_{i=1}^{N} i = S\left[\frac{N(N+1)}{2}\right] = S\left[\frac{N^2+N}{2}\right]$

ii) `for(k = 1; k <= S; k++)` $\rightarrow \sum_{k=1}^{S} i = S \times i = Si$

iii) `for(t = 1; t <= i; t++)` $\rightarrow \sum_{1}^{i} 1 = i \times 1 = i$

`printf("D");` $\rightarrow 1$

The answers below should be with respect to the whole code piece.

T(N, S) = $S + 2S + 3S + 4S + 5S + \dots + NS$

Closed form: $\frac{SN^2}{2} + \frac{SN}{2}$

Dominant term(s): $\frac{SN^2}{2}$

Θ(SN^2)

If you use any variable change, show your work as I did in the example at the end of this homework.

If the summations are complicated show your work below.

iii) Values of t : $1, 2, 3, 4 \dots t_{last} \le i$

$\Rightarrow \sum_{1}^{i} 1$ (roughly, without considering printf("D") iteration).

ii) Values of k : $1, 2, 3, 4 \dots k_{last} \le S$

$\Rightarrow \sum_{k=1}^{S}$

i) Values of i : $1, 2, 3, 4 \dots i_{last} \le N$.

b) (15 points) Fill in the table and T( ).

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

$\sum_{i=1}^{N} (\lg N)(\lg i) = \lg N \sum_{i=1}^{N} \lg(i)$

ii)
```
for(k = 1; k <= i; k = 2*k)
```

$\sum_{e=0}^{\lg i} \lg N = (\lg N)(\lg i)$

iii)
```
for(t = 1; t <= N; t = 2*t)
```

$\sum_{e=0}^{\lg N} 1 = \lg N$

```
printf("C");
```

T(N) = $\lg N \left[\sum_{i=1}^{N} \lg i \right]$

Note: you do NOT need to compute closed form and Theta (Θ) for this problem. Leave the final answer as a summation (not in closed form).
To those interested, a closed form and Θ for this summation can be found using the approximation by integrals (covered in Summation slides).

If you use any variable change, show your work as I did in the example at the end of this homework.

Values of t : $1, 2, 4, 8, \ldots, t_{last} \le N$

$t = 2^e$

Values of e : $0, 1, 2, 3, 4 \ldots, p$

$\Rightarrow \left. \begin{array}{l} t_{last} = N \\ t_{last} = 2^e \end{array} \right\} 2^p = N$

or $p = \lg N$ [Taking lg on both]

ii)

Values of k : $1, 2, 4, 8, \ldots, k_{last} \le i$

$k = 2^e$

Values of e : $0, 1, 2, 3, 4 \ldots, p$

$\Rightarrow \left. \begin{array}{l} k_{last} = i \\ k_{last} = 2^p \end{array} \right\} 2^p = i$

Taking lg on both side

$p = \lg i$

Q4. (8 points) Solve the summation below:

$\sum_{sz} sz = \sum_{e=0}^{\frac{n-10}{15}} (10 + 15e) = \sum_{e=0}^{\frac{n-10}{15}} 10 + \sum_{e=0}^{\frac{n-10}{15}} 15e = \frac{2(n-10)}{3} + \frac{(n-10)(n+5)}{30}$

$= \frac{20(n-10) + (n-10)(n+5)}{30}$

$= \frac{(n-10)[20+n+5]}{30}$

$= \frac{25n+n^2-250-10n}{30}$

$= \frac{n^2+15n-250}{30}$

Closed form: $\frac{n^2}{30} + \frac{15n}{30} - \frac{25}{3}$

Dominant term with multiplication constant: $\frac{n^2}{30}$

$\Theta(n^2)$

Q5. (14 points) Write code that has the time complexity below and after that show why it has that complexity (that is, take your code **and derive the time complexity for it**). The code must have **2 nested loops** (a total of two loops, one nested inside the other) and **must NOT be recursive.**

You can use any C functions available in the C library.

Write the answer code in the pdf, but it should be proper C code (if copy/pasted in a C file it should run).

$T(n) = 1 + 3^1+3^2+3^3+ \ldots +3^n$ $= 3^0 + 3^1 + 3^2 + 3^3 + \ldots + 3^n = \frac{3^{n+1}-1}{2} \Rightarrow \Theta(3^n)$ [roughly]

As a verification step, derive the time complexity of your code to check that you get the same as above.

```c
#include <stdio.h>

int main (void)
{
    int k, l;
    for (k = 1; k <= n; k = k*3)
    {
        for (l = 1; l <= k; l++)
        {
            printf ("Hello!");
        }
    }
    return 0;
}
```

Values of $k = 1, 3, 3^2, 3^3, 3^4, \ldots, k_{last}$

$k = 3^e$, where

$e = 0, 1, 2, 3 \ldots\ldots p$

$\Rightarrow 3^p \le n$

$\log_3(3^p) \le \log_3(n)$ [Taking $\log_3$ on both sides]

$\Rightarrow p \le \log_3(n)$

$T(N) = \sum_{e=0}^{p} k = \sum_{e=0}^{\log_3(n)} 3^e = 3^0 + 3^1 + 3^2 + \ldots + 3^{\log_3 n}$

$= 1 + 3^1 + 3^2 + \ldots + 3^n$

Q6. (6 points) Find the dominant terms and write Θ for each of the functions below. (Pay close attention.)

(a)

$N^3 + 500N^2 + NM + 10^6 = \Theta(N^3 + NM)$

$100N^3 + 20N^2 + 15M + 5N = \Theta(N^3 + M)$

M and N are independent variables. The value of M can be much larger than the value of N and vice-versa. i.e. (it is possible for) $NM > N^3$ or $M > N^3$

Q7. (13 points)

a) (12 points) Run each piece of code below and record the actual time (e.g. seconds) it takes. Use a desktop or laptop. DO NOT use an online complier. **Pay attention (and draw your own conclusions)** to the following issues. You do NOT need to write down your conclusions.

1. `runtime_print` has the same code as `runtime_increment` except that instruction '`res = res+1`' was replaced with '`print...`' instruction. Notice how this affects the time the code takes to run.
2. After you record the time for `runtime_increment`, pay attention to how the performance gets worse as N gets larger.
3. After you record the time for `runtime_pow`, note 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 the other 2 functions (compare both the actual time, and the time complexity).

In the table below the code fill in the time complexity (as Θ) and the approximate "clock time" each function takes to run. You do NOT need to show your derivations for computing Θ. You also do not need to report the exact time. You can say: "< 1 sec", "few seconds", "few minutes", "more than 15 minutes"

```
void runtime_increment(int N){
     int i, k, t, res = 0;
     for(i = 1; i <= N; i=i+1)
          for(k = 1; k <= i; 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 <= i; k++)
               for(t = 1; t <= N; t++)
                    printf("A");
}
```

Function	Values of N			
	10	100	300	1000
runtime_increment Θ(N^3)	< 1sec	< 1sec	< 1sec	few seconds (1.123)
runtime_print Θ(N^3)	< 1sec	< 1sec	few seconds (11.679)	few minutes (7 min 9.585s)

// When compiling this function you need to link the math library needed for pow. E.g.: `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 Θ(2^n)	< 1sec	< 1sec	< 1sec	few seconds (2s42)	few minutes (1m 22.13s)

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?

<u>Note that you do not need to compute the time complexity for runtime_rec. We did not cover that yet.</u> 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).

The function runtime_pow() has the time performance closer to that of the runtime rec in the code.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void runtime_rec(int N, char * str){
    if (N==0) {
        //printf("%s\n", str);
        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);
}
```

Q8. (**11** points) The **grow_array.c** program implements an add function for a flexible array that grows as needed as discussed in class. It has 2 ways to compute the new size (double it or +15) and 2 ways to reallocate memory and copy (using realloc or malloc and manual copy with loop). That produces 4 variations in the implementation. You can choose which version you run, based on what you give as input for the 2 parameters. (It will become clear when you look at the code and run it. It will read 3 numbers from the user.)

Compare program performance for all 4 implementations, for different values on N. In particular, try powers of 10 for N (i.e. 1000, 10000, 100000,...) . For each value of N run it with all 4 combinations of the parameters for resizing and reallocation: (1 1), (1 0) , (0 1), (0 0).

a) (4 points) **Find the first value of N for which (0 0) is very slow, but (1 1) , (1 0) and (0 1) are still very fast.** For example, on my machine I got that for $N = 10^7$.

Value of N	Parameter used for resize type (1 or 0), where: 1 => double 0 => +15	Parameter used for reallocation type (1 or 0) where: 1 => realloc 0 => malloc and user copy	Time it takes to run Use: <1sec, few seconds, few minutes, more than 15minutes
N = 10^6	1	1	few seconds
	1	0	few seconds ·
	0	1	few seconds
	0	0	more than 15 mins ·

b) On my machine, for $N = 10^7$ the program took about a second or less with combinations (1 1), (1 0) , (0 1), but it took more than 20 minutes (I did not wait for it to finish) with (0 0).
Running with (1 1) uses the most efficient implementation (doubles the size and uses realloc).
Running with (1 0) doubles the size and so the time complexity should be $\Theta(N)$ and so should still be fast.
Running with (0 1) does not double the size. This should have $\Theta(N^2)$ but it runs in a few seconds while (0 0) took more than 20 minutes. That implies a difference of order of magnitude between them, but they should both be $\Theta(N^2)$.
(7 points) **How do you explain this behavior (that the theoretical analysis does not match the actual behavior)?** Hint: check the statistics about realloc that the program prints at the end of its run.
Give a brief and clear answer. In your answer underline the 3 most meaningful/relevant words. Out of the 7 points, 3 will be for the clarity/quality of the answer.

The use of realloc in this program required 3 iterations for a roughly a data size of 10^5 whereas the use of malloc () will require copying the data from 0 to the oldest capacity for each run of program. So, depending upon the input size, realloc and malloc along with their functionality influence this behavior.

You do NOT need to run any of this code on omega.