Time complexity and growth of functions summary

Tuesday, January 25, 2022 7:56 AM

1. O (BigOh) - motivation, calculate O for simple code
2. 2 or more variables in O expression
3. O - arithmetic
4. TC of functions
    1. Function definition
    2. Function calls
5. TC of loops
    1. for-loops : 4 possible cases
    2. while loops - same concept as for for-loops
    3. See Math review for summations in beginning of Time complexity of loops
6. TC for conditionals (if statements)
    1. give worst case,
    2. if possible, look at best and average cases as well
7. O, Θ (Theta), Ω (Omega)
    1. Definitions
    2. Usage
    3. Properties
    4. Same arithmetic as for O
8. Upper bounding and lowerbounding when we cannot compute an exact TC
    1. Can you find upper and/or lower bounds for:
        a. lg(1)+lg(2)+lg(3)+….+lg(N-1)+lg(N) Can you find a lower and/or upper bound for this?
        b. 1*2*3*…*N Can you find a lowerbound for this?
9. See Practice problems throughout this document and also on the web, on the Slides page: PRACTICE time complexity of loops, Solution to problems A6-A16 except A15 (pdf, docx)

From <https://ranger.uta.edu/~alex/courses/3318/slides.html>

Time complexity: BigOh - part 1

1. O-Functions, their growth, and how that affects the scalability of a program

a) The exact count of instructions is often a polynomial function of N (input size)
b) The term with the highest exponent dominates the time (see 31 years vs 1.6 minutes)
=> keep only that dominant term: drop lower order terms and drop constant
c) Examples
   a. $100N+3N^2+1000 = O(N^2)$
   b. $200N + 1000 = O(N)$
d) O(1): $35 = 35*N^0 = O(1)$; 100 = O(___)
e) We will be able to compute O of large piece of code using O of its smaller components .
f) After Assume We have programs P1, P2, P3, P4 and P5 , P6with the time complexities given in the table. Further assume that each program takes 1 hour to finish, when its input size is N = 1000. - new problem

	Program 1 O(N)	Program 2 $O(N^2)$	Program 3 $O(N^3)$	Program 4 $O(2^N)$	Program 5 $O(\log_2 N)$	Program 6 O(1)
N= 1000	1 hr	1 hr	1hr	1hr	1hr	1hr
N=10000 (input size is 10 times bigger)						

2. Intuition behind using O

Assume

Company A implemented function:
`void processArr_CA(int arr[], int N)` // where N is the number of elements in arr
that executes $100N+3N^2+1000$ instructions.
We will say that this function is $O(N^2)$.

Company B implemented function:
`void processArr_CB(int arr[], int N)` // where N is the number of elements in arr
that executes 200N+1000 instructions.
We will say that this function is O(N) .

The table below will help understand why $O(N^2)$ and O(N) are used instead of the exact instruction count.

Assume you run this function on a ***machine that executes 10^9 instructions per second***.

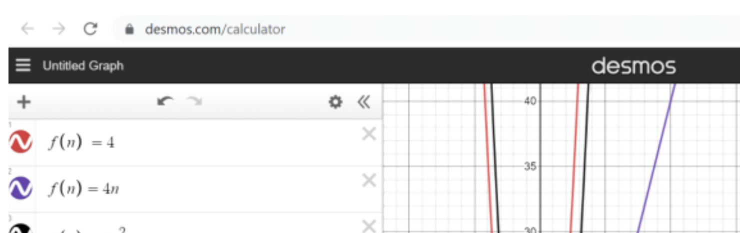
(Review: Sample time calculation: 10000 instructions will take: $10000/10^9 = 10^{-5}$ seconds)

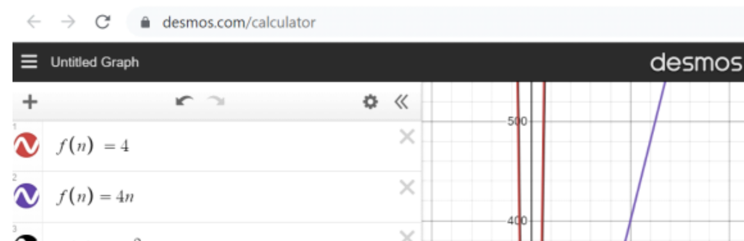
Values in table are approximations (not exact calculations).

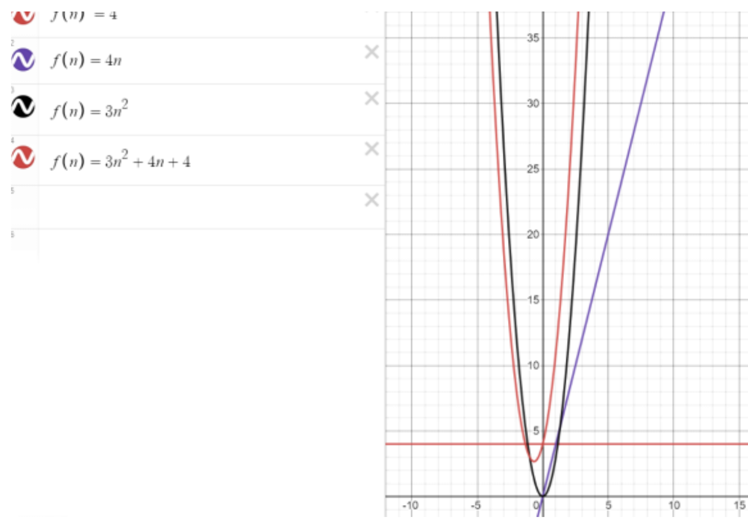
N	N^2	$3N^2$	100N	1000
10^4	Instructions: Time: 0.1sec	Instructions: Time: 0.3sec	Instructions: Time: 0.001sec	Instructions: Time: 10^{-6} sec
10^9	Instructions: Time: 31 yrs	Instructions: Time: ***95 yrs***	Instructions: Time: 100sec = ***1.6 min***	Instructions: Time: ***10^{-6} sec***

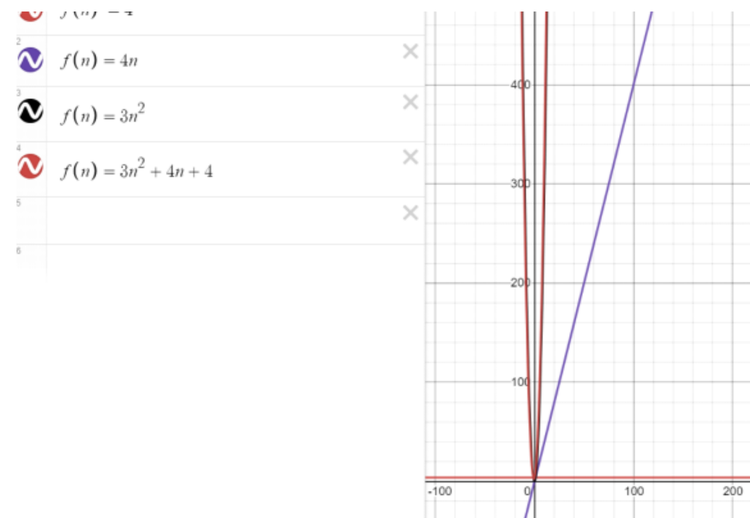
We care about behavior for very large input size (as N goes to infinity).

https://www.desmos.com/calculator









See the different shapes of the functions for each term, and see their growth.
See how the shape of the entire function, $3N^2+4N+4$, is determined by (looks mainly like) the function for the dominant term ,$3N^2$.

3. steps for identifying the dominant term

Steps

a) Identify dominant term: the term with the largest exponent
    a. There may be more than 1 dominant term, when we have multiple variables.
b) Remove all lower order terms (with smaller exponents)
c) Remove the constant of the dominant term

processArr_CA : $100N+3N^2+1000 = O(N^2)$

$100N+3N^2+1000N^0 = O(__)$

processArr_CB: $200N + 1000 = O(N)$

Practice:

a) $(5N^2)/2 + 1000N+500 = O(_____)$
b) $N^2 - 1000N+700 = O(_____)$

TC - multiple variables and O arithmetic

Sunday, January 23, 2022 7:31 AM

Getting started with O

1. O - behavior as data size is very large (goes to infinity)
2. Steps:
   a. Identify dominant term
   b. Drop lower order terms
   c. Drop constant of dominant term
   d. E.g.
3. O(1) - **always use O(1) for constants** (think $N^0=1$
   a. Do NOT say O(35), O(10), etc
4. Terminology:
   a. for O, I may say: "BigOh" or "order of" or "growth"
   b. E.g.: for $O(N^2)$ I may say: "BigOh of N^2", "order of N^2", "N^2 growth"
5. Practice:
   a. $N^2 + \log_2 N + 100 = O(__________)$
   b. $N^2 + N\sqrt{N} + 1000 = O(__________)$
   c. $T^4 + T^2\sqrt{T} + T^3 = O(________)$
   d. $\frac{1}{1000}N^2 + 100N = O(__________)$
   e. $N^2 - 1000N + 700 = O(_________)$

Multiple Variables

What if we have two or more arrays (or variables that control the repetitions)?

```
...
for(j=0; j<N; j++){
        printf("%d, ", users[j]);
}
for(k=0; k<T; k++){
        printf("%d, ", movies[k]);
}
...
```

O(N) + O(T) → O(N+T)

a) Solve:
   a) The above code has O(_____)
   b) N + M = O(______)
   c) $N^2 + M + N + 35 = O(_________)$
   d) $N^2M + M^2N = O(_________)$
   e) $N^2 + M^2 + N^2M^2 = O(_____________)$

b) Process: when multiple variables are present, consider ALL combinations of some variable(s) having a large value, and others having a small value. E.g. in order to compute O() for expression N + M I will look at:
   a) N is large, M is small => term N dominates term M
   b) N is small, M is large => term M dominates term N
   c) => I can see that I should keep both N and M => O(N+M)
   d) I should also consider the case when both M and N are large, and see if that would have a new dominant term.

c) Check your answers

O arithmetic

d) Add and keep the dominant terms:
   a) O(N) + O(S) = O(N+S)
e) Add and drop the lower order terms:
   a) O(N) + O(S+N^2) + O(U) = O(N+S+N^2+U) **= O(S + N^2 + U)** (we removed the lower order term N)
f) Multiplication:
   a) O(N)* O(k) = O(Nk)
   b) O(N)* O(1) = O(N)
   c) O(N)* 7 = O(N)
g) In a summation "pull O out": $\sum_{k=1}^{N} O(k) = O\left(\sum_{k=1}^{N} k\right) = O\left(\frac{N(N+1)}{2}\right) = O(N^2)$
h) If it is not clear why the above properties are correct, replace O() with a function that has that O(), e.g. replace O(N) with cN+d or 9N+100, do the math calculation and then the O() for that final expression.
   a) E.g. to show that O(N)* O(k) = O(Nk), say f(N) = cN+d=O(N) and g(k) = ak+b (note here N and k are variables, c,d,a,b are constants).
   f(N)*g(k) = (cN+d)*(ak+b) = caNk + cbN + dak + db where the variables are N and k. Note that db is a constant => Dominant term is Nk (Nk dominates N, and Nk dominates k, NK dominates db) => when we take O() we have:
   f(N)*g(k) = (cN+d)*(ak+b) = ~~ca~~Nk ~~+ cbN~~ + ~~dak~~ + ~~db~~ = O(Nk)
i) If the notation with O is confusing and , simply remove the O from the expression, simplyfy the expression if possible and then find O for the new expression.
   a) E.g. to fing O for : O(N) + O(S+N^2) + O(U) + O(N) + O(T)*O(T)

   write without O : simplify: calculate O:
   N + S + N^2 + U + N + T*T = 2N + S + N^2 + U + T^2 = O(S+N^2+U+T^2)
j)
k) Practice problems:
   a) O(1) + O(1) + O(1) + O(1) = __________
   b) O(1) + O(N) = ____________
   c) O(1) + O(N^2) = ____________
   d) O(N) + O(N^2) = ____________
   e) O(M) + O(N) = _______________
   f) M * O(N) = ____________
   g) O(M) * O(N) = ____________

TC - functions

Tuesday, January 25, 2022 8:00 AM

1. O() of function ***definition***
    1. variables used must be from the code being analyzed
        a. !!! do not just use O(N) even though you may have the correct understanding. It can lead you to make mistakes.
    2. use quantities (not arrays)
        a. E.g. if `nums1` is an array and `text` is a string, do not say "O(nums1)" or "O(text)". Use their lengths.
    3. DEFINE new variables for the size of the data that the program uses, if needed (if that variable is not in the program).
        a. E.g. function or piece of code iterates up to `strlen(text)`, chose a variable, say L, and explain that L that is the length of `text`. Note this is all a mathematical notation on paper. We are not talking about modifying the program.
2. O() of function ***call***
    1. map function arguments in function call to those in function definition and O.
    2. Pay attention to library functions. Since you do not see their definition, it may seem that they are O(1), but they may not be.
3. Be careful when analyzing string function calls in C . It may be different in other languages (e.g. if the length is stored as a member variable for an object, instead of computed as it is for strings in C). E.g.:
    1. `strlen(s)` - computes the length of string s by looking at each character in s until it finds '\0', Therefore it will have time complexity O(length(s))
    2. printf("%s", text); has to print each character of text, thus it will also have O(length(text)).
        a. `printf("%d", nums[k])` ; is constant, O(1), because array access with [] is constant (for `nums[k]`) and printing an integer is a fixed operation. It does not depend on the length of `nums`.
4. See problems from TC-Practice 1 and web practice

TC - loops

Tuesday, January 25, 2022 7:25 AM

Should I study from the slides of these notes?

The slides are more concise.
These notes try to follow all my steps and reasoning.
Run the code if needed.

Fun problem:

What is the TC of the code below? Note: k=1 and k<0. How many times does the loop repeat? O(___)

```
for(k=1; k<0; k++){
    printf("A");
}
```

Idea

1. Pay attention to each instruction in the code.
2. Solve one loop at a time.
3. When nested loops, solve them from innermost to outter most (one loop at a time). Use the time complexity (TC) of the entire inner loop when calculating the time complexity of one iteration of the outter loop.
4. If sequential loops, add their time complexities.
5. Pay attention to function calls.

Examples and motivation for the used template

Simple loop example: Example A

O(__________)

```
for(j=1; j<=N; j=j+1){
    for(k=0; k<M; k++){    //O(M)
        printf("A");
    }
    printf("\n");
}
```

How many As are printed when N is 7 and M is 10? Can you express this as a function of N and M?

A,A,….,A (10 of A on each row)
A,A,….,A
…
A,A,….,A
For 7 rows in total => 7*10 = 70 of A printed.
As a function of N and M : N*M (N rows by M columns of A)
=> TC is O(NM)
For simple loops like this one, to get the TC, you can focus on the innermost instruction

(`printf("A")`) and count or estimate how many times that executes.
But this method may fail if we have functions or more difficult loops.

To make my discussion more uniform, (so that it will be the same regardless of what code I am analyzing), I would compute O() for the for-k loop, and use that in analyzing the for-j loop.

Consider ***case 1 from*** ***Time complexity of loops*** that has a function call.

// Assume *int linear_search(int* ar, int T, int v)* has TC O(T)

```
for(j=1; j<=N; j=j+1){
   res = linear_search(nums1,M,7); // O(M)
   printf("index = %d\n", res);
}
```

Hard loop example: Example D

Note: j=j*2 and k<j

O(__________)
```
for(j=1; j<=N; j=j*2){
    for(k=0; k<j; k++){
        printf("D");
    }
    printf("\n");
}
```
This is one of the most difficult loops. Here it is non-trivial to even find out how many D are porinted on each line.
The template below will help us solve it. But before attempting this one, we will solve some easier ones first.

Template:

for-t: TC_{1iter}(___) = __________________________ dependent / independent

Change of var: ____

∑ ⬚/ repetitions ____

Closed form: ________________________________ O(_______________)

Here TC_{1iter} (____) is the time complexity of the code executed in ONE iteration of the loop. We do not worry about how many tiems the loop repeats at this point. Find O for just this part.
Why do this? So that we decompose our loop.

Note: this template is particularly useful for complex loops, but it can be applied to the trivial ones as well.

What is TC1iter for the Example A and Case 1 above? How simmilar are these two examples?

1iter = all instructions executed in one iteration of the loop
TC1iter is the time complexity fo those instructions (I do NOT worry about the repetitions)

Example A
O(__________)
```
for(j=1; j<=N; j=j+1){
     for(k=0; k<M; k++){
          printf("A");
     }
     printf("\n");
}
```

```
Case 1:
```
// Assume *int linear_search(int* ar, int T, int v)* has TC O(T)
```
for(j=1; j<=N; j=j+1){
   res = linear_search(nums1,M,7); //
   printf("index = %d\n", res);
}
```
$TC_{1iter}(j)=O(1)+O(M)+O(1)+O(1) = O(M)$
ALSO OK: $TC_{1iter}(j)= O(1)+O(M) = O(M)$
$TC_{1iter}(j)= O(M)$
TC of entire for-j loop is: N repetitions * $TC_{1iter}(j) = N*O(M) = O(NM)$

Solved example 5 from TC-practice 1:

E.g. for loop:
```
int ct = 0;
for(int k=0; k<strlen(text); k++) {
     if ( text[k] == 'A' ){
          ct++;
     }
}
```

The code executed in one iteration is:

Condition: `k<strlen(text)` ---> O(L) where L is length of text
Body of loop:
```
if ( text[k] == 'A' ){       -------> O(1)
     ct++;
}
```
Update: `k++` ---> O(1)

=> $TC_{1iter}(\mathbf{k}) = O(L)+O(1)+O(1) = O(L)$

The loop repeats L times and in each iteration it does O(L) => total time complexity for the loop: L*O(L) = $O(L^2)$
This example shows that we cannot assume the condition of the loop is O(1).
*** Note that the compiler may be optimizing this code and only calculate strlen(text) one time, save it,

and then reuse it for the following loop iterations. In that case, the actual code executed would use k<var which is O(1) and the entire loop will be O(L).

TC - conditionals (if)

Saturday, January 22, 2022 7:28 PM

1. When giving TC, we give the WORST case.
2. If possible, analyze and give best and average case as well.

Math review - log

Sunday, January 23, 2022 8:17 AM

log and exponentiation

Exponentiation

1. $a^p = a * a * a * \cdots * a \quad (where\ a\ is\ multiplied\ p\ times)$
2. Examples:
   a. $2^3 = 2 * 2 * 2 = 8$
   b. $a^5 = a * a * a * a * a$
3. Properties:
   a. $(a * b)^p = a^p * b^p$
   b. $a^{(p+r)} = a^p * a^r$
   c. $(a^p)^r = (a^r)^p = a^{pr}$

log

1. We will use lg for $\log_2$. E.g. $lgN = \log_2 N$.
2. $\log_a N = p, \quad where\ p\ is\ the\ number\ or\ times\ we\ multiply\ a\ to\ get\ N: \quad a^p = N$
3. $a^p = N \Leftrightarrow \log_a N = p$ (b.c. $\log_a N = \log_a(a^p) = p$)
4. *Examples:*
   a. $2^3 = 8\ \ and\ \ \log_2 8 = 3\ \ (b.c.:\ \log_2 8 = \log_2(2^3) = 3\ )$
   b. $3^6 = 729\ \ and\ \ \log_3 729 = 6\ \ (b.c.:\ \log_3 729 = \log_3(3^6) = 6 * \log_3 3 = 6 * 1 = 6\ )$
   c. $\log_a a = 1, \qquad \log_3 3 = 1, \quad \log_5 5 = 1$
5. Properties of log:
   a. $\log_a(XY) = \log_a X + \log_a Y$
      i. $\log_2(8 * 16) = \log_2 8 + \log_2 16 = 3 + 4 = 7$
      ii. $\log_7(25 * 61) = \log_7 25 + \log_7 61$
      iii. $\log_5(XY) = \log_5 X + \log_5 Y$
   b. $\log_a(N^p) = p * \log_a N$
   c. $\log_a N = \dfrac{\log_b N}{\log_a N}$
      i. E.g.
   d. $a^{\log_b N} = N^{\log_b a}$ (think that a and N can swap places)
      i. E.g. $5^{\log_3 N} = N^{\log_3 5}$
      ii. This is important as it shows that a function of N that may look like an exponential, $a^{\log_b N}$, is in fact a polinomial, $N^{\log_b a}$.

Math review - summations

Tuesday, January 25, 2022 7:19 AM

See first pages in slides: Time complexity of loops
and the Cheat sheet (1 page)

TC - practice 1

Saturday, January 22, 2022 7:09 PM

O for expressions with multiple variables

a) Solve:
   a) The code below has O(______)

```
for(j=0; j<N; j++){
    printf("%d, ", users[j]);
}
for(k=0; k<T; k++){
    printf("%d, ", movies[k]);
}
```

   b) N + M = O(_______)

   c) $N^2 + M + N + 35$ = O(__________)

   d) $N^2M + M^2N$ = O(__________)

   e) $N^2 + M^2 + N^2M^2$ = O(______________)

Functions and function calls

Compute O() for the function defintions below

Example 1

```
// Assumes array nums has X elements.
int middle_pos_elem(int nums[], int X){
    int idx = X/2;
    int val = nums[idx];
    return val;
}
```

Example 2A

```
// Assumes array nums has X elements.
void first_ten_1(int nums[], int X){
    for (int j=0; j<10; j++)
        printf(nums[j]);
}
```

Example 2B

```
// Assumes array nums has X elements.
void first_ten_2(int nums[], int X){
    for (int j=0; j<10; j++)
        count(nums, X, 35);  //defined below
}
```

Example 3

```
/* Sample calls:
   ...
   count1 = count(arr1, N, 35);
   ...
   countN = count(arr2, L, N);
   ...
*/
// Assumes array nums has N elements.
int count(int nums[], int N, int V){  //
    int ct = 0;
    printf("\n Counting occurrences of %d in array ...\n", V);
    for(int k=0; k<N; k++) {
        printf("%4d|", nums[k]);
        if ( nums[k] == V )
            ct++;
    }
    printf("\n");
    return ct;
}
```

Give TC (Time complexity) of:
   a) The function defintion

b) The code that includes the two function calls

Example 4

```
// Assumes array nums has N elements.
int count_pairs(int nums[], int N){
    int count = 0;
    for(int j=0; j<N; j++) {
        for(int k=0; k<N; k++) {
            //printf("(%d,%d), ",arr[j], nums[k] );
            count++;
        }
    }
    return count;
}
```

Example 5

```
// sample call: count_A(poem)  , where poem is a string
int count_A(char * text){
    int ct = 0;
    for(int k=0; k<strlen(text); k++) {
        if ( text[k] == 'A' ){
            ct++;
        }
    }
    return count;
}
```

Example 6

```
// sample call: print_counting_sheep(7,"sheep");
void print_counting_sheep(int X, char * animal){
    for (int k=0; k<X; k++){
        printf("%3d %s\n", k, animal);
    }
    printf("\n");
}
```

TC - practice 2

Tuesday, January 25, 2022 8:17 AM

Compute O() for each code piece below.
Assume all needed variables are declared and initialized. Assume the code is correct.

Example A

O(__________)
```
for(j=1; j<=N; j=j+1){
    for(k=0; k<M; k++){
        printf("A");
    }
    printf("\n");
}
```
How many As are printed when N is 7 and M is 10? Can you express this as a function of N and M?

Example B. Note: k<j

O(__________)
```
for(j=1; j<=N; j=j+1){
    for(k=0; k<j; k++){
        printf("B");
    }
    printf("\n");
}
```
How many Bs are printed when N is 7? How about when N is 8? Can you express this as a function of N?

Example C. Note: j=j*2

O(__________)
```
for(j=1; j<=N; j=j*2){
    for(k=0; k<M; k++){
        printf("C");
    }
    printf("\n");
}
```
a) What values does j take when N is 8? How about when N is 16? Can you express this as a function of N?

b) How many Cs are printed when N is 8? How about when N is 16? Can you express this as a function of N?

Example D. Note: j=j*2 and k<j

O(__________)
```
for(j=1; j<=N; j=j*2){
    for(k=0; k<j; k++){
        printf("D");
    }
    printf("\n");
}
```
a) What values does j take when N is 8? How about when N is 16? Can you express this as a function of N?

b) How many Ds are printed when N is 8? How about when N is 16? Can you express this as a function of N?

Example E

O(__________)
```
// Assume int linear_search(int* ar, int T, int v) has TC O(T
for(j=1; j<=N; j=j+1){
    res = linear_search(nums1,M,7);  // Assume you are told that this line is O(M)
    printf("index = %d\n", res);
}
```

Example F

O(__________)

// Assume *int linear_search(int* ar, int T, int v)* has TC O(T
```
for(j=1; j<=N; j=j+1){
    res = linear_search(nums1,j,7);  // note j
    printf("index = %d\n", res);
}
```

TC - practice 2-solution

Tuesday, January 25, 2022 8:17 AM

Compute O() for each code piece below.
Assume all needed variables are declared and initialized. Assume the code is correct.

Example A

O(__________)

```
for(j=1; j<=N; j=j+1){
    for(k=0; k<M; k++){
        printf("A");
    }
    printf("\n");
}
```

How many As are printed when N is 7 and M is 10? Can you express this as a function of N and M?

```
Example A: N=7, M=10
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAA
```

for-k: TC_{1iter}(____) = ______________________ dependent / independent

Change of var: ______________________

Σ / repetitions ______________________

Closed form: ______________________ O(______________)

for-j: TC_{1iter}(____) = ______________________ dependent / independent

Change of var: ______________________

Σ / repetitions ______________________

Closed form: ______________________ O(______________)

Example B. Note: k<j

O(__________)

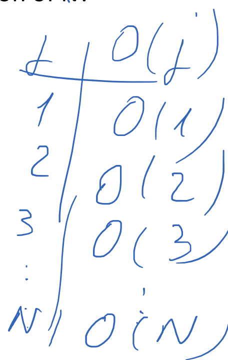
```
for(j=1; j<=N; j=j+1){
    for(k=0; k<j; k++){
        printf("B");
    }
    printf("\n");
}
```

How many Bs are printed when N is 7? How about when N is 8? Can you express this as a function of N?

```
    printf("\n");
}
```

How many Bs are printed when N is 7? How about when N is 8? Can you express this as a function of N?

```
Example B: N=7
B
BB
BBB
BBBB
BBBBB
BBBBBB
BBBBBBB
```



for-k: TC_{1iter}(___) = __________________________ dependent / independent

Change of var: ____

$\sum$⬚/ repetitions ____

Closed form: _______________________________ O(______________)

for-j: TC_{1iter}(___) = __________________________ dependent / independent

Change of var: ____

$\sum$⬚/ repetitions ____

Closed form: _______________________________ O(______________)

Example C. Note: j=j*2

O(___________)

```
for(j=1; j<=N; j=j*2){
    for(k=0; k<M; k++){
        printf("C");
    }
    printf("\n");
}
```

a) What values does j take when N is 8? How about when N is 16? Can you express this as a function of N?

b) How many Cs are printed when N is 8? How about when N is 16? Can you express this as a function of N?

Example C: N=8, M=10	Example C: N=16, M=10	Example C: N=32, M=10
j= 1, CCCCCCCCCC	j= 1, CCCCCCCCCC	j= 1, CCCCCCCCCC
j= 2, CCCCCCCCCC	j= 2, CCCCCCCCCC	j= 2, CCCCCCCCCC
j= 4, CCCCCCCCCC	j= 4, CCCCCCCCCC	j= 4, CCCCCCCCCC
j= 8, CCCCCCCCCC	j= 8, CCCCCCCCCC	j= 8, CCCCCCCCCC
	j= 16, CCCCCCCCCC	j= 16, CCCCCCCCCC
		j= 32, CCCCCCCCCC

for-k: TC_{1iter}(___) = __________________________ dependent / independent

Change of var: ____

$\sum$⬚/ repetitions ____

Σ / repetitions M reps * O(1)

Closed form: ______ X ______ O(M)

for-j: TC_{1iter}(j) = O(M)   ~~dependent~~ / independent

Change of var: Yes: j: 1, 2, 4, 8, ... 2^x, ... 2^p = j_{last} = N => $p = \log_2 N$ repetition

~~Σ~~ / repetitions p * O(M) = ($\log_2 N$) * O(M)

Closed form: ______ X ______ O($M \cdot \log_2 N$)

NO p in final answer

Example D. Note: j=j*2 and k<j

O(__________)

```
for(j=1; j<=N; j=j*2){
     for(k=0; k<j; k++){
          printf("D");
     }
     printf("\n");
}
```

a) What values does j take when N is 8? How about when N is 16? Can you express this as a function of N?
b) How many Ds are printed when N is 8? How about when N is 16? Can you express this as a function of N?

Example D: N=8	Example D: N=10	Example D: N=16
j= 1, D	j= 1, D	j= 1, D
j= 2, DD	j= 2, DD	j= 2, DD
j= 4, DDDD	j= 4, DDDD	j= 4, DDDD
j= 8, DDDDDDDD	j= 8, DDDDDDDD	j= 8, DDDDDDDD
		j= 16, DDDDDDDDDDDDDDDD

for-k: TC_{1iter}(k) = O(1)   ~~dependent~~ / independent

Change of var: NO

~~Σ~~ / repetitions j reps * O(1) = j·O(1) = O(j)

Closed form: ______ X ______ O(j)

for-j: TC_{1iter}(j) = O(j)   dependent / ~~independent~~

Change of var: Yes: j = 1, 2, 4, 8, ..., 2^x, ... 2^p = j_{last} = N => $p = \log_2 N$ $2^p = N$

Σ / ~~repetitions~~ $\sum_j O(j) = (\sum_{x=1}^{p} 2^x) = \frac{2^{p+1} - 1}{2 - 1} = (2 \cdot 2^p - 1) = 2 \cdot N - 1 = O(N)$

Closed form: ______ O(N)

$\sum_{j=1}^{N} O(j)$ wrong b.c. implies j takes consec values

Example F

O(__________)

// Assume *int linear_search(int* ar, int T, int v)* has TC *O(T)*

```
for(j=1; j<=N; j=j+1){
     res = linear_search(nums1,j,7);  // note j
     printf("index = %d\n", res);
}
```

$TC_{linear_search(nums1,j,7)}$=O(j)

for-j: TC_{1iter}(___) = __________________________ dependent / independent

Change of var: ____

∑ ⬚/ repetitions ____

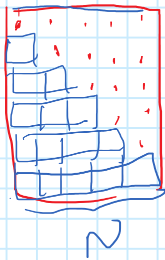
Closed form: ________________________________ O(_______________)

Intuition on sum of k and sum of 2^k

Wednesday, January 26, 2022 12:17 AM

1) 2^k: $1 + 2 + 4 + 8 + 16 + \dots + N = 2N - 1 = O(N)$, where $N = 2^p$

2) k: $1 + 2 + 3 + 4 + \dots + (N-1) + N = \frac{N(N+1)}{2} = O(N^2)$



$(N+1) \Rightarrow \frac{N \cdot (N+1)}{2}$

N

$N = 32$: $32 + 16 + 8 + 4 + 2 + 1 = 2N - 1$ (with $16 + 8 + 4 + 2 + 1 = N - 1$)