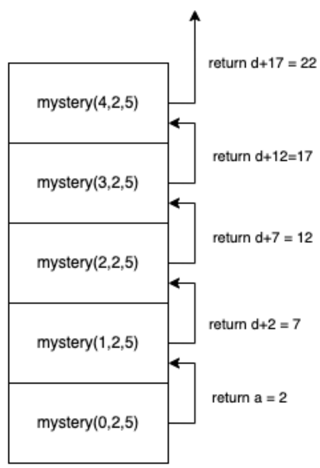
Name & UNI: Xintong Qi xq2224 Date: 23 September 2022

Point values are assigned for each question. Points earned: ____ / 100, = ____ %

1. Consider the following method:

```
public static int mystery(int n, int a, int d) {
    if (n == 0) {
        return a;
    }
    return d + mystery(n - 1, a, d);
}
```

Draw the call stack showing the stack frame for each method call with its corresponding return value when calling `mystery(4, 2, 5)`. (6 points)



What value is returned by `mystery(4, 2, 5)`? (4 points)
22

2. Find an upper bound for $f(n) = n^4 + 10n^2 + 5$. Write your answer here: $O(n^4)$ (4 points)

Prove your answer by giving values for the constants c and n_0. Choose the smallest integer value possible for c. (6 points)

Let $c = 2$
Suppose we have $g(n) = 2n^4$. We need to find a n_0 such that for every $n \geq n_0$, $g(n) > f(n)$

$$2n^4 \ > n^4 + 10n^2 + 5$$
$$n^4 - 10n^2 - 5 > 0$$

Substitute $n = 4$, and the inequality holds.
Since $F(n) = n^4 - 10n^2 - 5$ increases monotonously for $x > 0$, we know for any $n > 4$, the inequality also holds. Therefore $g(n) > f(n)$ is valid for any $n \geq n_0$ where $n_0 = 4$
And we prove that for $c = 2$ and $n_0 = 4$, there is an upper bound for $f(n) = n^4 + 10n^2 + 5$

3. Find a lower bound for $3n - 4$. Write your answer here: $\Omega(n)$ (4 points)

   Prove your answer by giving values for the constants c and n_0. Choose the largest integer value possible for c. (6 points)

   Let $c = 2$
   Suppose we have $g(n) = 2n$. We need to find a n_0 such that for every $n \geq n_0$, $g(n) < f(n)$

$$2n < 3n - 4$$
$$n > 4$$

   This means the inequality holds for all $n > 4$ starting from $n_0 = 5$
   Therefore $g(n) < f(n)$ is valid for any $n \geq n_0$ where $n_0 = 5$
   And we prove that for constants $c = 2$ and $n_0 = 5$, there is a lower bound for $f(n) = 3n - 4$

4. Find an asymptotically tight bound for $f(n) = 3n^3 - 2n$. Write your answer here: $\Theta(n^3)$ (4 points)

   Prove your answer by giving values for the constants c_1, c_2, and n_0. Choose the tightest integer values possible for c_1 and c_2. (6 points)

   Let $c_1 = 2, c_2 = 3$
   Lower bound:
   Suppose we have $g(n) = 2n^3$. We need to find a n_0 such that for every $n \geq n_0$, $g_1(n) < f(n)$

$$2n^3 < 3n^3 - 2n$$
$$2n < n^3$$
$$2 < n^2$$

   Substitute $n = 2$ and the inequality holds.
   Since n^2 increases monotonously for $n > 0$, the inequality holds for all $n \geq 2$. Therefore $g_1(n) < f(n)$ holds for all $n \geq 2$

Upper bound:
Suppose we have $g(n) = 3n^3$. We need to find a n_0 such that for every $n \geq n_0$, $g_2(n) > f(n)$

$$3n^3 > 3n^3 - 2n$$

This is true for every $n > 0$. Therefore $g_2(n) > f(n)$ holds for all $n \geq 1$

Combine the two conditions together, we know there is an asymptotically tight bound for $f(n) = 3n^3 - 2n$ that is valid for all $n \geq 2$, and the tight bound is of order n^3

5. Write the following asymptotic efficiency classes in **increasing** order of magnitude.
   $O(n^2)$, $O(2^n)$, $O(1)$, $O(n \lg n)$, $O(n)$, $O(n!)$, $O(n^3)$, $O(\lg n)$, $O(n^n)$, $O(n^2 \lg n)$ (1 point each)

   $$O(1) < O(lgn) < O(n) < O(nlgn) < O(n^2) < O(n^2lgn) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

6. Determine the largest size *n* of a problem that can be solved in time *t*, assuming that the algorithm takes *f(n)* milliseconds. Write your answer for *n* as an integer [no scientific notation]. (2 points each)

   a. *f(n)* = n, *t* = 1 second   1000__________
      $n \leq 1 * 1000$
   b. *f(n)* = $n \lg n$, *t* = 1 hour   204094__________
      $n \lg n \leq 60 * 60 * 1000$
      Code for binary search in python

```
import math

def binary_search(low, high):

  if high >= low:
    mid = (high + low) // 2
    if mid*math.log(mid,2)<3600000 and (mid+1)*math.log(mid+1,2)>3600000:
      return mid
    elif mid*math.log(mid,2)>3600000:
      return binary_search(low, mid - 1)
    else:
      return binary_search(mid + 1, high)
  else:
    return -1

low=1
high=3600000

binary_search(low,high)
```

      the result gives 204094
   c. *f(n)* = n^2, *t* = 1 hour   1897__________

$n^2 \leq 1 * 60 * 60 * 1000$

d. *f(n)* = n^3, *t* = 1 day 442__________

$n^3 \leq 24 * 60 * 60 * 1000$

e. *f(n)* = $n!$, *t* = 1 minute 8__________

$n! \leq 60 * 1000$

Substitute $n = 0,1,2, \ldots, 8$ and $n! \leq 60 * 1000$. When $n = 9$, the inequality no longer holds

7. Suppose we are comparing two sorting algorithms and that for all inputs of size n the first algorithm runs in $4n^3$ seconds, while the second algorithm runs in $64n\ lg\ n$ seconds. For which integral values of n does the first algorithm beat the second algorithm? $\underline{2 \leq n \leq 6}$ ________________ (4 points)

   Explain how you got your answer or paste code you wrote that solves the problem (6 points):

   The first algorithm beats the second if the first one needs less time, so we have the following inequality.

   $$4n^3 < 64n \lg n$$

   Simplifies it and we get

   $$n^2 < 16 \lg n$$

   To find the value of n that makes the inequality hold, we need to first find the solution to the equation $n^2 - 16 \lg n = 0$.

   Taking the derivative of $f(x) = n^2 - 16 \lg n = n^2 - \frac{16}{\ln(2)} lnn$, we get $f'(x) = 2n - \frac{16}{\ln(2)} * \frac{1}{n}$.

   Setting the derivative to 0, we get the value of $n = 3.4$, which means that function $f(x)$ increases monotonously for $n > 3.4$. Since n can only be integers, that means the minimum value for $f(x)$ is taken at either $n = 3$ or $n = 4$.

   For $n \leq 3$, $f(x)$ decreases; for $n \geq 4$, $f(x)$ increases.

   Notice when $n = 1, n^2 - 16 \lg n = 1 > 0$, and when $n = 7, n^2 - 16 \lg n = 4.08 > 0$

   Therefore, $4n^3 < 64n \lg n$ holds for $n = 2,3,4,5,6$

   There are lots of other ways to find the values of n, such as implementing a binary search, which is what is taught in class. I ran it with the binary search program in question 6 to validate the result, and it gave me the output 6, which proves that my calculus method is right.

   But I think the calculus method is the most trivial and rigorous, and the idea of using a binary search is essentially backed by the calculus idea☺

8. Write pseudocode for an efficient algorithm for computing the LCM of an array A[1..n] of integers. You may assume there is a working `gcd(m, n)` function that returns the gcd of exactly two integers. (10 points)

```
ALGORITHM LCM(A[1..n]):
    // Computes the least common multiple of all the integer in array A
    int lcm=A[1];
    if(A.length==1){
          return lcm; // the element itself is its LCM
```

```
}

for (int i=2; i<=A.length; i++){
      lcm = (lcm*A[i])/gcd(lcm,A[i]); // lcm(a,b) = a*b/gcd(a,b)
}
return lcm;
```

9. Give the complexity of the following methods. Choose the most appropriate notation from among O, Θ, and Ω. (4 points each – 2 for symbol, 2 for function)

```
int function1(int n) {
    int count = 0;
    for (int i = n / 2; i <= n; i++) {
        for (int j = 1; j <= n; j *= 2) {
            count++;
        }
    }
    return count;
}
```

Answer: $\Theta(n\log_2 n)$___________

Outer loop runs for n-n/2+1 times, inner loop runs for $\log_2 n$ times

No early loop breaking, use theta.

```
int function2(int n) {
    int count = 0;
    for (int i = 1; i * i * i <= n; i++) {
        count++;
    }
    return count;
}
```

Answer: $\Theta(n^{\frac{1}{3}})$__________

The loop starts running until $i > n^{\frac{1}{3}}$, and i increment by 1 each time, so the loop runs approximately $n^{\frac{1}{3}}$ times (but may be a little bit more or less than this depending on if $n^{\frac{1}{3}}$ is an integer)
No early loop breaking, use theta.

```
void function3(int n) {
    if (n % 2 == 0) {
        return;
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            System.out.print("*");
            break;
        }
    }
}
```

Answer: $O(n)$__________
The loop may early exit if n is even, so use big-O.
Consider the two loops, the outer loop runs for n times and the inner loop runs for only once for each i.

```
void function4(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j += i) {
            System.out.print("*");
        }
    }
}
```

Answer: $\Theta(n^2)$__________
The outer loop runs for n times.
Consider the pattern of j in the inner loop, 1, 1+i, 1+2i, 1+3i…until 1+xi=n and x+1 is the number of times the inner loop runs for each i
So for each i the inner loop runs for x+1=(n-1)/i+1 times
And the total number of time taken is
1* (n-1)/1+1 + 2* (n-1)/2+1 + … + n* (n-1)/n+1=n^2
Again, there is no early breaking, so use theta

```
int function5(int n) {
    int count = 0;
    for (int i = 1; i <= n; i++) {
```

```
        count++;
    }
    for (int j = 1; j <= n; j++) {
        count++;
    }
    return count;
}
```

Answer: $\Theta(n)$___________

The first loop runs for n times. The second loop runs for n times. Since the second loop runs after the first loop, the total time is n + n = 2n

No early loop breaking, use theta