University of Nevada, Las Vegas Computer Science 477/677 Spring 2025

Answers to Assignment 2: Due Saturday February 1, 2025

1. Write either O, Ω or Θ in each blank. Write Θ if that is correct, otherwise write O or Ω.

   (a) $n - 100 = \Theta(n - 200)$

   (b) $n^{1/2} = O(n^{2/3})$

   (c) $100n + \log n = \Theta(n + \log^2 n)$

   (d) $n \log n = \Omega(10n + \log(10n))$

   (e) $\log(2n) = \Theta(\log(3n))$

   (f) $10 \log n = \Theta(\log(n^2))$

   (g) $n^{1.01} = \Omega(n \log^2 n)$

   (h) $n^2 / \log n = \Omega(n \log^2 n)$

   (i) $n^{0.1} = \Omega(\log^2 n)$

   (j) $(\log n)^{\log n} = O(n / \log n)$

   (k) $\sqrt{n} = \Omega(\log^3 n)$

   (l) $n^{1/2} = O_{--}(5^{\log_2 n})$

   (m) $n2^n = O(3^n)$

   (n) $2^n = O(2^{n+1})$

   (o) $n! = \Omega(2^n)$

   (p) $\log_2 n^{\log_2 n} = O(2^{(\log_2 n)^2})$

   (q) $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$

2. Look up Fibonacci numbers $F_1, F_2, F_3 \ldots$ if you are not familiar with them. Recall that $F_i + F_{i+1} = F_{i+2}$. The first few Fibonacci numbers are 1, 1, 2, 3, 5, 8, ... Find the smallest constant C such that $F_n = O(C^n)$.

   Assume that $F_n = C^n$. This is false, but it's good enough for asymptotic computation. Then $C^n +$

$C^{n+1} = C^{n+2}$. Divide both sides by C^n, and we have the quadratic equation $1 + C = C^2$. Then $C = \dfrac{1+\sqrt{5}}{2}$.

3. Consider the following C++ program.

```
void process(int n)
 {
  if(n > 1) process(n/2);
  cout << n%2;
 }

int main()
 {
  int n;
  cout << "Enter a positive integer: ";
  cin >> n;
  assert(n > 0);
  process(n);
  cout << endl;
  return 1;
 }
```

The output of `process(n)` is a string of bits. What does this bitstring represent?

The output is the binary numeral for n.

4. The C++ code below implements a function, "mystery." What does it compute?

```
float mystery(float x, int k)
 {
  assert(k > 0 or x != 0);
  if (k == 0) return 1.0;
  else if(x == 0.0) return 0.0;
  else if (k < 0) return 1/mystery(x,-k);
  else if (k%2) return x*mystery(x,k-1); // k is positive odd
  else return mystery(x*x,k/2); // k is positive even
 }
```

The function returns x^k, but requires the input condition that if $x = 0$, k must be positive, and that if $k = 0$, x must be non-zero. The reason is that, in those cases, x^k is mathematically undefined, I added an assert command, which will cause the program to halt in those cases.

Here is how the code computes mystery$(2, -13)$.

Recursive functions use the runtime stack. When I searched for that term, I got the following on a Google page:

“ A runtime stack is a data structure that stores information about functions and variables while a program is running. It's used to pass parameters to functions, return values from functions, and store local variables.

“

How it works.

“ When a function is called, parameters are pushed onto the stack. When a function returns, the parameters are popped off the stack. The stack record for a function is created and populated when the function is called. The stack record for a function is removed when the function completes. What it stores Local variables, Parameter variables, Temporary variables created by the compiler, and The return address of the code that follows the function call.”

Let us examine, in detail, the computation of mystery(2.0,-7). Since $-7 < 0$, the program will return 1/mystery(2.0,7).

At recursion depth 1, the program computes mystery(2.0,7). Since 7 is odd, the program will return 2.0∗mystery(2.0,6).

At recursive depth 2, the program computes mystery(2.0,6), Since 6 is even, the program will return mystery(4.0,3).

At recursion depth 3, the program computes mystery(4.0,3). Since 3 is odd, the program will compute 4.0∗mystery(4.0,2) = 64.0.

At recursion depth 4, the program computes mystery(4.0,2). Since 2 is even, the program will return mystery(16.0,1).

At recursion depth 5, the program computes mystery(16.0,1). Since 1 is odd, the program will compute 16.0∗mystery(16.0,0)

At recursion depth 6, the program computes mystery(16.0,0). Since $k = 0$, the program returns 1.0.

At recursion depth 5, the program returns 16.0∗mystery(16,1) = $16.0 * 1.0 = 16.0$

At recursion depth 4, the program returns mystery(4,0,2) =mystery(16,1)= 16.0.

At recursion depth 3, the program returns mystery(4.0,3) = 4.0∗ mystery(4.0,2) = $4.0 * 16.0 = 64.0$).

At recursion depth 2, the program returns mystery(2.0,6) = mystery(4.0,3) = 64.0.

At recursion depth 1, the program returns mystery(2.0,7) = 2.0∗mystery(2.0,6) = $2.0 * 64.0 = 128.0$

The main program returns 1/mystery(2.0,7) = 1/128.0 = .00781255

5. The C++ code below implements a function. What does that function compute?

```
int gcd(int n, int m)
 {
  assert(n != 0 or m != 0);
  if(n < 0) return gcd(-n,m);
  else if(m < 0) return gcd(n,-m);
  else if(n < m) return gcd(m,n);
  else if(m > 0) return gcd(m,n%m);
```

```
  else return n;
}
```

The greatest common divisor of n and m. gcd(0,0) is not mathematically defined.