1.3 a. $f(n) = \log(n^{kn})$ $g(n) = \log((kn)^n)$ $(n^n)^k$ $k^n \cdot n^n$ Greater

$f(n) = \Omega(g(n))$

(b) $\theta \to 0$ $\sin\theta = \theta$. $f(n) = \Omega(g(n))$

(d) $\boxed{\log_a^n = x \quad a^x = n}$

$n^{\log \log_a^n} = (a^x)^{\log_a^x} = (a^{\log_a^x})^x = x^x$

$(\log_a n)^{\log_a^n} = x^x$ $f(n) = \Theta(g(n))$

Written Assignment 1: Algorithm Analysis

1.1 $f(n) = \Theta(g(n))$

$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$g(n) = \Theta(h(n))$

$g(n) = O(h(n))$ and $g(n) = \Omega(h(n))$

$\Rightarrow f(n) = O(O(h(n)))$ and $f(n) = \Omega(\Omega(h(n)))$

$\Rightarrow f(n) = \Theta(h(n))$

1.2 $h(n) = f(n) = 2^n$

$a \cdot 2^{\frac{n}{2}} \le 2^n \le b \cdot 2^{\frac{n}{2}}$

$\Rightarrow$ prove $\forall n > k$. $2^{\frac{n}{2}} \le b$.

$\Rightarrow \lim_{n\to\infty} 2^{\frac{n}{2}} = +\infty$.

$\Rightarrow n > k$ (when) $\Rightarrow 2^{\frac{n}{2}} > b$

$\Rightarrow f(n) \ne \Theta(h(n))$

1.3 (e) $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \dots + n^2$

$n \sum_{i=1}^{n} (n-i) = n[\underbrace{(n-1)+(n-2)+\dots+1}_{(n-1)\text{ times}}] = n \times \frac{n-1}{2} \times n$

$f(n) = \Omega(g(n))$

1 O, Ω, Θ

1.1 Problem 1 (10 points)

Given three positive functions f, g, and h, use the definition of Θ to prove the following: if $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n) = \Theta(h(n))$.

1.2 Problem 2 (10 points)

Given a positive function f, prove or disprove the following: $f(n) = \Theta(f(n/2))$.

1.3 Problem 3 (25 points)

For each pair of functions f and g below, decide whether $f(n) = O(g(n))$, $\boxed{f(n) = \Omega(g(n))}$, or $\boxed{f(n) = \Theta(g(n))}$. Justify your answers step by step using the definitions of these asymptotic notations. Note that more than one of these relations may hold for a given pair; list all correct ones.

(a) $f(n) = kn\log(n)$ and $g(n) = n\log(kn)$ for any positive constant k.

(b) $f(n) = n(\sin(n))^2$ and $g(n) = \sqrt{n}$

(c) $f(n) = \sqrt{n}^{\sqrt{n}}$ and $g(n) = \sqrt{n^n}$

(d) $f(n) = n^{\log(\log(n))}$ and $g(n) = \log(n)^{\log(n)}$

(e) $f(n) = \Sigma_{i=1}^{n} i^2$ and $g(n) = n\Sigma_{i=1}^{n}(n-i)$

2 Recurrence Relations

2.1 Problem 4 (30 points)

Given a positive number k, solve the following recurrence relation to find an exact formula for $T(n)$ (to simplify, you can assume that n is a power of 2):

$$T(n) = \begin{cases} kT(n/2) + \log_2(n/2) & n > 1 \\ 1 & n = 1 \end{cases}$$

Note: $S(x, n) = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1}-1}{x-1}$ when $x \neq 1$. And therefore, by derivation with regard to x: $S'(x, n) = 1 + 2x + 3x^2 + \ldots + nx^{n-1} = \frac{nx^{n+1}-(n+1)x^n+1}{(x-1)^2}$ when $x \neq 1$.

Note: the formula for $T(n)$ might be different when $k = 1$.

After you have computed the exact formula, give a tight asymptotic bound for $T(n)$ (again, the result might be different for different values of k). There is no need to explain, just give the final result.

2.2 Problem 5 (15 points)

Consider the following C code to compute 2^n recursively:

```
int pow(int n) {
  if(n == 0)            → 1.
    return 1;           → 1.
  int pmid = pow(n / 2);   → 1+(n/2)^n
  if(n % 2 == 0) {      → 2.
    return pmid * pmid;   → 2.
  } else {
    return 2 * pmid * pmid;  → 3.
  }
}
```

$0 \le n < 2 = 1$

$2 \le n < 4 : 2$

$p(n) = 1 + 1 + 1 + (\frac{n}{2})^n + 2 + 2 + 3$

$p(n) = T(n) = 10 + (\frac{n}{2})^n \longrightarrow O(n)$

$2^n = (2^{\frac{n}{2}})^2$; when n is odd: $2^n = 2 * 2 (2^{(n-1)/2})^2$.

$2 \le T(n) \le$ $\log_2 n + 10.$ 不确定（不会）

If n is even then $2^n = 2^{n/2} * 2^{n/2} = (2^{n/2})^2$. If n is odd then $2^n = 2 * 2^{(n-1)/2} * 2^{(n-1)/2} = 2 * (2^{(n-1)/2})^2$ (and $(n-1)/2$ is the same as $n/2$ when n is odd and you are using the C language's integer division).

Call $T(n)$ the amount of time required to execute the function `pow(n)`. For each line of the C code above, give the exact number of operations required (see slide 19 of the lecture notes for week 5). Each C operator counts as one operation, and `return` counts as one operation too.

Then write a recursive inequation of the form "$\ldots \le T(n) \le \ldots$" Explain in English how you get the inequation. (There is no need to solve the equation.)

2.3 Problem 6 (10 points)

Consider the following C code to compute the integer logarithm $\log_2(n)$ (the biggest natural number k such that $2^k \le n$) using binary search:

```
int log(int n) {
  int start = 0, end = n;     → 2
  while(start < end - 1) {    → 2
    int mid = (start + end) / 2;   → 3
    if(pow(mid) <= n) {   → mid+1
      start = mid;      → 1
    } else {
      end = mid;      → 1
    }
  }
  return start;     → 1
}
```

$T(n) = 7 + mid + 4 = 11 + mid.$

$U(n) = \log_2 n (\log_2 n + 7) + 3$

$\Rightarrow \Theta(\log_2 n)$

The algorithm starts with the interval $[0, n)$, and checks whether the power of 2 of the midpoint is less than or equal to n (in which case the right half of the interval becomes the new search interval) or bigger than n (in which case the left half of the interval becomes the new search interval). The algorithm then repeats the whole process until the interval has a length of 1, at which point `start` is the answer.

Call $U(n)$ the amount of time required to execute the function `log(n)`. Write a recursive equation of the form "$U(n) = \ldots$" Explain in English how you get the equation. You can assume that $T(n) = \Theta(\log(n))$. You do not have to count the exact number of operations: write a recurrence relation for $U(n)$ using Θ and simplify as much as possible. (There is no need to solve the equation.)