**Problem Set 5, Part I**

**Problem 1: Expressing Big-O**

1. **a(n) = O(n)**
2. b(n) = O(n^2)
3. c(n) = O(n^3)
4. d(n) = O(log n)
5. e(n) = O(n log n)
6. f(n) = O(n^2)
7. g(n) = O(log n)
8. n(n) = O(n^2)

**Problem 2: Computing Big-O**

2-1)

outer loop: n times

middle loop: n times

inner loop: (n - 1)/2 times

f(n) = n \* n \* (n - 1)/2 = O(n^3)

2-2)

outer loop: log n times

middle loop: n times

inner loop: 1000 times

f(n) = log n \* n \* 1000 = O(n log n)

2-3)

outer loop: n times

middle loop: 2n times

inner loop: log n times

f(n) = n \* 2n \* log n = O(n^2 log n)

**Problem 3: Sum generator**

3-1)

outer loop: n times

inner loop: (n - 1)/2 times

f(n) = n \* (n - 1) / 2 = (n^2 - n) / 2

3-2)

f(n) = n \* (n - 1) / 2 = O(n^2)

Because n^2 / 2 - n / 2 <= n^2 for all n >= 0

3-3)

文本

描述已自动生成

3-4)

loop: n times

f(n) = n = O(n)

Because we have only one loop in the function and n<= n for all n >= 0

**Problem 4: Comparing two algorithms**

4-1)

**best case:** A(n) = O(1)

the largest element is the first element

**average case:** A(n) = O(n/2)

the largest element is the middle element

**worst case:** A(n) = O(n)

the largest element is the last element or all the elements are equal

4-2)

**best case:** B(n) = O(n)

the array is already sorted, and no moves are needed

**average case:** B(n) = O(n^2)

B(n) = n \* (n - 1) / 4 = O(n^2)

**worst case:** B(n) = O(n^2)

B(n) = n \* (n - 1) / 2 = O(n^2)

the array is in reverse order, and every comparison leads to a swap

4-3)

the Algorithm A has a more efficient run-time complexity.

Comparing the best case of two algorithm, O(1) is faster than O(n);

Comparing the average case, O(n/2) is faster than O(n^2);

Comparing the worst case, O(n) is faster than O(n^2).

Since for all three of best, average, and worst cases Algorithm A is faster than Algorithm B, Algorithm A has a more efficient run-time complexity.