Exam Questions

Section 1: Complexity Analysis

Questions: Worst Case Running Times:

A.

```
void foo(int n){
  int count = 0;
  for(int i =0;i<n; i++){
    for(int j = i^2; j>0; j--){
      count = count + 1;
    }
  }
  return count;
}
```

Number of primitive operations:

2

2 + (n+1) + n

n(3 + (i^2+1) + i^2)

n * i^2 * 2

1

Running Time, T(n)

2 + 2 + n + 1 + n + 3n + ni^2 + n + ni^2 + $2n^2$ + 1

T(n) = 6 + 6n + $4ni^2$

For worst case: i = n

T(n) = 6 + 6n + 4n(n^2) = $4n^3$ + 8n + 6 = O(n^3)

B.

```
void foo(int n){
  if(n<=0) return;
  System.out.println(n);
  foo(n/4);
}
```

Number of Primitive Operations:

2

1

$T(n/4)$

The time complexity will be:

T(n) = 3 + $T(n/4)$, n > 0 &

T(n) = 2, n≤0

Let's do integer division each step:

Step 1: T(n) = 3 + T([n/4])

Step 2: T(n) = $2 * 3$ + T($[n/4^2]$)

Step 3: T(n) = 3*3 + T($[n/4^3]$)

At kth step (when it reaches the best case):

T(n) = k * 3 + T($[n/4^k]$) which is

= k * 3 + T(0) → bestcase

$[n/4^k]$ = 0

⇒ $4^k > n$

⇒ log 4^k > log n

⇒ k > logn/2

take k = logn ⇒ logn > 1/2 logn = True, n> 0

T(n) = 3 * logn + T(0)

⇒ 3 log n + 2

⇒ O(log n)

C.

```
void foo(int n){
  for (int i = 0; i<n; ++1){
    for(int j =0, j<i; ++j){
      for (k = 0; k<10; ++k){
        System.out.println(j);
      }
    }
    for(k - 0; k<i; ++k){
      System.out.println(k);
    }
  }
}
```

Solution:

Number of primitive operations for each lines:

2 + (n+1) + n

n(2 + (n+1) + n)

n * n(2 + (10 + 1) + 10)

n(2 + (n + 1) + n)

Running time T(n):

2 + n + 1 + n + 2n + n^2 + n + n^2 + 2n^2 + 10n^2 + n^2 + 10+ 2n + n ^2 + n + n

16n^2 + 9n + 13

Worst Case Running Time is **O(n^2)**

D.

```
void foo(int n){
  if(n<=0) return;
  System.out.print(n);
  foo(n/3) * foo(n/2)
}
```

Numer of primitive operations for each lines:

2

1

T(n/3) * T(n/2)

Worst Case Running Time is **O(2^n)**

E.

```
int foo(int n){
  m = 0;
  while(n>=2){
    n--;
    m = m+1;
  }
  return m;
}
```

Number of primitive operations;

1

n-2

(n-2)+1

(n-1)+2

1

Running Time T(n) = 1 + n - 2 + n -2 + 1 + n-1 + 2 + 1 = 3n

Worst Case Running Time is **O(n)**

Questions: Asymptotic Analysis

1. Discuss whether the next statements are true for the given function f(n) = 2n + n log n

a) f(n) = O(n^2)

for f(n) = O(n^2) to be true, lim n → infinity f(n) / n^2 must result in finite number

lim n → infinity (2n + n log n)/n^2

lim n → inifinit 2/n + lim n → inifinity log n   = 0 + 0 = 0

f(n) = O(n^2) **True**

In Other way:

For f(n) = O(n^2) to be true, there must be c & n > n0, such that

f(n) ≤ cg(n)

2n + n log n ≤ cn^2

2/n + n logn ≤ c , take n0 = 2

1 + 1/2 ≤ c

c ≥ 3

Let's take c = 2

2n + n log n ≤ 2 n^2

2/n + log n / n ≤ 2

2/n ≤ 1 for n0 ≥0

log n / n ≤ 2 for n0 ≥ 2

Therefore, 2/n + logn/2 ≤ 2 is True

Hence, f(n) = O(n^2) is True

b) f(n) = Theta (n log n)

Let's check Big O first:

For f(n) = Theta(n log n) to be true, lim n→ infinity (2n + n log n)/n log n must result in a finite number:

lim n → infinity 2n/nlogn + lim n→ infinity n log n/ n log n

= 0 + 1 = 1 is a finite number.

Hence f(n) is O(n log n)

Let's check Big Omega:

For f(n) to be Omega (n log n), g(n) must be less than or equal to c f(n)

n log n ≤ c(2n + n log n)

(n log n) / (2n + n log n) ≤ c

1/ ((2/nlogn) + 1) ≤ c, let's take n0 = 2

1/ (2/2+1) + 1) ≤ c

1/2 ≤ c

Let's take c = 1

nlogn ≤ 2n + n log n

0 ≤ 2n, n > 2 True

Hence: f(n) = Theta (n log n) is true, as it satisfies both Big O and Big Omega conditions.

2. For each of the following below, give the tightest asymptotic bound possible.

a. f(n) = 10 n^5 log (n^10) + n log (n^20) + 5n

Ans: O(n^5)

b. f(n) = 2 ^ 100 + n^4 + n!

Ans: O(n^4)

3. Multiple Choice Questions

1. **What is the tightest asymptotic bound for the following f(n): f(n) = $2 * n^n +$ 2 ^ 100**

Ans: O(n^n)

2. The tightest asymptotic bound for the following f(n): f(n) = $n^2 log(n^5) + 5nlog(n^5) + n$

Ans: $O(n^2 log(n^5))$

3. Which of the following is not O(n^3):

a. N ^ 2.98

b. 15^10 n + 1000

c. n^4/n^1/2

d. 2^20 n ^ 2

Ans: c

4. Which of the given options provide the increasing order of asymptotic complexity of functions f1, f2, f3, and f4?

f1(n) = 2n

f2(n) = n!

f3(n) = n log n

f4(n) = 7 ^ n

Ans: f1 < f3 < f4 < f2

O(2n) < O(n log n) < O(7 ^ n) < O(n!)

4. If you have three functions f1, f2, and f3, where f1 is in Theta(n), f2 is in O(n), and f3 is in big-Omega(n).Explain the time complexity for each function using the definition of each asymptotic relationship.

Solution:

f1 = Theta(n)

f2 = O(n)

f3 = Omega(n)

Definition 1: f(n) is Theta(g(n)) iff f(n) is both O(g(n)) and Omega(g(n)).

f1 = Theta(n) means f1 is both O(n) and Omega(n)

Definition 2: f(n) is O(g(n)) iff there are positive constants c and n0 such that f(n) <= cg(n) for all n >= n0.

f2 is O(n) means that there are positve const c and n0 such that f2 ≤ cg(n) for all n ≥ n0.

Definition 3: f(n) is Omega(g(n)) iff g(n) is O(f(n)). This means that n is O(f3).

5. Prove that the order of f(n) = 2n^3 + 5n is O(n^3) using the definition of big-O(n)?

Solution:

Definition: f(n) is O(g(n)) iff there are positve constants c and n0 such that f(n) ≤ cg(n) for all n ≥ n0.

2n^3 + 5n ≤ c * n^3

2 + 5/n^2 ≤ c

Let's take n0 = 2

2+ 5/4 ≤ c

c ≥ 3

Let's take c = 2

2n^3 + 5n ≤ 2* n^3

2 + 5/n^2 ≤ 6

2 + 5/n^2 ≤ 6 for n0 ≥ 0 is True

Hence, f(n) = O(n^3).

Section 2: Analysis

1. Give tight asymptotic bound for the following recurrences, show your work.

a. T(n) = 4T(n/2) + 1

Using Master Formula:

a = 4

b = 2

c = 1

k = 0

This satisfies the condition a > b ^ k which means T(n) = Theta (n ^ log (a/b)) = Theta (n ^ log2) = **Theta (n ^ 2)**

b. T(n) = T(n/2) + n ^ 3

Using Master Formula:

a = 1

b = 2

c = 1

k = 3

which satisfies the condition a < b ^ k , T(n) = Theta (n ^ k) = Theta (n ^ 3)

2. Given an array A with m integer elements and an array B with n integer elements, where m≤n. There may be duplicate elements. We want to find whether every eleent of B is an element of A. Propose an algorithm to solve this problem in O(m log n) or better.

```
Algorithm isSubset(A, B, m, n)
  Input array A of m integers and array B of n integers
  Output True or False

  H <- Hastable of Size m
  for i <- 0 to m - 1 do
    Insert A[i] in H

  for i <- 0 to n - 1
    if B[i] is not in H
      return False

  return True
```

Time Complexity O(m) + O(n) = O(m+n)

3. Suppose we are doing a sequence of operations (numbered 1, 2, 3, ...) such that the ith operation:

costs 1 if i is "not" a power of 2, and

costs i if i "is" a power of 2

For example, the following table shows the cost for each of the few operatios

Operation#	1	2	3	4	5	6	7	8	9	...
cost	1	2	1	4	1	6	1	8	1	....

Give the amortized cost per operation:

Solution:

Ammortized cost per operation is summation of all the costs divide by n

cost = 1 + 2 + 1 + 4 + 1 + 6 + 1 + 8 + 1 + ...

= (1 + 1 + 1 + 1+ 1+ ....) + 2 (1 + 2 + 3 + 4+ ...)

= Sum of all 1 from i as 1 to n + 2 * Sum of all i from i as 1 to n

= n + 2 * n(n+1)/2

= n + n ^ 2 + n = n ^ 2 + n

Ammortized cost per operation is (n ^ 2 + n) / n = n + 1 = O(n)

4: Use the asymptotic definition of Big-O to prove that the order of f(n) = n ^ 2 + 10n is O(n^2).

Solution:

f(n) = n ^ 2 + 10n is O(n ^ 2) iff f(n) ≤ c * g(n) for n ≥ n0 and c > 1

n^2 + 10n ≤ cn^2

(n^2 + 10n) / n^2 ≤ c

1 + 10/n ≤ c

take n0 = 1

c ≥ 1 + 10 = 11

n^2 + 10n ≤ 11 n ^ 2

⇒ 10n ≤ 10 n ^ 2 True

Thus: f(n) = n ^ 2 + 10n is O(n^2).

Section 3: Sorting Algorithms:

1. Answer the following questions by True or False:

a. Bubble Sort algorithm is a divide and conquer sort algorithm. **False**

b. Selection Sort makes more swap operations than Merge Sort. **True**

c. The worst running time for Quick Sort algorithm is O(n^2). **True**

d. Insertion sort is unstable sort algorithm. **False**

2. Which of the following trees is a max heap? Explain your answer for all given trees:

Condition to be a max heap:

- has to be a complete binary tree
- Each node is greater than or equal to the values in the children of that node.

A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

3. Given the Selection Sort algorithm below, What is the partially sorted array after three complete passes of insertion sort? Given that you have the following array of integers:

[15, 50, 4, 11, 22, 17, 14, 10, 8, 20]

```
void selectionSort(){
  int len = arr.length;
  int temp = 0;
  for(int i = 0; i<len; ++i){
    int nextMinPos = minpos(i, len-1);
    swap(i, nextMinPos);
  }
}
```

```
void swap(int i, int j){
  int temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}

int minpos(int bottom, int top){
  int m = arr[bottom];
  int index = bottom;
  for(int i = bottom + 1; i<=top; ++i){
    if(arr[i] < m){
      m = arr[i];
      index = i;
    }
  }
  return index;
}
```

First Pass: 4, 50, 15, 11, 22, 17, 14, 10, 8, 20

Second Pass: 4, 8, 15, 11, 22, 17, 14, 10, 50, 20

Third Pass: 4, 8, 10, 11, 22, 17, 14, 15, 50, 20

4. Sort the given array using Quick Sort. Pivots in Quick Sort are chosen using Median of Three strategy. When you use Median of Three, the pivot is found by computing the median of the 0th, last, and middle values in the input list.

Array: [15, 50, 4, 11, 22, 17, 14, 10, 8, 20]

Median of three: 15, 22, 20. So, the median is 20.

Step 1: 20 is the pivot. Move all elements greater than 20 to the right.

[15, 4, 11, 17, 14, 10, 8 → 4, 8, 10, 11, 14, 15, 17]   [50, 22 → 22, 50]

pivot = (15, 17, 8)

Step 2: 15 is the pivot. Move all elements greater than 15 to the right.

[4, 11, 14, 10, 8 → 4, 8, 10, 11, 14]   [17 → 17]

pivot = (4, 14, 8)

Step 3: 8 is the pivot. Move all elements greater than 8 to the right.

[4 → 4]   [11, 14,10 → 10, 11, 14] pivot = (11, 14, 10)

Step 4: 11 is the pivot Move all elements greater than 11 to the right.

[10 → 10]   [14 → 14]

Sorted array: [4, 8, 10, 11, 14, 15, 17, 20, 22, 50]

5. What is the worst-case running time for quick-sort?

The worst case running time for quick sort is Omega(n^2).

6. Explain why and how the worst case for quick-sort happens?

The efficience of the Quicksort algorithm depends on the selection of the pivot element. If we select a pivot element which is either maximum or minimum value in the array, then we will have an unbalanced array with one of L(less than pivot element) and G (greater than pivot element) has size n-1 and the other has size 0.

7. How can we avoid the worst-case scenario?

We can avoid the worst-case scenario in Quicksort by choosing an by making a "good self-call". Good self call means that the sizes of L and G are each less than 3n/4.

There can be a various ways to do this. We can either pick an item in the middle of an array, or selection of random pivot elements or we can choose median of three pivot candidates. Median of three is considered to be a good way, because in this we take a left most element, middle element, and right most element from the input array, and then we arrange them to left partition, pivot element, and right partition.