**EECS2030 Fall 2019** **Name: (Last, First)** ______________________
**Advanced OOP**
**Exam Practice Written Questions** **Student ID** ______________________
**Solution**

# 1 Written Exercises

1. Consider the following classes of functions:
   - $O(n)$
   - $O(log(n))$
   - $O(n^2)$
   - $O(1)$
   - $O(2^n)$
   - $O(n^3)$
   - $O(n \cdot log(n))$

   Say each of the above functions maps from input size $n$ to the *approximated* algorithm running time. Sort, from left to right, the above classes of functions from the cheapest to the most expensive. **Caution:** You will lose **all** marks if the order is not completely correct.

   **Solution:** $O(1)$ $O(log(n))$ $O(n)$ $O(n \cdot log(n))$ $O(n^2)$ $O(n^3)$ $O(2^n)$

   [ of 10 marks]

2. Consider the following statements:

   (**A**) $3n + 7$ is $O(n \cdot log(n))$
   (**B**) $3n + 7$ is $O(n)$
   (**C**) $3n + 7$ is $O(1)$
   (**D**) $3n + 7$ is $O(2^n)$
   (**E**) $3n + 7$ is $O(log(n))$
   (**F**) $3n + 7$ is $O(n^2)$

   (a) Which of the above statement or statements are *correct*?

   **Solution:** Statements **A** **B** **D** **F**

   [ of 10 marks]

   (b) Among the above statement or statements that are *correct*, which one is the most *accurate*?

   **Solution:** Statement **B**

   [ of 5 marks]

   (c) Justify your answer to the previous question. That is, clearly explain why it is more *accurate* than all other *correct* statements.

   **Solution:** The highest power of $n$ in $3n + 7$ is one. So Statement B is the most accurate by saying that $3n + 7$ is $O(n)$. The class $O(n)$ is strictly contained by $O(n \cdot log(n))$, which is strictly contained by $O(n^2)$, which is strictly contained by $O(2^n)$.

   [ of 10 marks]

3. In order to prove that $f(n) = 4n^3 - 5n^2 + 59 + n^4 + 9n$ is $O(n^4)$, you need to choose values for two constants: constant $c$ as a factor for $n^4$ and constant $n_0$ as some starting value of $n$.

   (a) Write down the precise condition for which $c$ and $n_0$ must satisfy in order for the proof to succeed. **Hint:** Your answer should involve $n^4$, $f(n)$, $c$, and $n_0$.

   **Solution:**

   $$c \cdot n^4 \geq f(n) \quad \text{for } n \geq n_0$$

   [ of 5 marks]

   (b) Give values of $c$ and $n_0$ that will complete the proof.

   **Solution:** Choose $c = 78$ and $n_0 = 1$.

   [ of 5 marks]

4. Consider the following Java program:

```java
1 void prog(int[] a, int n)
2   for (int i = 0; i < n; i++) {
3     for (int j = i; j < n; j++) {
4       for (int k = j; k > 0; k--) {
5         System.out.println(i * j + k);
6       }
7     }
8   }
```

Determine the **most accurate** asymptotic upper bound of the above program, using the big-Oh notation. You **must** show in detail how you determine the bound. Without a convincing derivation process, you will only receive partial marks.

**Solution:**

- Line 5 is a primitive operation that requires some constant running time: $O(1)$. Therefore, the overall running time can be determined by the number of times this print statement is executed: this can be determined by changes of the loop counters $i$, $j$, and $k$.
- From Line 2, we know that the body of the outer loop will run $n$ times.
- From Line 3, we know that:
  - 1st iteration of outer-most loop where $i = 0$, body of the middle loop runs with:
    * $j = 0$: the inner loop does not run [ 0 iteration ]
    * $j = 1$: the inner loop runs with $k = 1$ [ 1 iteration ]
    * $j = 2$: the inner loop runs with $k = 2, 1$ [ 2 iterations ]
    * $j = 3$: the inner loop runs with $k = 3, 2, 1$ [ 3 iterations ]

      ...
    * $j = n - 1$: the inner loop runs with $k = n - 1, n - 2, \ldots, 1$ [ $n - 1$ iterations ]

    **Subtotal # of iterations when** $i = 0$: $\boxed{\frac{(0+(n-1))\times(n-0)}{2}}$
  - 2nd iteration of outer-most loop where $i = 1$, body of the middle loop runs with:
    * $j = 1$: the inner loop runs with $k = 1$ [ 1 iteration ]
    * $j = 2$: the inner loop runs with $k = 2, 1$ [ 2 iterations ]
    * $j = 3$: the inner loop runs with $k = 3, 2, 1$ [ 3 iterations ]

      ...
    * $j = n - 1$: the inner loop runs with $k = n - 1, n - 2, \ldots, 1$ [ $n - 1$ iterations ]

    **Subtotal # of iterations when** $i = 1$: $\boxed{\frac{(1+(n-1))\times(n-1)}{2}}$
  - 3rd iteration of outer-most loop where $i = 2$, body of the middle loop runs with:
    * $j = 2$: the inner loop runs with $k = 2, 1$ [ 2 iterations ]
    * $j = 3$: the inner loop runs with $k = 3, 2, 1$ [ 3 iterations ]

      ...
    * $j = n - 1$: the inner loop runs with $k = n - 1, n - 2, \ldots, 1$ [ $n - 1$ iterations ]

    **Subtotal # of iterations when** $i = 2$: $\boxed{\frac{(2+(n-1))\times(n-2)}{2}}$

    ...
  - $n^{th}$ iteration of outer-most loop where $i = n - 1$, body of the middle loop runs with:
    * $j = n - 1$: the inner loop runs with $k = n - 1, n - 2, \ldots, 1$ [ $n - 1$ iterations ]

    **Subtotal # of iterations when** $i = 2$: $\boxed{\frac{((n-1)+(n-1))\times(n-(n-1))}{2}}$
- Adding the above subtotal numbers of iterations:

$$\sum_{i=0}^{n-1} \frac{(i + (n-1)) \times (n-1)}{2} = \sum_{i=0}^{n-1} \underbrace{\frac{n^2 + (i-2) \cdot n + 1}{2}}_{T}$$

- To obtain the asymptotic upper bound, we drop multiplicative constants and lower terms:

$$O(\sum_{i=0}^{n-1} n^2) = O(n \cdot n^2) = O(n^3)$$

- Therefore, the running time of the above algorithm is $O(n^3)$.

[ of 15 marks]

5. Consider the following Java code:

```java
1  boolean isSorted(int[] a) {
2    return isSortedHelper(a, 0, a.length - 1);
3  }
4  boolean isSortedHelper(int[] a, int from, int to) {
5    if (from > to) {
6      return true;
7    }
8    else if(from == to) {
9      return true;
10   }
11   else {
12     return a[from] <= a[from + 1]
13       && isSortedHelper(a, from + 1, to);
14   }
15 }
```

Prove, via mathematical induction, that the method `isSorted` method above correctly returns `true` if the array `a` is sorted in a non-descending order; and `false` otherwise.

**Solution:**

We first prove that the recursive helper method `isSortedHelper` (Line 4 – Line 15) is correct (i.e., is the subarray {`a[from]`, `a[from + 1]`, . . . , `a[to]`} sorted).

1. **Base Cases**
   (a) **Concept**: In an empty array, there is no witness (i.e., adjacent numbers that are not sorted) ∴ result is **true**.
   (b) **Link to Code**: **Lines 5 – 7 (or just Line 6)** of the above code does this.
   (c) **Concept**: In an array of size 1, the only one element is automatically sorted.
   (d) **Link to Code**: **Lines 8 – 10 (or just Line 9)** of the above code does this.
2. **Inductive Cases**
   (a) **Inductive Hypothesis (I.H.)**: The recursive call `isSortedHelper(a, from + 1, to)` returns **true** if a[from + 1], a[from + 2], . . . , a[to] are sorted in a non-descending order; **false** otherwise.
   (b) **Concept**: `isSortedHelper(a, from, to)` should return **true** if:
       **1)** `a[from]` $\leq$ `a[from + 1]`; and
       **2)** the subarray {`a[from + 1]`, . . . , `a[to]`} is sorted.
   (c) **Link to I.H.**: By I.H., condition **2)** is satisfied.
   (d) **Link to Code**: **Line 12** in the above code does condition **1)**.
       ∴ **Lines 12 – Line 13** perform a correct combination.
3. Given that the recursive helper method `isSortedHelper` (Line 4 – Line 4) is correct, we now argue that the method `isSorted` (Line 1 – Line 3) is correct.
   (a) **Concept**: `isSorted(a)` is correct by invoking `isSortedHelper(a, 0, a.length - 1)`, examining the entire array.
   (b) **Link to Code**: **Line 2** of the above code does this.

[ of 20 marks]