CSE 214 SUMMER 2021

1. [5 minutes] Write the order of complexity for the following programs using Big O notation if the programs execute the following number of operations for n inputs:

a) $6n + 7$ **b) $4n + 5n^2 + n^4$** **c) $7n^2(\log_2 n) + 8n$**
d) $2^n + n^{1000}$ **e) 1900** **f) $((\log_2)^3 n) + n/8$**

Answer: a) O(n) b) $O(n^4)$ c) $O(n^2(\log_2 n))$
d) 2^n e) O(1) f) O(n)

a. $O(6n + 7) = O(6n) = O(n)$ ∵ Drop coefficient of term with largest order of power on n
b. $O(4n + 5n^2 + n^4) = O(n^4)$ ∵ Use term with largest order of power on n
c. $O(7n^2 \log_2 n + 8n) = O(7n^2 \log_2 n) = O(n^2 \log_2 n)$ ∵
Drop coefficient of term with largest order of power on n
d. $O(2^n + n^{1,000}) = O(2^n)$ ∵ Use term with largest order of power on n

Proof:

$$\frac{2^{n+1}}{2^n} = 2^{n+1-n} = 2^1 = 2$$

$$\frac{(n+1)^{1,000}}{n^{1,000}} = \left(\frac{n+1}{n}\right)^{1,000} = \left(1 + \frac{1}{n}\right)^{1,000}$$

$$\lim_{n\to\infty} 2 = 2$$

$$\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^{1,000} = \left(1 + \frac{1}{\infty}\right)^{1,000} = (1+0)^{1,000} = 1^{1,000} = 1$$

$$2 > 1 \Rightarrow 2^n > n^{1,000} \text{ as } n \to \infty$$

$$\therefore 2^n > n^{1,000} \text{ for sufficiently large } n$$

e. $O(1{,}900) = O(1 \cdot 1{,}900) = O(1)$ ∵ Drop coefficient of term with largest order of power on n
f. $O\left(\log_2(n)^3 + \frac{n}{8}\right) = O\left(\frac{n}{8}\right) = O(n)$ ∵ Use term with largest order of power on n

2. [10 minutes] Write the complete specification for the following Java method:

```
public static double farToKel(double degFar)
{   if(degFar < -459.67)
        throw new IllegalArgumentException("Temperature too low");
    return (((degFar - 32) * 5/9) + 273.15);
}
```

Answer:
public static double farToKel(double degFar)

A method that converts a temperature in Far. to a temperature in Kelvin.

Parameters:
degFar- the temperature to be converted

Precondition:
degFar is a valid Far. temperature (above -459.67 degrees).

Returns: degFar converted into a Kelvin temperature.

Throws:
IllegalArgumentException- indicates that degFar is below -459.67.

3. [15 minutes] An array A of size n contains n unique integers in the range [0,n]. (That is, there is one number from this range that is not in A.)

(a) [7.5 minutes] Write psuedocode for an O(n)-time algorithm for finding that number. Use an additional array of size n+1 to help you. Verify that the algorithm is O(n).

(b) [7.5 minutes] Write pseudocode for an O(n)-time algorithm for finding that number, but use only a constant amount of additional space besides the array A itself. Verify that the algorithm is O(n) and the extra space used is O(1).

Answer:
(a) Set all elements in the new array B to false (0). Then for each A[i],
set B[A[i]] to true (1). Then scan array B for the single element that remains false.
(b) Add all the elements in A. Subtract this sum from n(n+1)/2 to find the missing number

4. [5 minutes] What is the number of operations (assignment statements) that occur in the following code fragment in closed form in terms of n? In Big O notation? (Note: Ignore assignment operators inside loop headers and you may assume n is a power of 2)

```
int d = 0;
for(int i = 1; i <= n; i++)
{   for(int j = 1; j <= i; j++)
        d = d + i + j;
    d = d + 2;
}
for(int k = 1; k < n; k = k*2)
{   d = d - 5;
}
```

Answer: *Closed:* $((n^2 + n)/2 + n) + (\log_2 n)) + 1$
Big O: $O(n^2)$

5. [20 minutes] Write a javadoc for following java code:

```
package jb;
import java.util.NoSuchElementException;
```

```
import java.util.ArrayList;

public class Stack
{
	Public void push(Object item) {this.elmenet.add(item);}

	public Object Pop() throws NoSuchElementException
	{
		int length = this.elements.size();
		if (length == 0) throw new NoSuchEelementException();
		return this.elements.remove(length – 1);
	}
	public Object peek() throws NoSuchElementException
	{
		int length = this.element.size();
		if (length == 0) throw new NoSuchElementException();
		return this.elements.get(length – 1);
	}
	public boolean isEmpty() { return this.elements.isEmpty();}
	private ArrayList elements = new ArrayList();

}
```

Answer:

```
package jb;
import java.util.NoSuchElementException;
import java.util.ArrayList;

/**
* The Stack class represents a last-in-first-out stack of objects.
* @author M. M. Javanmard
* @version 1.0, July 2016
* Note that this version is not thread safe.
*/
public class Stack
{
/**
* Pushes an item on to the top of this stack.
* @param item the item to be pushed.
*/
Public void push(Object item) {this.elmenet.add(item);}

/**
* Removes the object at the top of this stack and returns that object.
* @return The object at the top of this stack.
* @exception NoSuchElementException if the stack is empty.
	*/
```

```
public Object Pop() throws NoSuchElementException
{
int length = this.elements.size();
if (length == 0) throw new NoSuchEelementException();
return this.elements.remove(length 1);
}

/**
* Returns the object at the top of this stack without removing it.
* @return the object at the top of this stack.
* @exception NoSuchElementException if the stack is empty.
*/
public Object peek() throws NoSuchElementException
{
int length = this.element.size();
if (length == 0) throw new NoSuchElementException();
return this.elements.get(length 1);
}

/**
* Tests if this stack is empty.
* @return true if this stack is empty and false otherwise.
*/
public boolean isEmpty() { return this.elements.isEmpty();}
private ArrayList elements = new ArrayList();
```

}

6. [10 minutes] Consider the following code fragment:

```
int[] myArr = new int[n][n];
int[] mySecondArr = new int[n][2*n];
for i = 1 to n {
    for j = 1 to n {
        myArr[i][j] = i*i;
        mySecondArr[i][2*j] = i * j;
        mySecondArr[i][2*j+1] = i * 2 * j;
    }
    for j = 1 to n*n {
        myArr[j/n][j%n] += j;
    }
}
```

a.) What is the order of complexity of the runtime?
b.) What is the order of complexity of the memory usage?

Answer:

a) O(n^3)

b) O(n^2)