Lab solution

Problem 1:

a. TRUE: $4n^3 + n$ is $\Theta(n^3)$ because

$$\lim_{n\to\infty} \frac{4n^3 + n}{n^3} = \lim_{n\to\infty} (4 + n^{-2}) = 4$$

b. TRUE: $\log n$ is $o(n)$ because

$$\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{(1/n)\log e}{1} = \lim_{n\to\infty} \frac{\log e}{n} = 0$$

The first equality uses the L'Hôpital's rule for finding limit.

c. TRUE: 2^n is $\omega(n^2)$ because

$$\lim_{n\to\infty} \frac{n^2}{2^n} = \lim_{n\to\infty} \frac{2n^1}{2^n \ln(2)} = \lim_{n\to\infty} \frac{2n^0}{2^n ln2ln2} = \lim_{n\to\infty} \frac{2}{2^n ln2ln2} = 0$$

Where I have used the L'Hopital's rule twice.

d. TRUE: 2^n is $o(\,3^n)$ because

$$\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} (0.666..)^n = 0$$

Problem 2:

Show that for all $n > 4$, $2^n < n!$

1. **Base case**: is true because for n = 5, $2^n < n! \; \equiv 32 < 120$
2. **Assumption for some integer n**: For some n, $2^n < n!$
3. **Checking if it applies for n+1**: For n+1, $2^{n+1} = 2 * 2^n < 2 * n! < (n+1) * n! = (n+1)!$

Where the second inequality follows from the assumption (2)

Problem 3:

The code uses Euclidean Algorithm

```java
/**
 * GreatestCommonDivisor
 */
public class GreatestCommonDivisor {
    public static void main(String[] args) {
        System.out.println(gcd(12, 42));
        System.out.println(gcd(7, 9));
    };

    // Using Euclidean Algorithm
    public static int gcd(int a, int b){
        if(a == 0) return b;
        if(b == 0) return a;
        return gcd(b, a%b);
    }
}
```

Problem 4:

```java
public class SecondSmallest {
    public static int secondSmallest(int[] arr) {
        if(arr==null || arr.length < 2) {
            throw new IllegalArgumentException("Input array too small");
        }
        int smallest = arr[0];
        int secondSmallest = arr[1];
        for(int z: arr){
            if(z == smallest)
                continue;
            if(z < smallest){
                secondSmallest = smallest;
                smallest = z;
                continue;
            }
            if(z< secondSmallest)
                secondSmallest = z;
        }
        return secondSmallest;
        }
}
```

Problem 5:

1. First approach will be getting all the possible combinations (subsets) of the set and check if any of them sum up to the target sum. So, the java code would be to get all the subsets of S, and for each subset, calculate the sum and compare it to k. if the sum equals k, then return that subset otherwise after all the process is done, return null.
2. Another approach is a recursive approach by considering two cases. The first case adds the nth element to a subset and moves on to see what will happen to the rest. The second case doesn't consider the nth element and moves on to see if we can find a subset that sums to k with the rest.

Here is the implementation of (2)

```java
package problem05;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

// A recursive solution for subset sum
// problem
class SubsetSum {

	// Returns true if there is a subset
	// of set[] with sum equal to given sum
	static List<Integer> isSubsetSum(List<Integer> set,
							int n, int sum)
	{
//		List<Integer> list = new ArrayList<>(indices);
		// Base Cases
		if (sum == 0)
			return new ArrayList<>();
		if (n == 0 && sum != 0)
			return null;

		// If last element is greater than
		// sum, then ignore it
		if (set.get(n - 1) > sum)
			return isSubsetSum(set, n - 1, sum);

		/* else, check if sum can be obtained
		by any of the following
			(a) including the last element
			(b) excluding the last element */

		List<Integer> include = isSubsetSum(set, n - 1, sum);
		if(include == null) {
			List<Integer> exclude = isSubsetSum(set, n - 1, sum - set.get(n - 
1));
			if(exclude == null)
				return null;
			exclude.add(set.get(n-1));
			return exclude;
		}
		return include;
```

```
        }

        /* Driver program to test above function */
        public static void main(String args[])
        {
                List<Integer> set = Arrays.asList(3,34,4,12,2);
//              set = Arrays.asList(2, 1, 4, 12, 15, 2);
                Integer sum = 9;
                int n = set.size();
                System.out.println(isSubsetSum(set, n, sum));
        }
    }
```

Source: https://stackoverflow.com/questions/58417757/return-elements-of-array-in-subset-sum-problem