Lab 2

(1) Determine the asymptotic running time of the following procedure (an exact computation of number of basic operations is not necessary):

```
int[] arrays(int n) {
    int[] arr = new int[n];
    for(int i = 0; i < n; ++i){
       arr[i] = 1;
    }
    for(int i = 0; i < n; ++i) {
        for(int j = i; j < n; ++j){
            arr[i] += arr[j] + i + j;
        }
    }
    return arr;

}
```

Solution. The first for loop takes $O(n)$. The second (nested) for loop requires $O(n^2)$. Asymptotic running time: $O(n) + O(n^2) = O(n^2)$.

(2) See the Java file Merge.java. It is easy to see that there is essentially just one loop depending on n (the sum of the lengths of the two input arrays), so running time is $O(n)$.

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

(b) $\log n$ is $o(n)$.

(c) 2^n is $\omega(n^2)$.

(d) 2^n is $o(3^n)$.

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

Solution: Since both polynomials have the same degree, the result follows by the Polynomial Theorem.

(b) $\log n$ is $o(n)$.

Solution:

$$\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{\frac{\log e}{n}}{1} = \lim_{n\to\infty} \log e \cdot \frac{1}{n} = 0.$$

(c) 2^n is $\omega(n^2)$.

Solution:

$$\lim_{n\to\infty} \frac{n^2}{2^n} = \lim_{n\to\infty} \frac{2n}{2^n \cdot \ln 2} = \lim_{n\to\infty} \frac{2}{2^n \cdot \ln^2 2} = 0.$$

(d) 2^n is $o(3^n)$.

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

(4) Power Set: See the Java file PowerSet.java, reproduced here:

```
public static <T> List<Set<T>> powerSet(List<T> X) {
  List<Set<T>> P = new ArrayList<Set<T>>();
  Set<T> S = new HashSet<T>();
  P.add(S);
  if(X.isEmpty()) {
    return P;
  }
  else {
    while(!X.isEmpty()) {
      T f = X.remove(0);
      List<Set<T>> temp = new ArrayList<Set<T>>();
      for(Set<T> x : P) {
        temp.add(x);
      }
      for(Set<T> x : temp) {
        S = new HashSet<T>();
        S.add(f);
```

```
        S.addAll(x);
        P.add(S);
      }
    }
  }
  return P;
}
```

(5) The removeDups algorithm shown in the slides contains nested loops, both depending on input size, so the running time is $\Theta(n^2)$.

```
public static List<Integer> removeDups(List<Integer> list) {
  final Integer ZERO = new Integer(0);
  HashMap<Integer, Integer> h = new HashMap<>();
  List<Integer> retval = new LinkedList<>();
  for(Integer x : list) {
    if(!h.containsKey(x)) {
      h.put(x, ZERO);
      retval.add(x);
    }
  }
  return retval;
}
```