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) **Solution.**

$$\begin{aligned} T(n) &= T(n-1) + T(n-1) + T(n-2) + d \\ &\geq 3T(n-2) + d \\ &\geq 3T(n-2) \end{aligned}$$

(a) **Lemma**. Suppose $T(n)$ satisfies $T(1) = c$, $T(n) \geq 3T(n-2)$. Define a recurrence relation for $S(n)$ by $S(1) = c, S(n) = 3S(n-2)$. Then for all $n \geq 1$, $T(n) \geq S(n)$.

Proof. Proceed by induction on n to show T(n) $\geq$ S(n). This is obvious for `n = 1`. Assume T(k) $\geq$ S(k) whenever `k < n`. Then T(n) $\geq$ 3T(n-2) $\geq$ 3S(n-2) = S(n)

(b) Use guessing method to show S(n) is $\Theta\left(\sqrt{3}\right)^n$

$$\begin{aligned} S(1) &= c \\ S(3) &= 3 \cdot S(1) = 3 \cdot c \\ S(5) &= 3 \cdot S(3) = 3 \cdot 3 \cdot c = 3^2 c \\ S(7) &= 3 \cdot S(5) = 3 \cdot 3 \cdot 3 \cdot c = 3^3 c \\ S(9) &= 3 \cdot S(7) = 3 \cdot 3 \cdot 3 \cdot 3 \cdot c = 3^4 c \\ S(n) &= 3^{n/2} \cdot c = (\sqrt{3})^n c, \text{ which is } \Theta\big((\sqrt{3})^n\big) \end{aligned}$$

(c) Verify that the fomular S(n) = $3^{n/2} \cdot c$ is a solution to the recurrence S(1) = c, S(n) = 3S(n-2).

Proof.
For n = 1, we have S(1) = $3^{1/2} \cdot c = c$
In general,
3S(n-2) = $3 \cdot 3^{n-2/2} \cdot c = 3^{n/2} \cdot c$ = S(n)
as required.

(d) By guessing method, we know that S(n) is $\Theta\left(\sqrt{3}\right)^n$. By part (a), we know $T(n) \geq S(n)$, therefore T(n) is $\Omega\left(\sqrt{3}\right)^n$

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

(5) Devise an iterative algorithm for computing the Fibonacci numbers and compute its running time.

Below is an iterative Java method that computes F_n on input n. It executes a single loop that depends on n, so running time is O(n). It also uses O(n) space, using the `store` array.

```
int[] store;
public int fib(int n) {
  store = new int[n+1];
  store[0] = 0;
  store[1] = 1;
```

```
    //store[i] = F_i
    for(int i = 2; i <= n; i++) {
      store[i] = store[i-1] + store[i-2];
    }

    return store[n];
  }
```

(6) We use the Master Formula to solve this recurrence relation:

$$T(n) = \begin{cases} 1 \text{ if } n = 1 \\ T(n/2) + n \text{ otherwise} \end{cases}$$

Here, $a = 1, b = 2, c = 1, d = 1, k = 1$. Note $a < b^k$. The Master Formula tells us therefore that $T(n) \in \Theta(n)$.