1. Give the pseudo-code of a logarithmic-time ($\theta(logn)$-time) algorithm for computing the n-th Fibonacci number.

피보나치 수열 F_n에 대해 다음이 성립한다 ($F_0 = 0, F_1 = 1$)

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} \to \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} \to \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

따라서 의사코드는 다음과 같다

```
exp(A : Matrix, n : int):
    if n == 1:
        return A
    A = exp(A, n/2)
    return A * A

fibo(n : int):
    A = exp([[1, 1], [1, 0]], n - 1)
    A = A * [1, 0]
    return A[0]
```

2. Prove that $n^2 \in o(2^n)$ using the formal definition of small-o notation.

엡실론 델타 논법

$$\lim_{x\to\infty} f(x) = L \Leftrightarrow \forall\epsilon > 0,\ \exists\delta > 0,\ x > \delta \to |f(x) - L| < \epsilon$$

$small - o$의 정의

$$o(f(x)) = g(x) \mid \forall c > 0,\ \exists N > 0,\ \forall x > N,\ g(x) < cf(x)$$

로피탈의 정리

$$\lim_{x\to\infty} f(x) = \infty,\ \lim_{x\to\infty} g(x) = \infty,\ f, g\text{는미분가능},\ \exists \lim_{x\to\infty} \frac{f'(x)}{g'(x)} \text{ 이면}$$

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$$

보조정리 1 임의 x에 대해 $x > a \to p(x) \Leftrightarrow \forall x > a\ \ p(x)$

보조정리 2 : $g(x) > 0,\ f(x) > 0$ 일 때 $\lim_{x\to\infty} \frac{g(x)}{f(x)} = 0 \to g(x) \in o(f(x))$

보조정리 2 증명 :

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = 0$$

$$\to \forall\epsilon > 0,\ \exists\delta > 0,\ x > \delta \to \frac{g(x)}{f(x)} < \epsilon \text{ (엡실론 델타 논법)}$$

$$\to \forall\epsilon > 0,\ \exists\delta > 0,\ \forall x > \delta,\ g(x) < \epsilon f(x) \text{ (보조정리 1)}$$

$$\to g(x) \in o(f(x)) \text{ (small-o의 정의)}$$

본 문제 증명:

$$\lim_{x \to \infty} \frac{n^2}{2^n} = \lim_{x \to \infty} \frac{2n}{ln2\ 2^n} = \lim_{x \to \infty} \frac{2}{(ln2)^2 2^n} = 0 \text{ (로피탈의 정리)}$$

$$\lim_{x \to \infty} \frac{n^2}{2^n} \text{ 이므로 보조정리 } 2\text{에 의해 } n^2 \in o(2^n)$$

3. Give the closed forms for the following recurrence relations. Solve them using their characteristic equations.

(3a) $f_{n+2} = f_{n+1} + f_n \ (n \geq 0),\ f_0 = 0,\ f_1 = 1$

위 재귀 관계의 특성 방정식과 해는 아래와 같다

$$r^2 - r - 1 = 0,\ r = \frac{1 \pm \sqrt{5}}{2}$$

따라서 재귀관계를 풀면 아래와 같다

$$f_n = \alpha_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + \alpha_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

$$f_0 = 0, f_1 = 1 \text{ 이므로, } \alpha 1 + \alpha 2 = 0$$

(3b) $f_{n+2} = f_{n+1} + f_n + 1 \ (n \geq 0),\ f_0 = 0,\ f_1 = 1$

(3c) $T(n) = 3T(n/3) + 2n/3,\ T(1) = 0 \ (\text{You may assume that } n = 3^k,\ k \geq 0)$

(3d) $T(n) = 3T(n/3) + 2n,\ T(1) = 0 \ (\text{You may assume that } n = 3^k,\ k \geq 0)$

4. Consider MergeSort algorithm in the textbook.

(4a) Given array size n, find a recurrence relation for the best-case time complexity for MergeSort.

교재의 병합정렬 알고리즘은 아래와 같다

1. 1개의 원소가 될 때까지 배열 반으로 쪼개기
2. 1의 것들을 정렬하며 병합
   2.1. 병합시 두 배열의 첫 부터 시작해서 각각 비교, 순회하며 정렬 순서로 채워넣음

1은 배열이 무엇이든 반으로 나누므로 길이가 같으면 수행 횟수는 고정이다
2의 경우는 두 배열의 상태에 따라 수행시간이 달라진다 최선의 경우는 이미 정렬된 두 배열 A, B를 병합하는 것으로 이 경우는 $min(|A|, |B|)$의 시간이 소요된다

최선의 경우 반씩 쪼개지고 병합되므로 배열의 길이 n에 대한 최선 시간복잡도 $B(n)$은 아래와 같다

$$B(n) = 2B(\frac{n}{2}) + \frac{n}{2}$$

(4b) Solve the recurrence relation for (1), given n (= 2k for some integer k > 0).

5. (Theoretically Fast QuickSort) There exists a linear-time ($O(n)$-time) algorithm that computes a median value among given n values. Assume that we have already known this algorithm. Using this algorithm

(5a) design a variant QuickSort algorithm of which the worse-case time complexity is $O(nlogn)$ (just give its pseudo-code)

선형시간에 중앙값의 위치를 구할 수 있는 알고리즘을 Median이라 하자
의사코드는 아래와 같다

```
Partition(A : array):
    pivot, p_i = A[0], 0
    for i in range(len(A)):
        if A[i] < pivot:
            p_i += 1
            swap(A[i], A[p_i])
    swap(A[p_i], A[0])

FastQuickSort(A : array, low : int, high : int):
    i = Median(A)
    swap(A[0], A[i])
    Partition(A)
    FastQuickSort(A, low, i)
    FastQuickSort(A, i, high)
```

(5b) prove that its time complexity is $O(nlogn)$
$O(n)$으로 중앙값을 구하고 분할한뒤 반으로 쪼개진다
분할 시간은 배열 상태와 상관 없이 배열의 크기이므로
배열 크기 n에 대한 시간복잡도 $T(n)$의 재귀관계는 아래와 같다

$$T(n) = 2T(\frac{n}{2}) + \frac{n}{2} + O(n)$$