Problem 4

I. Answer the following questions on logic circuits.

Consider n-input logic functions $f(x_1, x_2, \ldots, x_n)$ where $x_1, x_2, \ldots,$ and x_n $(n \geq 1)$ are logical variables.

(1) Show that any 1-input logic functions $f(x_1)$ can be expressed using only three kinds of logical operations: AND, OR, and NOT.

(2) Show that any n-input logic functions $f(x_1, x_2, \ldots, x_n)$ can be expressed using only three kinds of logical operations: AND, OR, and NOT.

(3) Show that any n-input logic functions $f(x_1, x_2, \ldots, x_n)$ can be expressed using NAND operations only.

Sequential circuit A is composed of three D flip-flops as shown in Fig. 1 and outputs a 3-bit signal $q_0 q_1 q_2$ in synchronization with a clock signal.

(4) Show the state transition table and the state transition diagrams of circuit A.

(5) When the initial value of each D flip-flop is zero, circuit A can be used as a 6-state counter. Describe the characteristics of this 6-state counter.

(6) Circuit A failed to operate as a correct 6-state counter due to a malfunction. Here, circuit A is modified to circuit B that can revert to the correct behavior of the 6-state counter after one clock signal even when a malfunction occurs. We realize circuit B by changing the signal for input D_0 as shown in Fig. 2. Describe the logical expression for the circuit inserted to X. The logical expression must be simplified as much as possible.

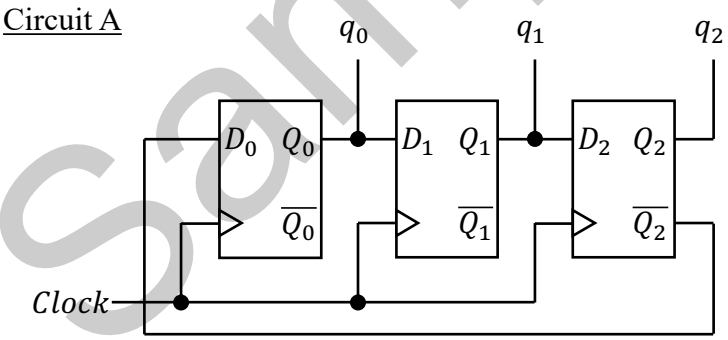

Fig. 1

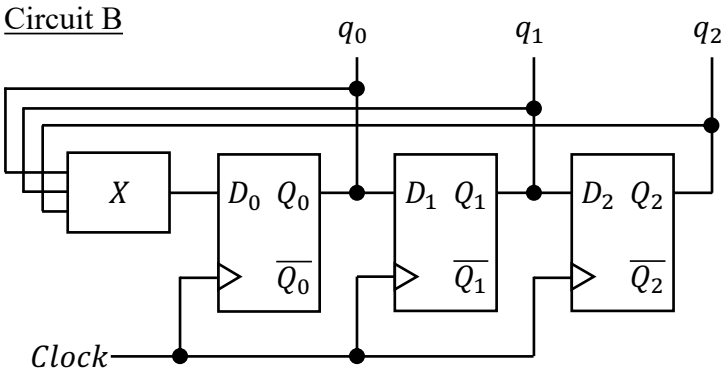

Fig. 2

II. Consider algorithms that sort a list of integers in ascending order. Answer the following questions.

(1) The merge operation combines two sorted lists and produces a new single sorted list. `Program 1` describes `merge` in the C programming language that merges two sorted lists `X` and `Y` which are stored in an array as shown in Fig. 3. Write appropriate codes for blanks (A) and (B).

(2) Consider implementing `sort1` that sorts an array of integers in ascending order using `merge`. Write an appropriate code for blank (C) in `Program 2`. You must define `sort1` by using recursive calls.

(3) For the number of elements `n`, find the orders of time complexity of `sort1` for the average case and the worst case.

```
/* Program 1 */
void merge(int data[], int low, int mid, int high) {
    int i, j, k;
    int tmp[high+1];

    for (i = low; i <= mid; i++) {tmp[i] = data[i];}
    for (i = mid+1, j = high; i <= high; i++, j--) {tmp[i] = data[j];}

    i = low; j = high;

    for (k = low; k <= high; k++) {
        if (tmp[i] <= tmp[j]) {
            (A)
        } else {
            (B)
        }
    }
}
```

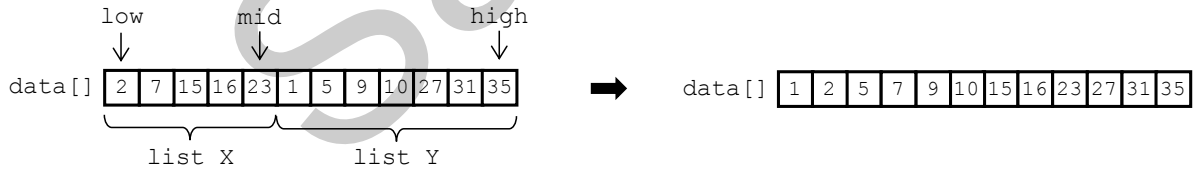

Fig. 3

```
/* Program 2 */
void sort1(int data[], int low, int high){
    int mid, i, j;

    if (low >= high) {return;}

    mid = (low + high)/2;

    (C)
}
```