Problem 4

I.

Answer the following questions on sequential circuits. We consider a circuit that has four states S={s0, s1, s2, s3}. The circuit changes its state and output Y according to the value of input X. Table 1 shows the state transition table of the circuit.

(1) Draw the state transition diagram that is equivalent to the state transition table in Table 1.

(2) Show the shortest input sequence that outputs Y=1 when the initial state is s2.

Next, assume that we represent the state S with two bits as s0=00, s1=01, s2=10, s3=11. We consider designing the circuit with two JK-Flip Flop (JK-FF) F_A and F_B. We represent the higher bit of the states with Q_A, and the lower bit with Q_B such as s=Q_AQ_B. F_A and F_B hold the states Q_A and Q_B in the circuits, respectively. A JK-FF has one bit state Q, and the state transits as shown in Fig. 1 according to inputs J and K.

(3) Copy Table 1 to your answer sheet and complete the table by filling the blanks. Here, (J_A, K_A) and (J_B, K_B) are the inputs to F_A and F_B, respectively. Use *d* for "don't care" conditions.

(4) Simplify the input functions of J_A, K_A, J_B, K_B and output function of Y by using Karnaugh maps.

(5) Draw the circuit using the symbols shown in Fig. 2.

Table 1

Current state S(t)	Input X	Next state S(t+1)	Output Y	$Q_A(t)$	$Q_B(t)$	$Q_A(t+1)$	$Q_B(t+1)$	J_A	K_A	J_B	K_B
s0	0	s1	0								
s0	1	s2	0								
s1	0	s3	0								
s1	1	s2	0								
s2	0	s0	0								
s2	1	s2	0								
s3	0	s0	0								
s3	1	s2	1								

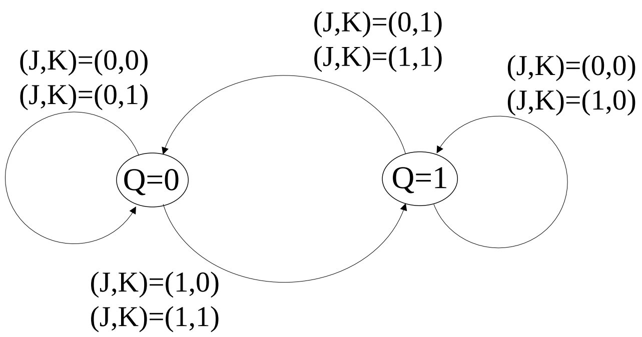

Fig. 1

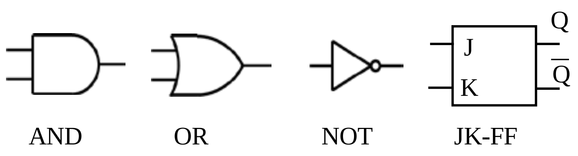

Fig. 2

II.

Answer the following questions on heapsort. Assume that max heap is an array representation of a binary tree where the value in a parent node is greater than the values in its two children nodes. The structure of the array is as follows: assuming that the index i is an integer greater than or equal to 1, the value in the left child node of node i is stored in the $(i \times 2)$-th element of the array, and the value in the right child node of node i is stored in the $(i \times 2 + 1)$-th element of the array.

The function `make_heap` in Program 1 is a function that generates the max heap from an `array` with `n` items. The first element of the array is stored in `array[1]`. The function `swap(array, i, j)` in Program 1 is a function that switches the values of `array[i]` and `array[j]`. Here, the values in the array are integers, and there is no duplication among them.

(1) Assume that the number of items is 7 (`n`=7), and the input `array` stores the values {1, 7, 8, 9, 6, 4, 5} in this order. Find the max heap that is generated by `make_heap` in Program 1.

(2) Draw the process of generating the max heap in Question (1) with binary trees. The arrays follow the same structure as the max heap even during the process of making max heap. You do not need to draw the trees that remain unmodified in the process.

(3) Give the order of the time complexity of `make_heap` according to the number of comparisons of the values.

Assume that we sort a max heap by iteratively taking the maximum value and rebuilding a max heap with the remaining items.

(4) Describe a function by using the C programming language in as few lines as possible to sort a max heap `array` with `n` items in descending order. Use the function `make_heap` in Program 1 for rebuilding the max heap in your function.

(5) Describe an overview of an algorithm to sort an array using max heap, which is more computationally efficient than that obtained in Question (4). The sort order can be either ascending order or descending order.

```
/* Program 1 */
void push_down(int array[], int n, int node) {
    if(node * 2 > n)    return;
    int    parent = node, child;
    do {
        child = parent * 2;
        if ((child < n) && (array[child] < array[child + 1]))
            child = child + 1;
        if (array[child] < array[parent])
            break;
        swap(array, child, parent);
        parent = child;
    }while (parent * 2 <= n);
}

void make_heap(int array[], int n){
    int    node;
    for (node = n / 2; node >= 1; --node)
        push_down(array, n, node);
}
```