Problem 0:

Problem 1: 羅煜翔、胡家榆、張晴昀

Problem 2: 羅煜翔

Problem 3: 許睿尹

Problem 4: All by myself.

Problem 1:

1(A).

	0	1	2	3	4	5	6
42	42						
11	42					11	
25	42					11	25
1	42			1		11	25
56	42	56		1		11	25
70	42	56	70	1		11	25
19	42	56	70	1	19	11	25

1(B).

	0	1	2	3	4	5	6
42	42						
11	42					11	
25	42		25			11	
1	42		25	1		11	
56	42	56	25	1		11	
70	42	56	25	1	70	11	
19	42	56	25	1	70	11	19

2.

For example, the light set is {A, B, C, D}, then the value would be 2^(A−1)+2^(B−1)+2^(C−1)+2^(D−1). In this way, the value for every set is unique, and since there are 17 rooms in this question, the maximum value would be 2^17−1 = 131071, which is smaller than the constraint 133333.

3.

```
sum = 0
max = 0
insert[0,0]
for i from 1 to n
    sum += energy_level[i]
    insert[sum, i]
    if get(sum−E) != NULL && i−get(sum−E) > max
        max = i − get(sum−E)
Return max
```

We run the for loop once so the time complexity is O(n), and since we insert n times, the space complexity is O(n).

4.
1/N * C(M,2)

5.
```
Merge_Heap(heap a, heap b)
    if(a.size > b.size)
        swap(a, b)
    while(!empty(a))
        insert(b, a.root)
        remove(a.root)
    return b
```

Assume that a node is always in the smaller heap, since every time we merge heaps, the size of the new heap would be at least double of the smaller one, we would insert the node for at most log(n) times. The time complexity for insert is O(log(n)), and we insert log(n) times, so the time complexity for every node is O(log^2(n)). There are n nodes in this problem, so the total time complexity is O(nlog^2(n)).

6.

```
Typedef struct hate{
    int id
    struct hate *next
}Hate
Typedef struct adj_list{
    struct hate *head
    struct hate *tail
}Adj_list
for i from 1 to N
    Adj_list[i].head = Adj_list[i].tail = NULL
for i from 1 to M
    a = arr[i][1], b = arr[i][2]
    hate A.id = a, A.next = NULL
    hate B.id = b, B.next = NULL
    if Adj_list[a].head = NULL
        Adj_list[a].head = Adj_list[a].tail = B
    else
        Adj_list[a].tail -> next = B
        Adj_list[a].tail = B
    if Adj_list[b].head = NULL
        Adj_list[b].head = Adj_list[b].tail = A
    else
        Adj_list[b].tail -> next = A
        Adj_list[b].tail = A
```

```
answer[N] = {0}
flag = 0
conflict = 0
for i from 1 to N
    if answer[i] == 0
        if DFS(Adj_list[ ], i, flag, answer[ ]) == 1
            conflict = 1
if conflict == 0
    for i from 1 to N
        if answer[i] == 1
            print i
    for i from 1 to N
        if answer[i] == -1
            print i
int DFS(Adj_list[ ], i, flag, answer[ ])
    for node from Adj_list[i].head to Adj_list[i].tail
        if answer[node.id] == 0
            answer[node.id] = flag
            return DFS(Adj_list[ ], node->next->id,
            -flag, answer[ ])
        else
            if answer[node.id] != flag
                return 1
    return 0
```

The time complexity for

1. initializing is O(N).
2. connecting adj_list is O(M).
3. Traversing is O(M+N), since there are 2M nodes and at most N students who don’t hate anyone.
4. Printing is O(N).

So the total time complexity is O(M+N).

Problem 2:

1.
Every node can left rotate when it has a right child, and every node can right rotate when it has a left child. And since it is a complete binary tree,

```
if N%2 == 0
    right rotate: N/2
    left rotate: N/2-1
else
    right rotate: N/2
    left rotate: N/2
```

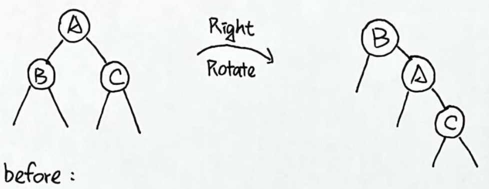
2.

```
while(node -> right != NULL)
    node = node -> right
while(node != NULL)
    while(node -> left != NULL)
        Right_Rotate(T, node)
    node = node -> parent
```

(1)We let the node be the right most node of the original tree.
(2)When it has left child, we right rotate it.

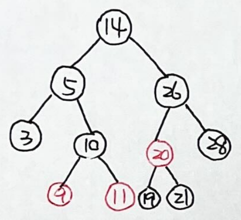
(3)When it doesn’t have left child, let the node be its parent.
By doing so, we can make the binary tee a right-going chain.
The time complexity of (1) is O(n), and we will right rotate for at most n-1 times, so the total time complexity is O(n).

3.



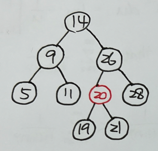
before :
Bh (B) = Bh (C)
if after right rotate , Bh(B) still equals Bh(C) , then Ⓑ and Ⓐ are both red , which violates the rules.

4.

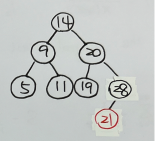


5.

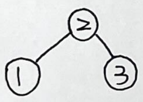
remove 3



remove 26



6.



Insert 2
Insert 1
Insert 3
Insert 4
Delete 4

if insert 1 , 2 , 3 or 1 , 3 , 2 or 2 , 1 , 3 or 2 , 3 , 1 or 3 , 1 , 2 or 3 , 2 , 1 It will be