1. (Programming: **35 points**)
   The selection sort is known to produce unstable sorting results. The attached code is a simple instance of unstable sorting results.
   First, explain the stable sorting result of the following input, and then implement '`selection_sort_stable`' that produces the stable sorting result.

   Note) It is not mandatory to follow the data structure used in the attached code. Feel free to revise it, if needed.

```
typedef struct data {
          int *id;
          int *score;
} data;


int data_id[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
int data_score[] = { 30, 25, 10, 20, 80, 30, 25, 10 };
```

```
Input data
1          30
2          25
3          10
4          20
5          80
6          30
7          25
8          10

Sorted data
3          10
8          10
4          20
7          25
2          25
6          30
1          30
5          80
```

Unstable sorting result

2. (Hand-written homework: **5 points**)
   Analyze and explain the pseudo code and its time complexity of ‘Partition’ in p36 (‘Another Solution for Partition’) of ‘DS-Lec09-Sorting’.

3. (Programming: **60 points**)
The radix sort was implemented using queue in p51 of the lecture note. Revise this code by using the counting sort in p42 instead of the queue.

Step 1) Generate the input data to be sorted randomly.
# of input data: 1000 integers (data range: 0 ~2^{24}-1, i.e., 24-bit positive integer)
Step 2) Divide each data into 4 segments, i.e., each segment consists of 6 bits.
Step 3) Sort values at each digit from LSD to MSD.

Namely, in p47 of 'DS-Lec09-Sorting.pdf', n=1000, b=24, r=6, b/r=4 (# of digits)

Note)
- Consider what the data structure is needed to implement the radix sort using the counting sort.
- It is recommended NOT to use expensive arithmetic operators (division / and remainder operator %).

```
RadixSort(A, d)
        for i=1 to d
            Counting Sort(A) on digit i
```