Lab11

Evaluation criteria

Category	Evaluation	
p11	100	
Total	100	

- *Use GCC **11** version*
- *No score will be given if the gcc version is different.*

Lab 11 Sorting

- The deadline for lab11 submission is May 24 at 11:59 PM.
- Folder name : lab11
- code name: p11_1.c, p11_2.c
- Each code will be tested by 5 different input files.
- 20 score for each input, if you don’t get the answer you get 0 score.

Lab 11-1 Merge Sort

- Implement MergeSort ADT in two different ways: **iterative** and **recursive**

- For recursive approach, you need to print whenever you merge sub-lists.

- For iterative approach, you can print the sub-list sorted at each iteration step.

Lab 11-1 Merge Sort

MergeSort make_list(int size);

- Make empty list for MergeSort

void printArray(MergeSort A, int l, int r)

- **Recursive approach : print whenever you merge sub-lists.**
- **Iterative approach : print the sub-list sorted at each iteration step.**

void Insert(MergeSort m, int a)

void RmergeSort(int* A, int* tmpA, int l, int r)

- Implement MergeSort ADT recursively.

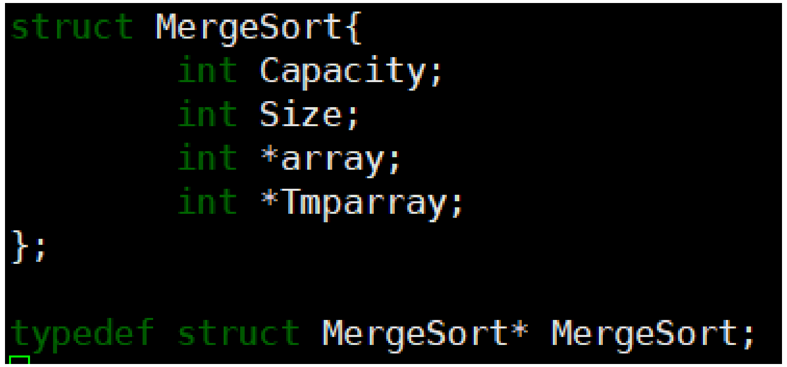
void ImergeSort(int* A, int* tmpA, int n)

- Implement MergeSort ADT iteratively.

void merge(int* A, int* tmpA, int l, int m, int r)

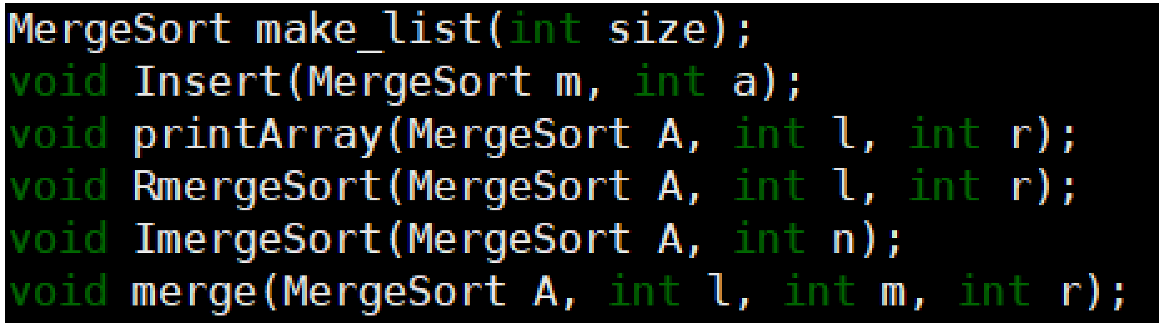
Lab 11-1 Merge Sort

- Structure



```
struct MergeSort{
        int Capacity;
        int Size;
        int *array;
        int *Tmparray;
};

typedef struct MergeSort* MergeSort;
```

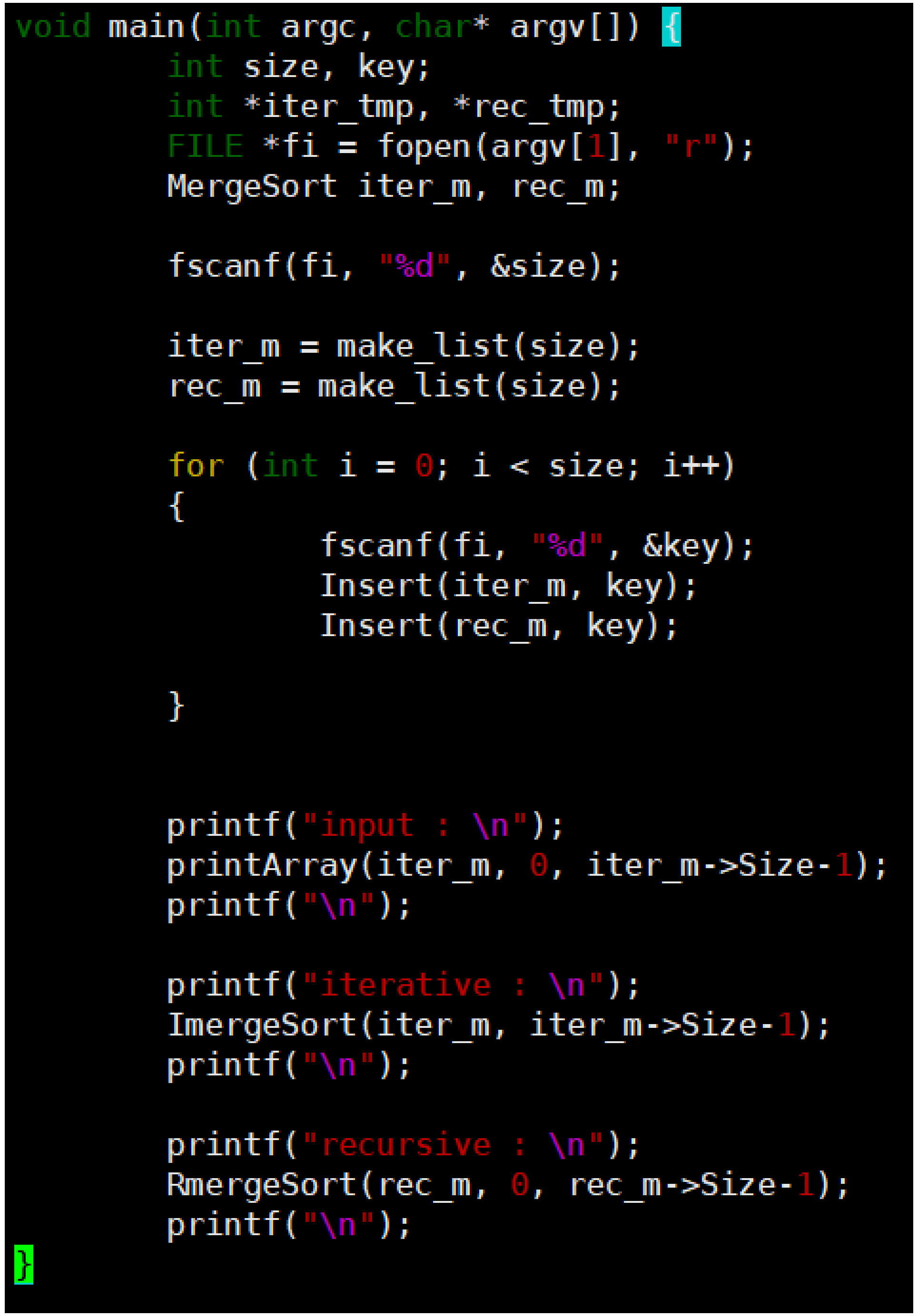
- Function



```
MergeSort make_list(int size);
void Insert(MergeSort m, int a);
void printArray(MergeSort A, int l, int r);
void RmergeSort(MergeSort A, int l, int r);
void ImergeSort(MergeSort A, int n);
void merge(MergeSort A, int l, int m, int r);
```

Lab 11-1 Merge Sort

- Main



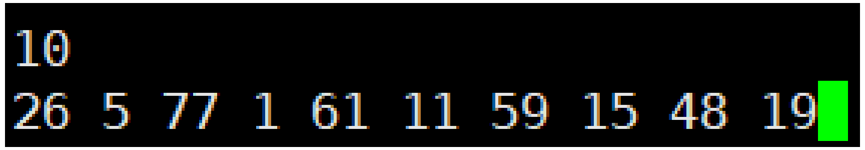
```
void main(int argc, char* argv[]) {
        int size, key;
        int *iter_tmp, *rec_tmp;
        FILE *fi = fopen(argv[1], "r");
        MergeSort iter_m, rec_m;

        fscanf(fi, "%d", &size);

        iter_m = make_list(size);
        rec_m = make_list(size);

        for (int i = 0; i < size; i++)
        {
                fscanf(fi, "%d", &key);
                Insert(iter_m, key);
                Insert(rec_m, key);

        }


        printf("input : \n");
        printArray(iter_m, 0, iter_m->Size-1);
        printf("\n");

        printf("iterative : \n");
        ImergeSort(iter_m, iter_m->Size-1);
        printf("\n");

        printf("recursive : \n");
        RmergeSort(rec_m, 0, rec_m->Size-1);
        printf("\n");
}
```

Lab 11-1 Merge Sort

- program name : p11_1.c

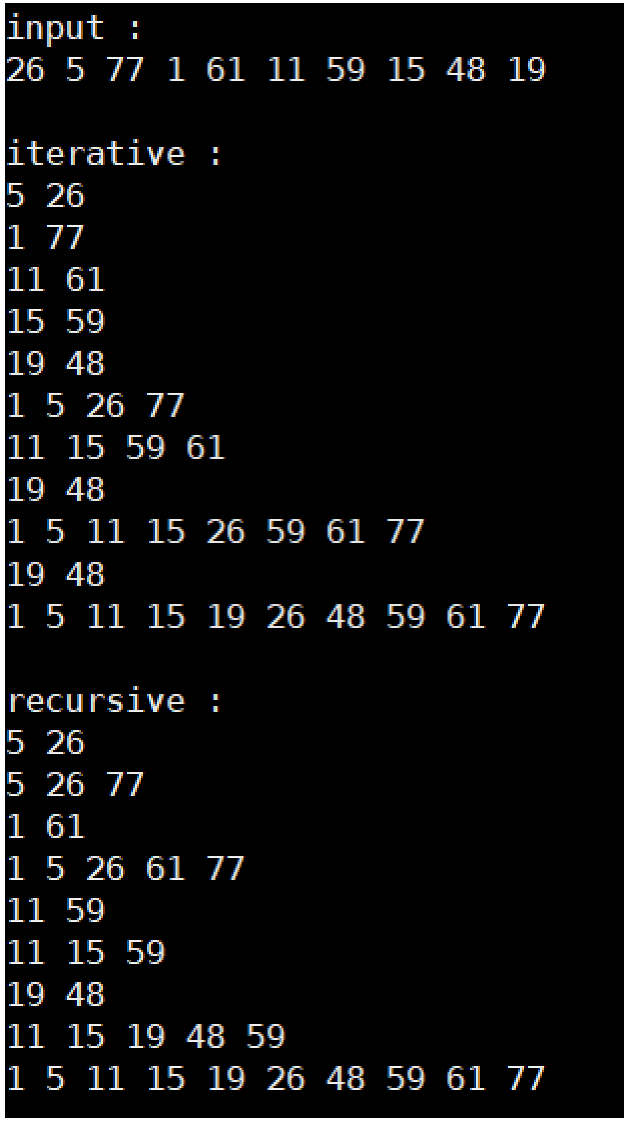
- input : max size of list ,

  the list of elements to be sorted



```
10
26 5 77 1 61 11 59 15 48 19
```

- output :



```
input :
26 5 77 1 61 11 59 15 48 19

iterative :
5 26
1 77
11 61
15 59
19 48
1 5 26 77
11 15 59 61
19 48
1 5 11 15 26 59 61 77
19 48
1 5 11 15 19 26 48 59 61 77

recursive :
5 26
5 26 77
1 61
1 5 26 61 77
11 59
11 15 59
19 48
11 15 19 48 59
1 5 11 15 19 26 48 59 61 77
```

Lab 11-2 Quick Sort

- Implement QuickSort ADT
- Our QuickSort ADT should allow three options for choosing pivot value

  : the **leftmost** element, **rightmost** element, the element in the **middle**
- Please print the list of elements at each iteration with a new pivot value

Lab 11-2 Quick Sort

QuickSort make_list(int size);

- Make empty list for QuickSort

void printArray(QuickSort q)

void Insert(QuickSort q, int a)

void swap(int* a, int* b)

int middle_partition(QuickSort q, int low, int high)

- Set the pivot to the middle element.

int leftmost_partition(QuickSort q, int left, int right)

- Set the pivot to the leftmost element.

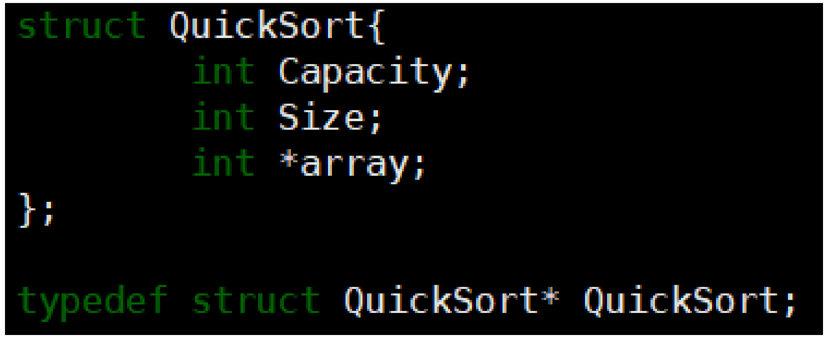
int rightmost_partition(QuickSort q, int left, int right)

- Set the pivot to the rightmost element.

void quicksort(QuickSort q, int left, int right, int type)

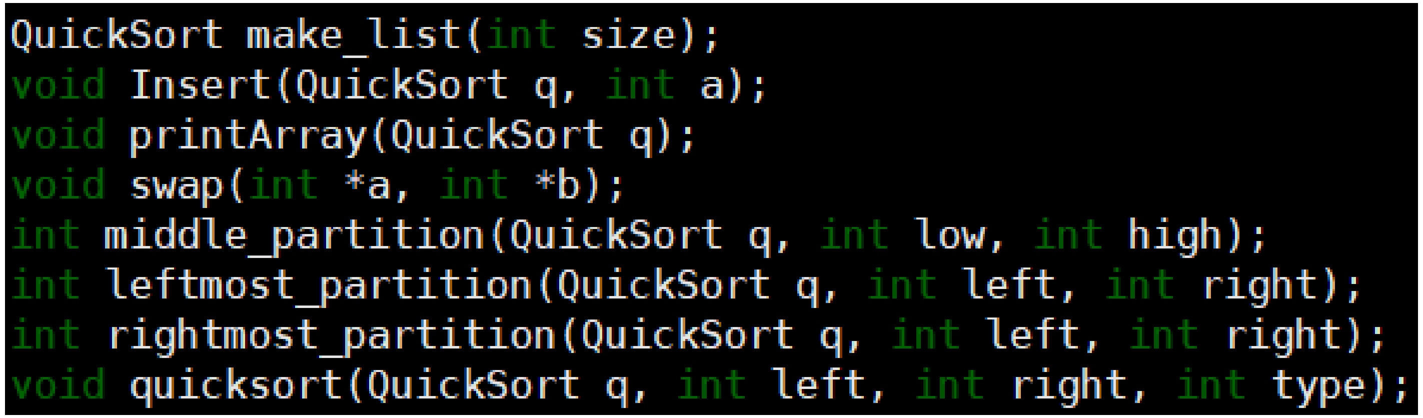
Lab 11-2 Quick Sort

- Structure



```
struct QuickSort{
        int Capacity;
        int Size;
        int *array;
};

typedef struct QuickSort* QuickSort;
```

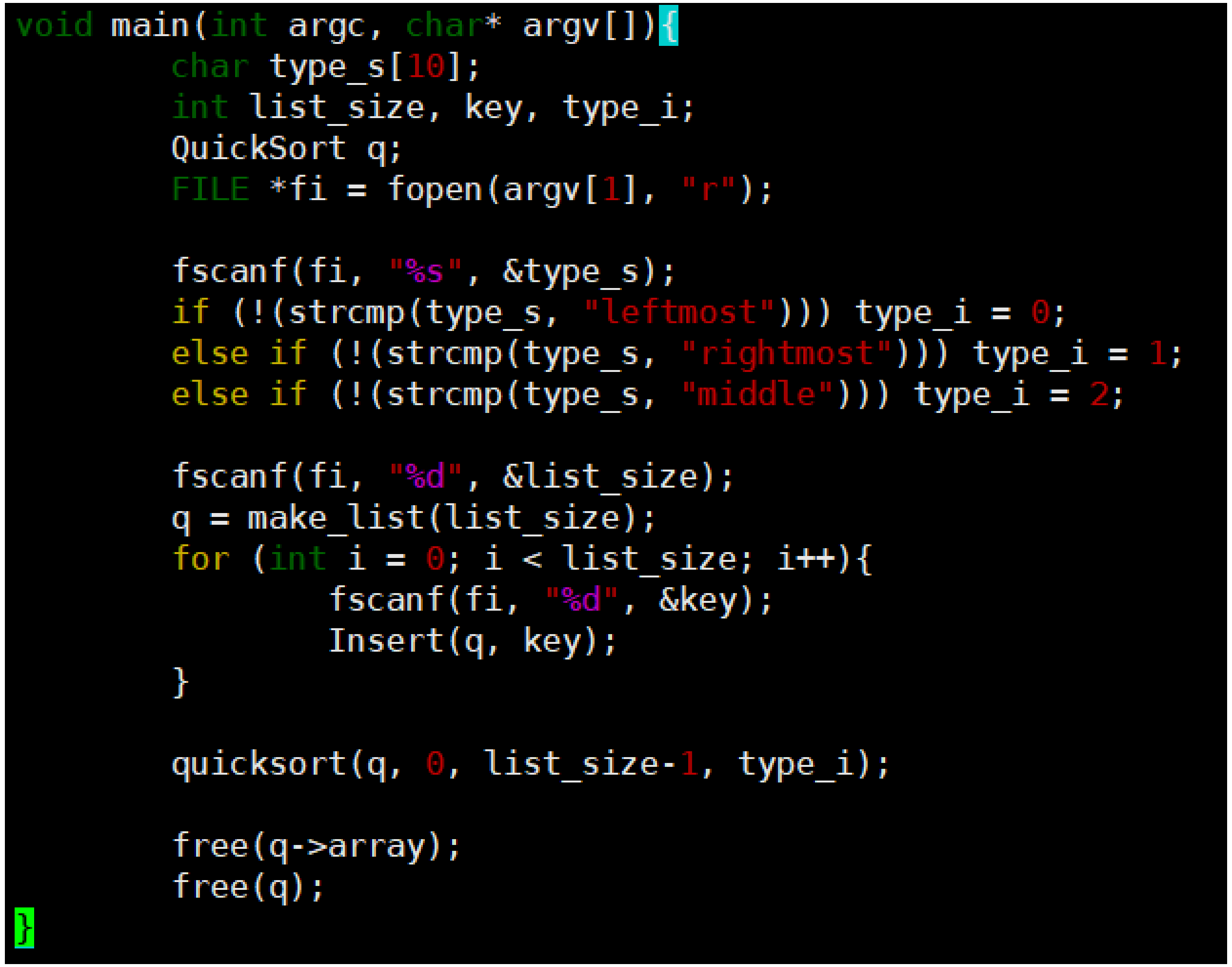
- Function



```
QuickSort make_list(int size);
void Insert(QuickSort q, int a);
void printArray(QuickSort q);
void swap(int *a, int *b);
int middle_partition(QuickSort q, int low, int high);
int leftmost_partition(QuickSort q, int left, int right);
int rightmost_partition(QuickSort q, int left, int right);
void quicksort(QuickSort q, int left, int right, int type);
```

Lab 11-2 Quick Sort

- Main



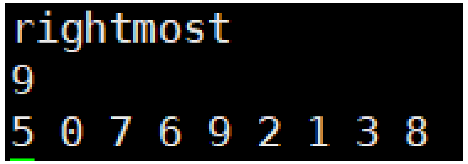
```
void main(int argc, char* argv[]){
        char type_s[10];
        int list_size, key, type_i;
        QuickSort q;
        FILE *fi = fopen(argv[1], "r");

        fscanf(fi, "%s", &type_s);
        if (!(strcmp(type_s, "leftmost"))) type_i = 0;
        else if (!(strcmp(type_s, "rightmost"))) type_i = 1;
        else if (!(strcmp(type_s, "middle"))) type_i = 2;

        fscanf(fi, "%d", &list_size);
        q = make_list(list_size);
        for (int i = 0; i < list_size; i++){
                fscanf(fi, "%d", &key);
                Insert(q, key);
        }

        quicksort(q, 0, list_size-1, type_i);

        free(q->array);
        free(q);
}
```

Lab 11-2 Quick Sort

- program name : p11_2.c
- input : option for choosing pivot value

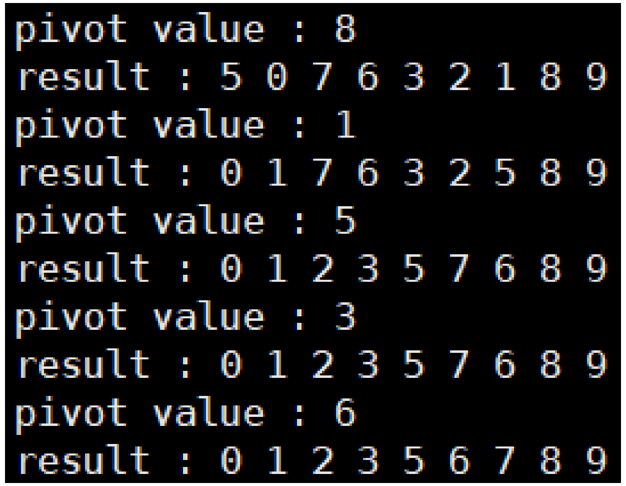
  max size of list ,

  the list of elements to be sorted



```
rightmost
9
5 0 7 6 9 2 1 3 8
```

- output : please print the list of elements whenever you pick a new pivot value.

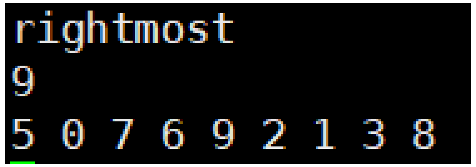


```
pivot value : 8
result : 5 0 7 6 3 2 1 8 9
pivot value : 1
result : 0 1 7 6 3 2 5 8 9
pivot value : 5
result : 0 1 2 3 5 7 6 8 9
pivot value : 3
result : 0 1 2 3 5 7 6 8 9
pivot value : 6
result : 0 1 2 3 5 6 7 8 9
```

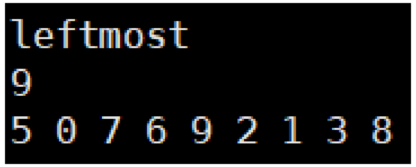
Lab 11-2 Quick Sort

- program name : p11_2.c

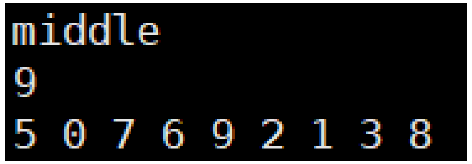
- input :



```
rightmost
9
5 0 7 6 9 2 1 3 8
```

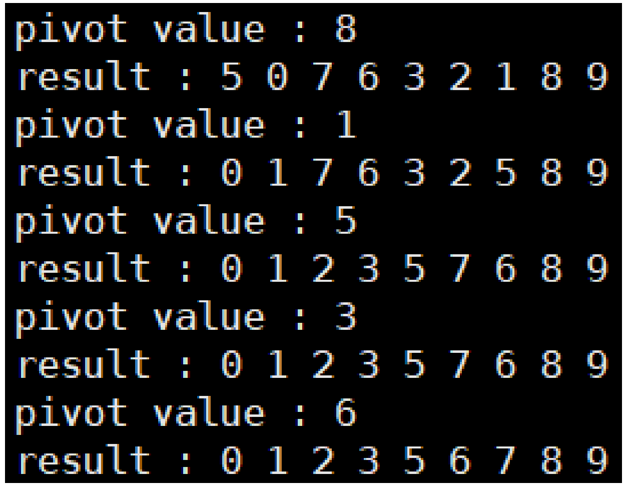


```
leftmost
9
5 0 7 6 9 2 1 3 8
```

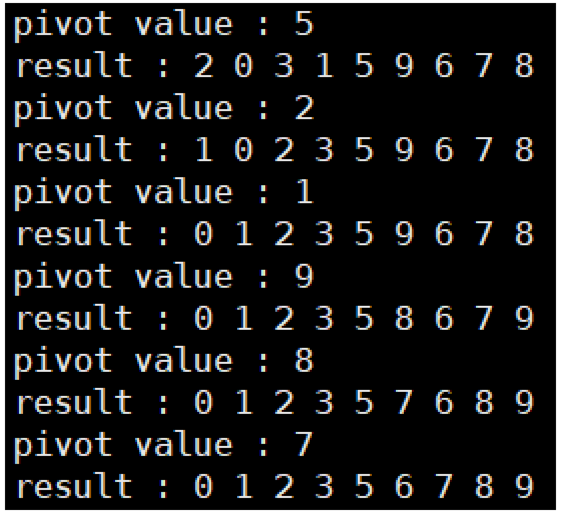


```
middle
9
5 0 7 6 9 2 1 3 8
```

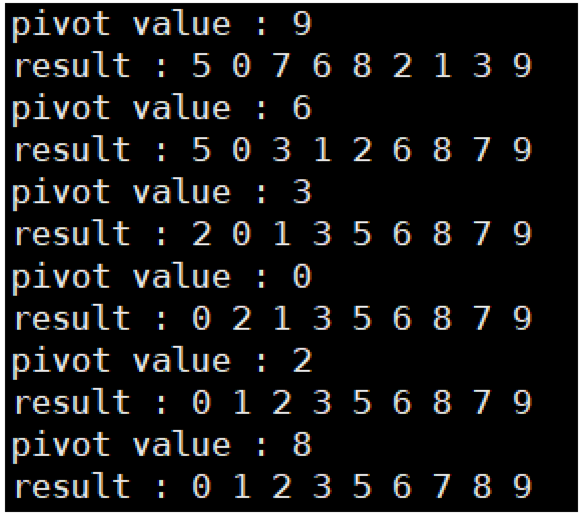
- Corresponding output:



```
pivot value : 8
result : 5 0 7 6 3 2 1 8 9
pivot value : 1
result : 0 1 7 6 3 2 5 8 9
pivot value : 5
result : 0 1 2 3 5 7 6 8 9
pivot value : 3
result : 0 1 2 3 5 7 6 8 9
pivot value : 6
result : 0 1 2 3 5 6 7 8 9
```



```
pivot value : 5
result : 2 0 3 1 5 9 6 7 8
pivot value : 2
result : 1 0 2 3 5 9 6 7 8
pivot value : 1
result : 0 1 2 3 5 9 6 7 8
pivot value : 9
result : 0 1 2 3 5 8 6 7 9
pivot value : 8
result : 0 1 2 3 5 7 6 8 9
pivot value : 7
result : 0 1 2 3 5 6 7 8 9
```



```
pivot value : 9
result : 5 0 7 6 8 2 1 3 9
pivot value : 6
result : 5 0 3 1 2 6 8 7 9
pivot value : 3
result : 2 0 1 3 5 6 8 7 9
pivot value : 0
result : 0 2 1 3 5 6 8 7 9
pivot value : 2
result : 0 1 2 3 5 6 8 7 9
pivot value : 8
result : 0 1 2 3 5 6 7 8 9
```