(합병정렬)

```
typedef struct Node {
        int data;
        Node *next;
}Node;
typedef struct List {
        Node *head;
        int count;
}List;

List *mergeSort(List *list) {
        List *L2 = NULL;
        L2 = mg_partition(list, (float)list->count/2+0.5));
        if (list->count > 1) list = mergeSort(list);
        if (L2->count > 1) L2 = mergeSort(L2);
        return merge(list, L2);
}
List *mg_partition(List *list, int k) {
        ....
        Pnode = list->head;
        L2 = (List *)malloc(sizeof(List));
        for (i=0; i<k, I++){
            if (i==k-1) {
                L2->head = Pnode->next;
                L2->count = list->count – k;
                list->count = k;
                Pnode->next = NULL;
            }
            Pnode = Pnode->next;
         }
        return L2;
}
```

```
List *merge(List *list, List *L2) {
  ....
  Lp = list->head;
  Rp = L2->head;
  TL = (List *)malloc(sizeof(List));
  ....
  while (Lp != NULL & Rp != NULL) {
            if (Lp->data > Rp->data) {
                 if (TL->head == NULL) {
                        TL->head = Rp;
                        Tp = TL->head;
                 }
                 else {
                        Tp->next = Rp;
                        Tp = Tp->next;
                 }
                 TL->count++;
                 Rp = Rp->next;
            }
            else {
                .....
  }
while (Lp != NULL) {
    ....
}
while (Rp != NULL) {
    ...
}
```

(퀵정렬)

```
int main() {

     srand(time(0));

}
int process(int A[], int p, int r) {
    ...
    if ( p < r) {
            q = partition(A, p, r);
            process(A, p, q-1);
            process(A, q+1, r);
    }
}
int partition (int A[], int p, int r) {
    ....
    pivot = pivot_choose(A, p, r);
    ....
}
int pivot_choose(int A[], int p, int r) {
    int num[3] = {0};

    for (i=0; i<3; I++) {
        num[i] = (rand() % (r-p)) + p;
    }
    SORT;
    return num[1];
}
```