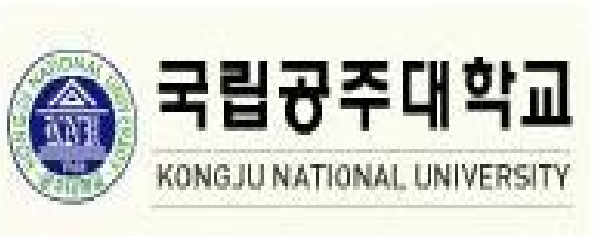
[R E P O R T]

정보통신공학전공
200301582
김성태



1. 삽입정렬

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/insertionSort/insertionSort.html

개념

삽입 정렬은 사람들이 카드 놀이를 할 때 손에 쥔 카드를 정렬하는 것과 방법이 같다. 왼손에 아무것도 쥐지 않고, 카드는 탁자 위에 뒤집힌 채 쌓여 있다고 하자. 이제 탁자에서 카드를 한 장씩 가져와 왼손의 적절한 위치에 그것을 삽입한다. 카드를 삽입할 적절한 위치를 찾기 위해서 새 카드를 이미 왼손에 들고 있는 카드와 오른쪽에서 왼쪽 방향으로 차례대로 비교해보면 된다. 왼손에 있는 카드는 항상 정렬되어 있고 그것은 탁자에서 처음에, 가장 윗부분에 쌓여 있던 카드들이다.

의사 코드

```
Insertion-Sort

for j <- 2 to length[A]
        do key <- A[j]
                ※ A[j] 를 정렬된 수열 A[1.. j – 1] 속에 삽입하기
                i <- j – 1
                while i > 0 and A[i] > key
                do A[i + 1] <- A[i]
                        i <- i – 1
                A[i + 1] <- key
```

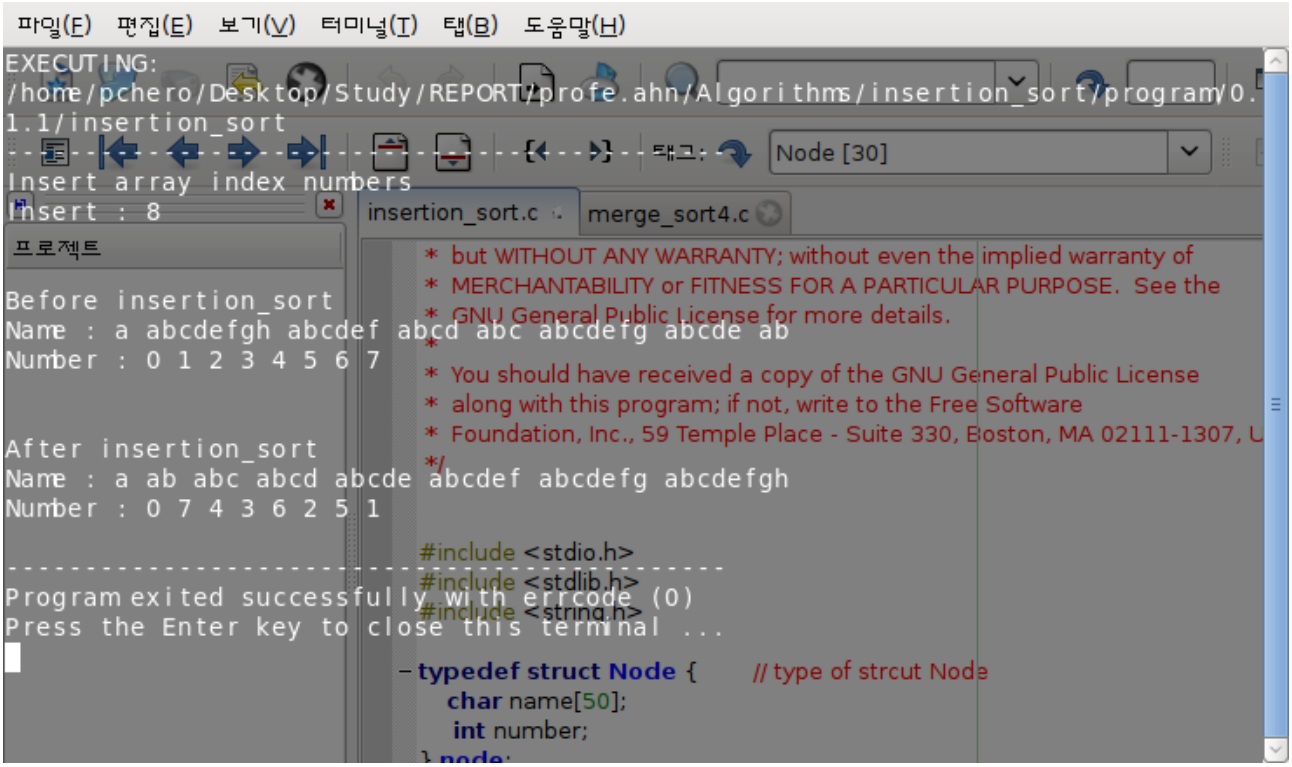
구현 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {              // type of strcut Node
    char name[50];
     int number;
} node;

/*****************************************************************************/
/*                    Functions                                              */

// sorting nodes by insertion sort algorithm.
void insertion_sort(int n, node *array);

// swap the each nodes.
void swap_node(node *a, node *b);

// make random array.
void make_RanArray(int max, node *array);

/*****************************************************************************/


int main()
{
        int array_size;                                 // number of index
        node *array;                                    // index array
        int i;                                          // tmp number

        printf("Insert array index numbers\n");
```

```
	printf("Insert : ");
	scanf("%d", &array_size);

	array = (node *)malloc(array_size * sizeof(node));

	make_RanArray(array_size, array);

	printf("\n\nBefore insertion_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');

	insertion_sort(array_size, array);

	printf("\n\nAfter insertion_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');


	return 0;
}

/* 삽입정렬 함수 */
void insertion_sort(int n, node *array)
{
	int i, j;
	node x;

	for(i = 1; i < n; i++) {
		strcpy(x.name, array[i].name);
		x.number = array[i].number;
		j = i - 1;

		while(j >= 0 && strcmp(array[j].name, x.name) > 0) {
			swap_node(&array[j + 1], &array[j]);
			j--;
		}
		swap_node(&array[j + 1], &x);
	}
}

/* swap 함수. 노드간의 내용을 바꾸어 준다. */
void swap_node(node *a, node *b)
{
	node tmp;

	strcpy(tmp.name, b->name);
	tmp.number = b->number;

	strcpy(b->name, a->name);
	b->number = a->number;

	strcpy(a->name, tmp.name);
	a->number = tmp.number;
}

/* 임의의 중복되지 않는 max 개의 배열을 만들어주는 함수 */
```

```
void make_RanArray(int max, node *array)
{
	int i, tmp_number;
	char char_tmp;

	char_tmp = 'a';

	for(i = 0; i < max; i++) {	/* max번 만큼 반복하여 */
		array[i].number = i;	/* 각각의 노드마다 0부터 max – 1 까지의 숫자를 입력한다 */
		if(i != 0) /* i 값이 0일때는 무시 */
			strcpy(array[i].name, array[i - 1].name);
		array[i].name[i] = char_tmp + i;
		array[i].name[i + 1] = '\0';
	}
	tmp_number = max;
       /* 노드들의 배열을 임의대로 섞어 준다. */
	for(i = 0; i < tmp_number; i++)
		swap_node(&array[rand() % (max - 1)], &array[--tmp_number]);

	for(i = 0; i < max; i++)
		array[i].number = i;
}
```

실행 화면



2. 선택정렬

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/selectionSort/selectionSort.html

개념

정렬되지 않은 데이터 중에서 가장 작은 혹은 가장 큰 데이터를 선택해서 정렬된 데이터에 삽입하는 방법.
김성태, 김윤겸, 방영배,....등등의 명함이 있을 때, 이를 하나씩 집어서 가, 나, 다 순으로 정렬하는 방법.

의사 코드

```
void selectionsort(int n, keytype S[])
{
        index I, j, smallest;

        for(i = 1; I <= n – 1; i++) {
                smallest = i;
                for(j = i + 1; j <= n; j++)
                        if(S[j] < S[smallest])
                                smallest = j;
                        exchage S[i] and S[smallest];
}
```

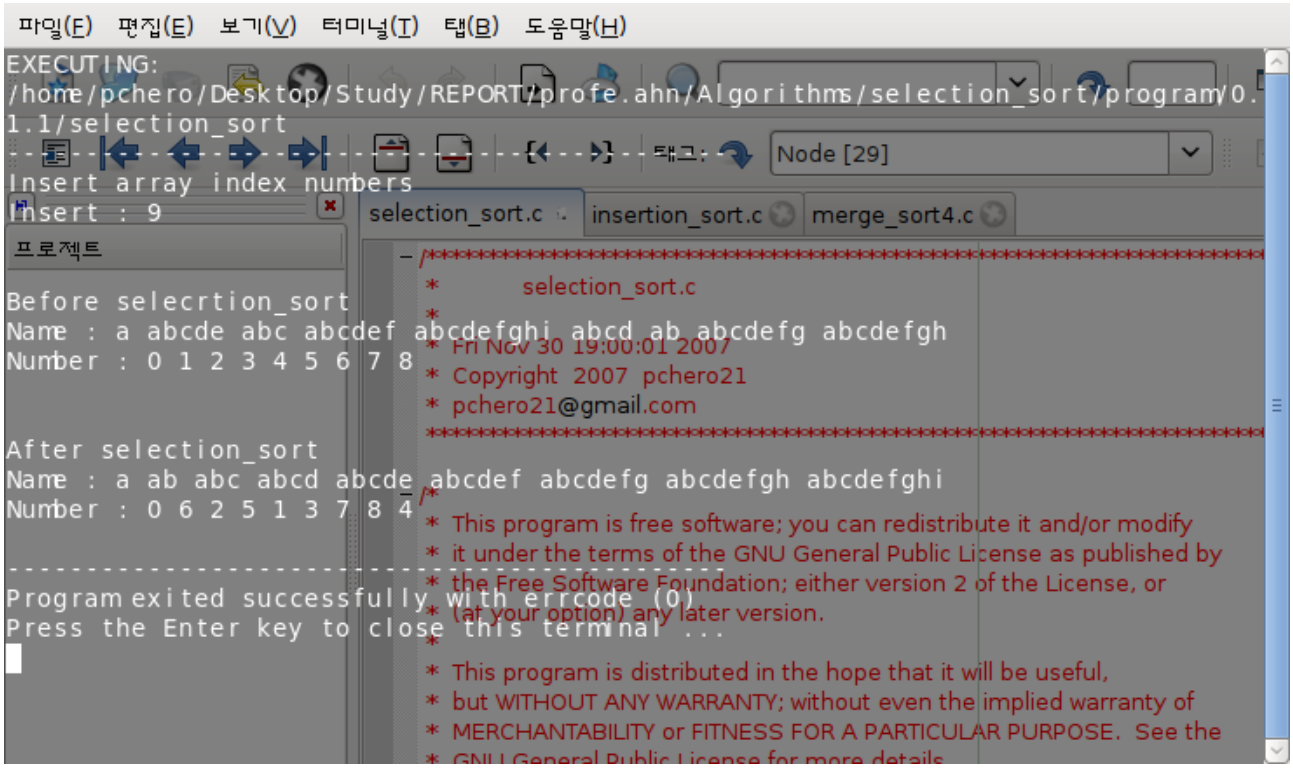
구현 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {           // type of strcut Node
   char name[50];
    int number;
} node;

/*****************************************************************************/
/*                                  Functions                                */

// sorting nodes by selecrtion sort algorithm.
void selection_sort(int n, node *array);

// swap the each nodes.
void swap_node(node *a, node *b);

// make random array.
void make_RanArray(int max, node *array);

/*****************************************************************************/

int main()
{
        int array_size;                                 // number of index
        node *array;                                    // index array
        int i;                                                  // tmp number

        printf("Insert array index numbers\n");
        printf("Insert : ");
        scanf("%d", &array_size);

        array = (node *)malloc(array_size * sizeof(node));
```

```
	make_RanArray(array_size, array);

	printf("\n\nBefore selecrtion_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');

	selection_sort(array_size, array);

	printf("\n\nAfter selection_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');

	return 0;
}

/* 선택 정렬 */
void selection_sort(int n, node *array)
{
	int i, j, smallest;

	for(i = 0; i < n - 1; i++) {
		smallest = i;	/* smallest 값을 얻는다. */

		for(j = i + 1; j < n; j++)
			if(strcmp(array[j].name, array[smallest].name) < 0)
				smallest = j;	/* 다음의 노드와 비교하여 smallest값을 변경 */
			swap_node(&array[i], &array[smallest]); /* 서로간의 노드를 교환 */
	}
}

/* swap 함수. 서로간의 노드들의 값을 바꿔준다. */
void swap_node(node *a, node *b)
{
	node tmp;

	strcpy(tmp.name, b->name);
	tmp.number = b->number;

	strcpy(b->name, a->name);
	b->number = a->number;

	strcpy(a->name, tmp.name);
	a->number = tmp.number;
}

/* 중복되지 않는 n개의 배열을 만드는 함수 */
void make_RanArray(int max, node *array)
{
	int i, tmp_number;
	char char_tmp;

	char_tmp = 'a';

	for(i = 0; i < max; i++) {
		array[i].number = i;
```

```
                if(i != 0)
                        strcpy(array[i].name, array[i - 1].name);
                array[i].name[i] = char_tmp + i;
                array[i].name[i + 1] = '\0';
        }
        tmp_number = max;
        for(i = 0; i < tmp_number; i++)
                swap_node(&array[rand() % (max - 1)], &array[--tmp_number]);

        for(i = 0; i < max; i++)
                array[i].number = i;
}
```

실행 화면



3. 합병정렬 – 1

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html

개념

기존의 합병정렬을 동적 계획법을 이용하여 구현한다.
기존의 합병정렬 수행시 단독아이템(singleton) 배열이 될 때까지 배열을 분할하지만, 동적 계획법으로 구성할 경우, 이와 같은 분할을 할 필요가 없다.

의사 코드

```
void mergesort(int n, keytype S[])
{
        int m;
        index low, mid, high, size;
        m = 2^(lg n);
        size = 1;
        repeat(lg m times) {
                for(low = 1; low <= m – 2 * size + 1; low = low + 2 *size) {
                        mid = low + size – 1;
                        high(minimum(low + 2 * size – 1, n);
                        merge3(low, mid, high, S);
                }
                size = 2 * size;
        }
}
```

구현 소스

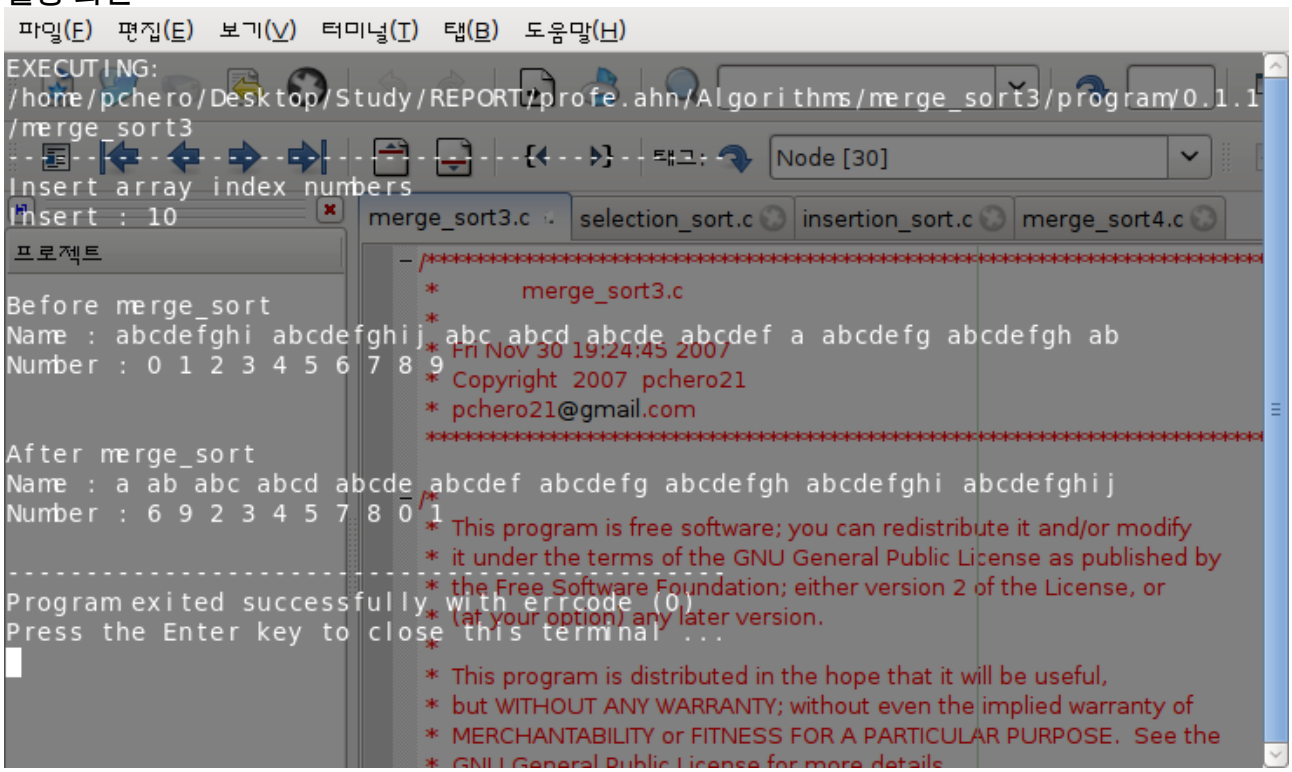
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

typedef struct Node {           // type of strcut Node
   char name[50];
    int number;
} node;

/*****************************************************************************/
/*                                    Functions                                */

// sorting nodes by merge sort algorithm.
void merge_sort(int n, node *array);

// swap the each nodes.
void swap_node(node *a, node *b);

// make random array.
void make_RanArray(int max, node *array);

// return mininum number
int minimum(int a, int b);

// merge array
void merge(int low, int mid, int high, node *array);

// function for log2
double logX(double x, double base);
/*****************************************************************************/
```

```
int main()
{
	int array_size;					// number of index
	node *array;					// index array
	int i;							// tmp number

	printf("Insert array index numbers\n");
	printf("Insert : ");
	scanf("%d", &array_size);

	array = (node *)malloc(array_size * sizeof(node));

	make_RanArray(array_size, array);

	printf("\n\nBefore merge_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');

	merge_sort(array_size, array);

	printf("\n\nAfter merge_sort\n");
	printf("Name : ");
	for(i = 0; i < array_size; i++)
		printf("%s ", array[i].name);
	putchar('\n');
	printf("Number : ");
	for(i = 0; i < array_size; i++)
		printf("%d ", array[i].number);
	putchar('\n');

	return 0;
}

/* 개량된 합병정렬 */
void merge_sort(int n, node *array)
{
	int m;
	int low, mid, high, size;
	int i, tmp;

	m = pow(2, (int)ceil(logX(n, 2.0))); /* 배열의 크기를 2의 거듭제곱으로 취함 */
	size = 1;

	for(i = 0; i < (int)ceil(logX(m, 2.0)); i++) {
		for(low = 0; low <= m - (2 * size) + 1; low = low + (2 * size)) {
			mid = low + size - 1;
			high = minimum(low + 2 * size - 1, n - 1);
			merge(low, mid, high, array);
		}
		size = 2 * size;
	}
}

/* 개량된 합병 알고리즘 */
void merge(int low, int mid, int high, node *array)
{
	int i , j, k;
	int tmp, count_tmp;
	node array_tmpLow[mid - low + 1], array_tmpHigh[high - mid + 1];

	if(low >= high || mid >= high)
		return;
```

```
	count_tmp = 0;
	for(tmp = low; tmp <= mid; tmp++) {
		strcpy(array_tmpLow[count_tmp].name, array[tmp].name);
		array_tmpLow[count_tmp].number = array[tmp].number;
		count_tmp++;
	}

	count_tmp = 0;
	for(tmp = mid + 1; tmp <= high; tmp++) {
		strcpy(array_tmpHigh[count_tmp].name, array[tmp].name);
		array_tmpHigh[count_tmp].number = array[tmp].number;
		count_tmp++;
	}

	j = k = low;
	i = mid + 1;

	while(j <= mid && i <= high) {
		if(strcmp(array_tmpLow[j - low].name, array_tmpHigh[i - mid - 1].name) < 0) {
			swap_node(&array[k], &array_tmpLow[j - low]);
			j++;
		} else {
			swap_node(&array[k], &array_tmpHigh[i - mid - 1]);
			i++;
		}
		k++;
	}
	if(i > high)
		for(; j <= mid; j++)
			swap_node(&array[k++], &array_tmpLow[j - low]);
	else
		for(; i <= high; i++)
			swap_node(&array[k++], &array_tmpHigh[i - mid - 1]);
}

/* 노드들을 서로 바꿔주는 swap */
void swap_node(node *a, node *b)
{
	node tmp;

	strcpy(tmp.name, b->name);
	tmp.number = b->number;

	strcpy(b->name, a->name);
	b->number = a->number;

	strcpy(a->name, tmp.name);
	a->number = tmp.number;
}

void make_RanArray(int max, node *array)
{
	int i, tmp_number;
	char char_tmp;

	char_tmp = 'a';

	for(i = 0; i < max; i++) {
		array[i].number = i;

		if(i != 0)
			strcpy(array[i].name, array[i - 1].name);

		array[i].name[i] = char_tmp + i;
		array[i].name[i + 1] = '\0';
	}
	tmp_number = max;
```

```
	for(i = 0; i < tmp_number; i++)
		swap_node(&array[rand() % (max - 1)], &array[--tmp_number]);

	for(i = 0; i < max; i++)
		array[i].number = i;
}

int minimum(int a, int b)
{
	return (a > b) ? b : a;
}


double logX(double x, double base) {
  return log(x) / log(base);
}
```

실행 화면



4. 합병정렬 – 2

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html

개념

기존의 합병정렬을 개량하는 두번째 알고리즘이다.
정렬 문제는 보통은 키값을 가지고 레코드를 결정한다. 만약 레코드가 크다면 합병정렬이 사용하는 추가적인 저장장소 사용량은 상당할 수 있다. 각 레코드에 링크(link) 필드를 추가하여 추가적인 저장장소를 절약할 수 있다. 그러면 레코드를 옮기는 대신 링크를 조정하여 레코드를 정렬된 연결된 리스트로 정렬한다. 이는 추가적인 레코드 배열을 만들 필요가 없다는 의미이다. 링크로 쓰여진 저장장소는 큰 레코드에 비해서 훨씬 작으므로, 절약되는 저장장소의 크기는 꽤 된다. 더구나, 링크를 조정하는 데 드는 시간은 큰 레코드를 옮기는 데 필요한 시간보다 작으므로, 시간절약도 된다.

의사 코드

```
struct node
{
        keytype key;
        index link;
};

void mergesort4(index low, index high, index& megedlist)
{
        index mid, list1, list2;
        if(low == high) {
                mergedlist = low;
                S[mergedlist] = low;
        } else {
                mid = (low + high) / 2;
                mergesort4(low, mid, list1);
                mergesort4(mid + 1; high; list2);
                merge4(list1, list2, mergedlist);
        }
}

void merge4(index list1, index list2, index& mergedlist)
{
        index lastsorted;
        if(S[list1].key < S[list2].key) {
                mergedlist = list1;
                list1 = S[list1].link;
        } else {
                mergedlist = list2;
                list2 = S[list2].link;
        }
        lastsorted = mergedlist;
        while(list1 != 0 && list2 != 0)
                if(S[list1].key < S[list2].key) {
                        S[lastsorted].link = list1;
                        lastsorted = list1;
                        list1 = S[list1].link;
                } else {
                        S[lastsorted].link = list2;
                        lastsorted = list2;
                        list2 = S[list2].link;
                }
        if(list1 == 0)
                S[lastsorted].link = list2;
        else
                S[lastsorted].link = list1;
```

```
}
```

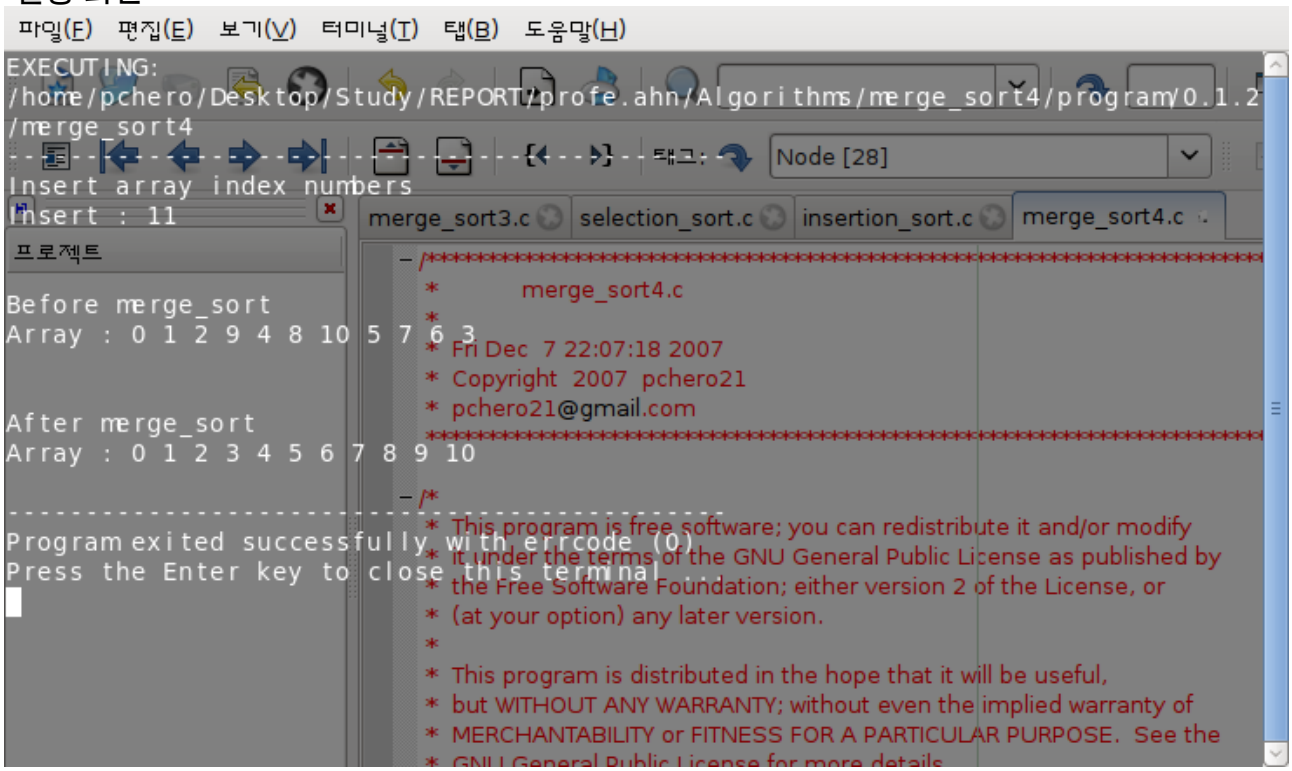
구현 소스

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {              // type of strcut Node
        int  number;
        struct Node *link;
} node;

/*****************************************************************************/
/*                                                                    Functions

                                          */

/* preform merge sort on the linked list */
node *mergesort(node *head);

/* merge the lists.. */
node *merge(node *head_one, node *head_two);

/* make random array */
void make_RanArray(int max, node *array);

/* swap the each nodes */
void swap_node(node *a, node *b);

/*****************************************************************************/

int main(void)
{
        node *array;
        node *head;
        node *current;
        node *next;
        int array_size;
        int i;

        head = NULL;

        printf("Insert array index numbers\n");
        printf("Insert : ");
        scanf("%d", &array_size);

        array = (node *)malloc(array_size * sizeof(node));

        make_RanArray(array_size, array);

        printf("\n\nBefore merge_sort\n");
        printf("Array : ");
        for(current = array, i = 0; current != NULL; current = current->link)
                printf("%d ", current->number);
        putchar('\n');


        /* sort the list */
        array = mergesort(array);

        /* print the list */
        printf("\n\nAfter merge_sort\n");
        printf("Array : ");
        for(current = array, i = 0; current != NULL; current = current->link)
                printf("%d ", current->number);
        putchar('\n');

        /* done... */
```

```
	return 0;
}

/* preform merge sort on the linked list */
/* 하나의 전체적인 노드를 분할한다. */
node *mergesort(node *head)
{
	node *head_one;
	node *head_two;

	if((head == NULL) || (head->link == NULL))
		return head;

	head_one = head;
	head_two = head->link;

	while((head_two != NULL) && (head_two->link != NULL)) {
		head = head->link;
		head_two = head->link->link;
	}

	head_two = head->link;
	head->link = NULL;

	return merge(mergesort(head_one), mergesort(head_two));
}

/* merge the lists.. */
/* 나눠진 노드들의 크기를 비교하고 정렬한다. */
node *merge(node *head_one, node *head_two)
{
	node *head_three;

	if(head_one == NULL)
		return head_two;

	if(head_two == NULL)
		return head_one;

	if(head_one->number < head_two->number) {
		head_three = head_one;
	head_three->link = merge(head_one->link, head_two);
	} else {
		head_three = head_two;
		head_three->link = merge(head_one, head_two->link);
	}

	return head_three;
}

/* 노드들의 내용을 서로 바꾸어주는 함수 */
void swap_node(node *a, node *b)
{
	node tmp;

	tmp.link = b->link;
	tmp.number = b->number;

	b->link = a->link;
	b->number = a->number;

	a->link = tmp.link;
	a->number = tmp.number;
}

/* 중복되지 않는 max개의 랜덤한 노드를 만드는 함수 */
void make_RanArray(int max, node *array)
```

```
{
        int i, tmp_number;

        for(i = 0; i < max; i++)
                array[i].number = i;
        tmp_number = max;
        for(i = 0; i < tmp_number; i++)
                swap_node(&array[rand() % (max - 1)], &array[--tmp_number]);

        for(i = 0; i < max - 1; i++)
                array[i].link = &array[i + 1];
        array[max].link = 0;
}
```

실행 화면



5. 빠른정렬

```
/*****************************************************************************/
 * 죄송합니다.. 교재 7장에 나오는 향상된 빠른 정렬 알고리즘은 구현하지 못했습니다.        *
 * 지금 나오는 소스는 예전에 구현한 빠른정렬 구현 소스입니다.                      *
 * 책에서 설명하는 알고리즘을 이해하지 못했습니다..;;                              *
/*****************************************************************************/
```

알고리즘 설명

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/quickSort/quickSort.html

개념

빠른 정렬은 원소 수가 n 개인 입력 배열을 최악의 경우 세타(n^2) 에 정렬하는 알고리즘이다. 이렇게 최악의 수행시간은 많이 들지만, 실제 문제에서는 퀵 정렬이 가장 좋은 경우가 많다. 평균 수행시간이 세타(n lg n)으로 매우 효율적이고, 세타(n lg n) 표현에 숨겨진 상수 인자도 매우 작기 때문이다. 게다가 퀵 정렬은 내부 정렬이고, 가상 메모리 환경에서도 잘 작동한다.

의사 코드

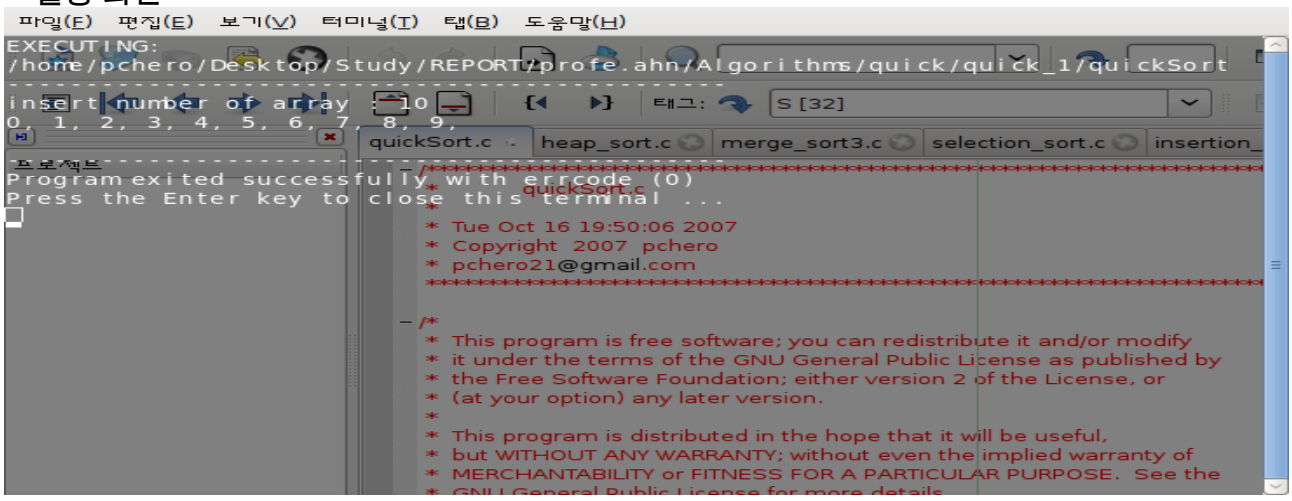
```
Quicksort(A, p, r)
if p < r
        then q <- Partition(A, p, r)
                Quicksort(A, p, q – 1)
                Quicksort(A, q +1, r)

Partition(A, p, r)
x <-A[r]
i <- p – 1
for j <- p to r – 1
        do if A[j] <= x
                then i <- i + 1
                        A[i] <-> A[j]              ※ 교환
A[i + 1] <-> A[r]                                  ※ 교환
return i + 1
```

구현 소스

```
#include <stdio.h>
#include <stdlib.h>

void quickSort(int low, int high);
void partition(int low, int high, int *pivotpoint);
void swap(int *a, int *b);

int *S;

int main(int argc, int **argv)
{

        int i, j;
        int arr_size;

        printf("insert number of array : ");
        scanf("%d", &arr_size);

        S = (int*)malloc(sizeof(int) * arr_size);

        // 난수 발생.
        j = arr_size;
        for(i = 0; i < arr_size; i++)
```

```
		S[i] = i;
	for(i = 0; i < (arr_size -1); i++)
		swap(&S[rand() % j], &S[--j]);

	quickSort(0, (arr_size - 1));

	for(i = 0; i < arr_size; i++)
		printf("%d, ", S[i]);
	printf("\n");

	free(S);
	return 0;
}

void quickSort(int low, int high)
{
	int pivotpoint;

	if(high > low) {
		partition(low, high, &pivotpoint);
		quickSort(low, pivotpoint - 1);
		quickSort(pivotpoint + 1, high);
	}
}

void partition(int low, int high, int *pivotpoint)
{
	int i, j;
	int pivotitem;

	pivotitem = S[low];
	j = low;
	for(i = low + 1; i <= high; i++) {
		if(S[i] < pivotitem) {
			j++;
			swap(&S[i], &S[j]);
		}
	}
	*pivotpoint = j;
	swap(&S[low], &S[*pivotpoint]);
}

void swap(int *a, int *b)
{
	int temp;
	temp = *a;
	*a = *b;
	*b = temp;
}
```

실행 화면



6. 힙정렬

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/heapSort/heapSort.html

개념

힙
(이진) 힙 자료구조는 완전 이진 트리로 볼 수 있는 배열 객체이다. 트리의 각 노드는 배열에서 그 노드의 값을 저장하는 원소와 대응한다. 이 트리의 가장 낮은 으을 빼고는 완전히 차 있고, 가장 낮은 층은 왼쪽부터 채운다. 힙을 나타내는 배열 A는 두 가지 인자를 갖는다. 배열 A에 있는 원소의 개수를 나타내는 length[A]와 배열 A의 원소 중 힙에 속하는 원소의 개수를 나타내는 heap-size[A]이다. 그리고 heap-size[A] <= length[A]이다. 다시 말해서 A[1...length[A]]에 있는 값들이 다 유용할 수는 있지만, A[heap-size[A]] 뒤에 있는 것들은 힙의 원소가 아니다. 트리의 루트는 A[1]이다. 그리고 노드의 인덱스 i 가 주어지면 부모 Parent(i), 왼쪽 자식 Left(i), 오른쪽 자식 Right(i)는 다음과 같이 간단히 구할 수 있다.

```
Parent(i)
    return i / 2
Left(i)
    return 2i
Right(i)
    return 2i + 1
```

힙 정렬 알고리즘
힙 정렬 알고리즘은 입력 배열 A[1...n] (n = length[A])를 최대 힙으로 만들면서 시작한다. 최대 원소가 루트 A[1] 에 저장되어 있으므로 이것을 A[n]과 교환하면 정확히 마지막 자리에 넣을 수 있다. 이제 힙에서 노드 n을 '제거'하면 A[1...(n – 1)]을 최대 힙으로 만드는 것은 그리 어렵지 않다. 루트의 자식들은 최대힙으로 남아 있지만 새로운 루트는 최대 힙의 특성을 어길 수 있다. 따라서 최대 힙의 특성을 다시 지키도록하는 Max-Heapify(A, 1) 호출한다. 힙 정렬 알고리즘은 이 과정을 힙 크기가 n – 1 일 때부터 2로 줄어들 때까지 반복한다.

의사 코드

```
Max-Heapify(A, I)
l <- Left(i)
r <- Right(i)
if l <= heap-size[A] and A[l] > A[i]
        then largest <- l
        else largest <- i
if r <= heap-size[A] and A[r] > A[largest]
        then largest <- r
if largest != I
        then A[i] <-> A[largest]                ※ 교환
             Max-Heapify(A, largest)

Build-Max-Heap(A)
heap-size[A] <- length[A]
for i <- length[A] / 2 downto 1
        do Max-Heapify(A, I)

Heapsort(A)
Build-Max-Heap(A)
for i <- length[A] downto 2
        do A[1] <-> A[i]                        ※ 교환
           heap-size[A] <- heap-size[A] – 1
           Max-Heapify(A, 1)
```

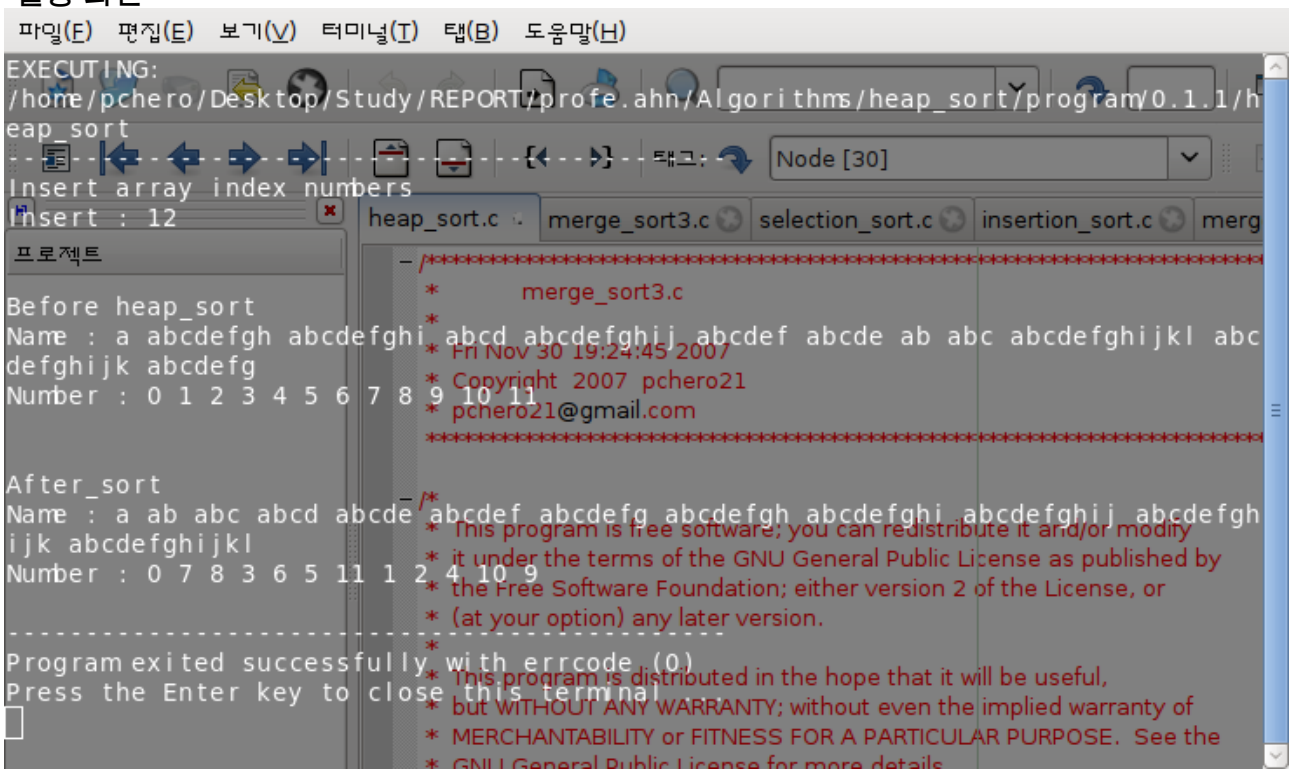
구현 소스

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {              // type of strcut Node
    char name[50];
     int number;
} node;

/*****************************************************************************/
/*                                    Functions                              */

/* preform the heapsort */
void heapsort(node *array, int array_size);

/* siftdown */
void siftdown(int pos, node *array, int array_size);

/* swap the each nodes */
void swap_node(node *a, node *b);

/* make random array */
void make_RanArray(int max, node *array);

/* make heap array */
void make_heap(int array_size, node *array);

/*****************************************************************************/

int main(void)
{

        int i = 0;
        int array_size;                                 // number of index
        node *array;                                    // index array

        printf("Insert array index numbers\n");
        printf("Insert : ");
        scanf("%d", &array_size);

        array = (node *)malloc(array_size * sizeof(node));

        make_RanArray(array_size, array);

        /* print the original array */
        printf("\n\nBefore heap_sort\n");
        printf("Name : ");
        for(i = 0; i < array_size; i++)
                printf("%s ", array[i].name);
        putchar('\n');
        printf("Number : ");
        for(i = 0; i < array_size; i++)
                printf("%d ", array[i].number);
        putchar('\n');

        heapsort(array, array_size);

        /* print the `heapsorted' array */
        /* print the original array */
        printf("\n\nAfter_sort\n");
        printf("Name : ");
        for(i = 0; i < array_size; i++)
                printf("%s ", array[i].name);
        putchar('\n');
        printf("Number : ");
        for(i = 0; i < array_size; i++)
                printf("%d ", array[i].number);
        putchar('\n');
```

```
	return 0;
}

/* 전체적인 정렬부분. */
/* 힙 정렬 이용시, 이 함수만 호출하면 된다. */
void heapsort(node *array, int array_size)
{
	int i = 0;
	node tmp;

	make_heap(array_size, array);

	for(i = array_size - 1; i > 0; i--) {
		tmp = array[0];
		array[0] = array[i];
		array[i] = tmp;
		siftdown(0, array, i);
	}
}

/* 실질적인 정렬 부분 */
/* 노드의 크리를 비교하여 힙 정렬 알고리즘을 이용하여 정렬한다 */
void siftdown(int pos, node *array, int array_size)
{
	int parent = 0;
	int largechild = 0;
	int tmp = 0;
	int left = 0;
	int right = 0;

	 parent = pos;

	for(;;) {
		left = 2 * parent + 1;
		right = left + 1;

		if(left >= array_size)
			return;
		else if(right >= array_size)
			largechild = left;
		else if(strcmp(array[left].name, array[right].name) > 0)
			largechild = left;
		else
			largechild = right;

		if(strcmp(array[parent].name, array[largechild].name) >0)
			return;

		swap_node(&array[parent], &array[largechild]);
		parent = largechild;
	}
}

/* swap 함수 */
void swap_node(node *a, node *b)
{
	node tmp;

	strcpy(tmp.name, b->name);
	tmp.number = b->number;

	strcpy(b->name, a->name);
	b->number = a->number;

	strcpy(a->name, tmp.name);
	a->number = tmp.number;
}
```

```
/* 중복되지 않는 Max 개의 랜덤 배열 만들기 */
void make_RanArray(int max, node *array)
{
	int i, tmp_number;
	char char_tmp;

	char_tmp = 'a';

	for(i = 0; i < max; i++) {
		array[i].number = i;

		if(i != 0)
			strcpy(array[i].name, array[i - 1].name);

		array[i].name[i] = char_tmp + i;
		array[i].name[i + 1] = '\0';
	}
	tmp_number = max;
	for(i = 0; i < tmp_number; i++)
		swap_node(&array[rand() % (max - 1)], &array[--tmp_number]);

	for(i = 0; i < max; i++)
		array[i].number = i;
}

/* 힙 자료구조를 만든다. */
void make_heap(int array_size, node *array)
{
	int i;

	for(i = array_size / 2; i >= 0; i--)
		siftdown(i, array, array_size);
}
```

실행 화면



7. 기수정렬

알고리즘 설명
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/radixSort/radixSort.html

개념

기수 정렬(radix sort)은 컴퓨터 박물관에서나 볼 수 있는 카드 정렬 기계에서 사용하는 알고리즘이다. 카드는 80열로 구성되고, 각 열에는 자리 12개가 있어 그 중 한 군데에 구멍을 찍을 수 있다. 정렬기는 기계적으로 "프로그램되어 있어서" 각 카드에서 열을 살펴 보고, 어떤 자리에 구멍이 뚫렸는지에 따라서 12개의 통 중 하나로 카드를 분배한다. 그리고 나서 작업자는 첫째 구멍이 뚫린 카드들이 둘째 구멍이 뚫린 카드들 위에 오고 하는 식으로 카드를 통별로 모을 수 있다.

기수 정렬(radix sort)은 숫자(digit)별로 나누어서 정렬을 하는데, 최소 유효 숫자(least significant)에서 최대 유효 숫자(most significant)로 한 번에 한 숫자씩 정렬한다. 예를 들어, 기수가 10인 숫자들{15, 12, 49, 16, 36, 40}을 기수 정렬을 사용해서 정렬할 때, 최소 유효 숫자(1의 자리)에 대해 정렬하면 {40, 12, 15, 16, 36, 49}가 되고, 최대 유효 숫자(10의 자리)에 대해 정렬하면 {12, 15, 16, 36, 40, 49}가 된다.

각 위치의 숫자값들을 정렬할 때 기수 정렬이 안정적이라는 점은 매우 중요하다. 왜냐하면 일단 적은 유효 위치의 숫자값에 따라 정렬되면, 그 정렬된 위치는 최대 유효 숫자값에 따른 정렬에서 바꾸지 않아도 된다면 바뀌지 않기 때문이다. 안정성은 제쳐 두고, 선형 시간에 실행되고 어떤 기수에 대해서도 가장 큰 정수를 알 수 있으므로 기수 정렬은 카운팅 정렬을 사용한다.

의사 코드

```
radixsort(A, d)
for l <- 1 to d
        do i번째 자리에 대한 안정된 정렬을 사용하여 배열 A를 정렬한다.(카운팅 정렬)
```

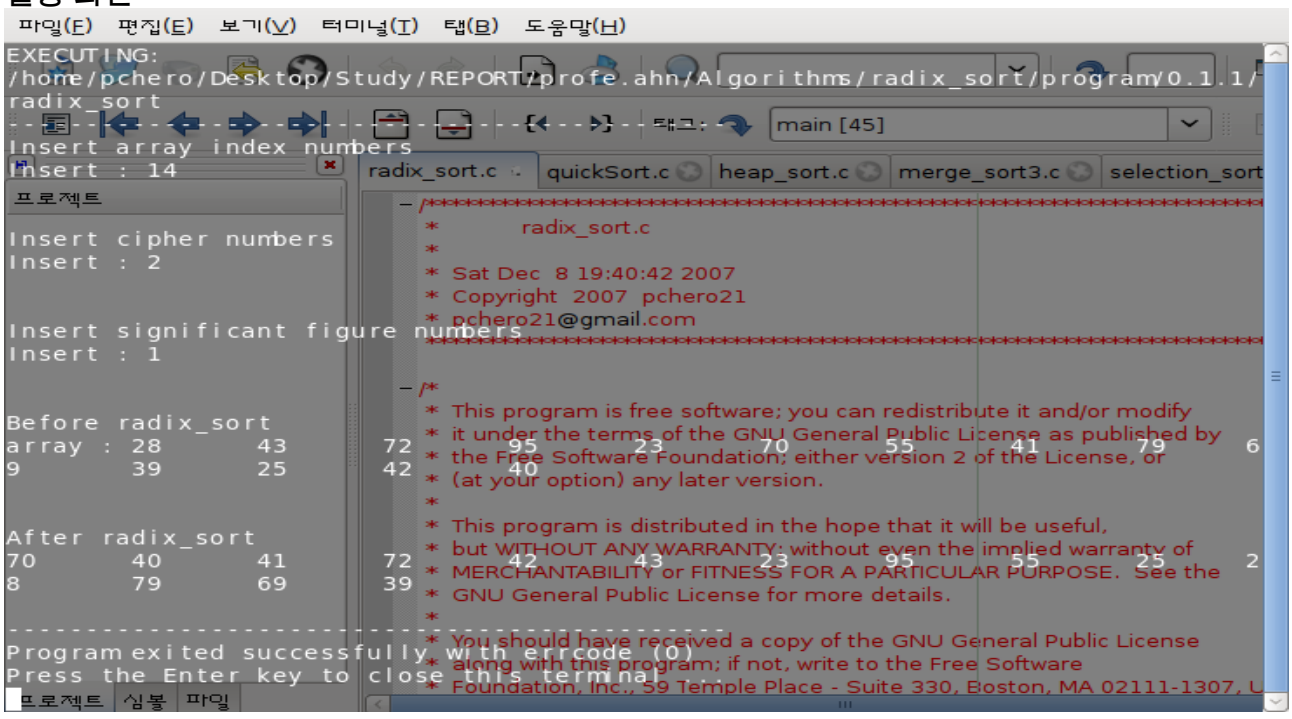
구현 소스

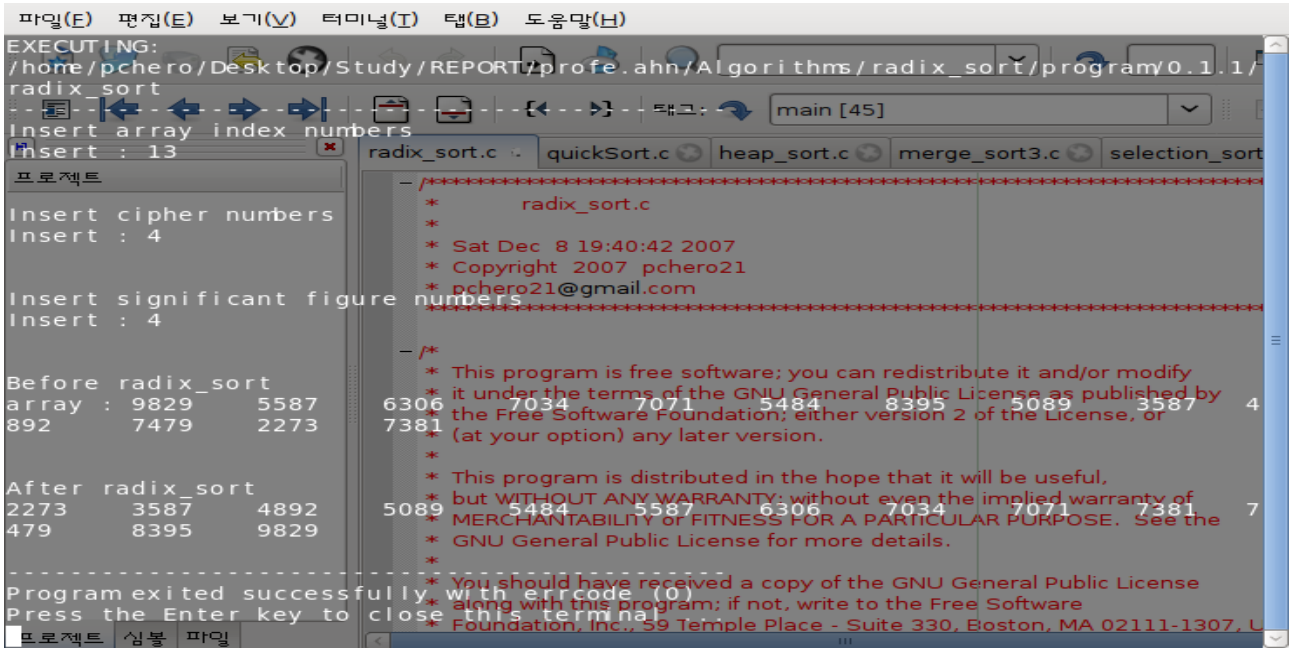
```
#include  <limits.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/*****************************************************************************/
/*                                                                           Functions

                                                */

/* radix sort */
int rxsort(int *data, int size, int p, int k);

/* make random array */
void make_RanArray(int max, int significant, int *array);

/* swap each numbers */
void swap(int *a, int *b);

/*****************************************************************************/
int main()
{
        int array_size;                                 // number of index
        int significant_number;         // significant figure number
        int cipher_number;                      // cipher number
        int *array;                                             // index array
        int i;                                                          // tmp number

        printf("Insert array index numbers\n");
```

```
	printf("Insert : ");
	scanf("%d", &array_size);

	printf("\n\nInsert cipher numbers\n");
	printf("Insert : ");
	scanf("%d", &cipher_number);

	printf("\n\nInsert significant figure numbers\n");
	printf("Insert : ");
	scanf("%d", &significant_number);

	array = (int *)malloc(array_size * sizeof(int));

	make_RanArray(array_size, cipher_number, array);

	printf("\n\nBefore radix_sort\n");
	printf("array : ");
	for(i = 0; i < array_size; i++)
		printf("%d\t", array[i]);
	putchar('\n');

	if(rxsort(array, array_size, significant_number, 10) < 0)
		return -1;

	printf("\n\nAfter radix_sort\n");
	for(i = 0; i < array_size; i++)
		printf("%d\t", array[i]);
	putchar('\n');

	return 0;
}

int rxsort(int *data, int size, int p, int k)
{
	int *counts, *tmp;
	int index, pval, i, j, n;

	/* memory allocate for counts */
	if((counts = (int *)malloc(k * sizeof(int))) == NULL)
		return -1;

	/* memory allocate for sort things */
	if((tmp = (int *)malloc(size * sizeof(int))) == NULL)
		return -1;

	/* sort from minimum position to maximum position */
	for(n = 0; n < p; n++) {
		/* init counts */
		for(i = 0; i < k; i++)
			counts[i] = 0;

		/* calculate location value */
		pval = (int)pow((double)k, (double)n);

		/* counting each number hit */
		for(j = 0; j < size; j++) {
			index = (int)(data[j] / pval) % k;
			counts[index] = counts[index] + 1;
		}

		/* each count setting for reflection the fore count */
		for(i = 1; i < k; i++)
			counts[i] = counts[i] + counts[i - 1];

		/* determine location each item using by counts */
		for(j = size - 1; j >= 0; j--) {
			index = (int)(data[j] / pval) % k;
			tmp[counts[index] - 1] = data[j];
```

```
                        counts[index] = counts[index] - 1;
                }

                /* ready to get the sorted data */
                memcpy(data, tmp, size * sizeof(int));
        }

        /* sorting memory release */
        free(counts);
        free(tmp);

        return 0;
}

void make_RanArray(int max, int significant, int *array)
{
        int i, tmp_number;

        for(i = 0; i < max; i++) {
                array[i] = (rand()  % ((int)(pow(10, significant) - 1)));
                if(array[i] < (int)(pow(10, significant - 1)))
                        i--;
        }

        tmp_number = max;
        for(i = 0; i < tmp_number; i++)
                swap(&array[rand() % (max - 1)], &array[--tmp_number]);
}

void swap(int *a, int *b)
{
        int tmp;

        tmp = *b;
        *b = *a;
        *a = tmp;
}
```

실행 화면




파일(F) 편집(E) 보기(V) 터미널(T) 탭(B) 도움말(H)
EXECUTING:
/home/pchero/Desktop/Study/REPORT/profe.ahn/Algorithms/radix_sort/program/0.1.1/
radix_sort
Insert array index numbers
Insert : 13
Insert cipher numbers
Insert : 4
Insert significant figure numbers
Insert : 4
Before radix_sort
array : 9829 5587 6306 7034 7071 5484 8395 5089 3587 4
892 7479 2273 7381
After radix_sort
2273 3587 4892 5089 5484 5587 6306 7034 7071 7381 7
479 8395 9829
Program exited successfully with errcode (0)
Press the Enter key to close this terminal...
프로젝트
심볼
파일
main [45]
radix_sort.c
quickSort.c
heap_sort.c
merge_sort3.c
selection_sort
radix_sort.c
Sat Dec 8 19:40:42 2007
Copyright 2007 pchero21
pchero21@gmail.com
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, U

8. REFERENCE

*** Introduction to Algorithms / Thomas H. Cormen 외 3 / The MIT Press**

*** C로 구현한 알고리즘 / 카일 루든 / O'Reilly**

*** Foundation of Algorithms usig C++ Pseudocode / Richard Neapolitan 외 1 / jones and Bartlett**

*** http://happycodings.com/**

*** http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/**
자바를 이용하여 알고리즘을 알기 쉽게 표현해 놓은 곳 **(**설명 모두들 이곳에서 펐습니다**.)**