**과제2**

**16010946 김은숙**

**소스코드 :**

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <time.h>

#define N 10000 //리스트의 크기

#define Limit 8000//거의 정렬과 역정렬을 위한 설정

//리스트가 담긴 구조체

typedef struct Arrays {

int \*A, \*B, \*C, \*A1, \*B1, \*C1;

}Arrays;

//퀵의 기준점 반환 구조체

typedef struct quick {

int a, b;

}quick;

//자리 바꾸는 함수

void swap(int \*a, int \*b) {

int tmp;

tmp = \*a;

\*a = \*b;

\*b = tmp;

}

//병합하는 함수

void merge(int\* A, int l, int m, int r) {

int i = l, k = l;

int j = m + 1;

int \*B = (int\*)malloc(sizeof(int)\*N);//N크기의 배열 동적할당

//두개의 리스트들의 원소들 중 작은 원소부터 외부 배열의 동일한 위치에 반복적으로 복사

while (i <= m&&j <= r) {

if (A[i] <= A[j]) {

B[k++] = A[i++];

}

else {

B[k++] = A[j++];

}

}

// 방금의 반복 과정에서 처리되지 않은 원소들을 외부 배열에 반복적으로 복사

while (i <= m) {

B[k++] = A[i++];

}

while (j <= r) {

B[k++] = A[j++];

}

//원래의 배열에 외부배열의 원소들을 복사해줌

for (k = l; k <= r; k++) {

A[k] = B[k];

}

}

void rMergeSort(int\* A, int l, int r) {

int m;

if (l < r) {

//리스트를 반으로 분할한다.

m = (l + r) / 2;

//재귀를 통해 계속 왼쪽과 오른쪽을 계속 반으로 분할한다.

rMergeSort(A, l, m);

rMergeSort(A, m + 1, r);

//병합한다.

merge(A, l, m, r);

}

}

//재귀로 하는 mergesort를 부르는 함수

void mergeSort(int\* A) {

rMergeSort(A, 0, N - 1);

}

//제자리 분할 함수

quick inPlacePartition(int \*A, int l, int r, int k) {

int p, i, j;

int a, b;

int sav;

quick q;

p = A[k];//기준원소 pivot을 저장한다.

i = l;

j = r - 1;

//중복된 원소를 왼쪽과 오른쪽으로 몰기 위한 변수

a = l;

b = r;

//i가 j를 넘지 않는 동안 정렬 진행

while (i <= j) {

//기준점보다 작거나 같은 값을 찾음

while (i <= j&&A[i] <= p) {

//같으면 swap

if (A[i] == p) {

swap(&(A[a]), &(A[i]));

a++;

}

i = i + 1;

}

//기준점보다 크거나 같은 값을 찾음

while (j >= i&&A[j] >= p) {

//기준점이랑 같은 값이면 배열 오른쪽에 둠

if (A[j] == p) {

b--;

swap(A + j, A + b);

}

j = j - 1;

}

//i가 j보다 작으면 swap

if (i < j) {

swap(A + i, A + j);

}

}

//몰아 놓은 중복 원소들을 배열 중앙으로 보낸다.

sav = i - 1;

//a가i보다 큰 동안 중복된 값을 중앙으로 옮긴다.

while (a > i) {

swap(A + a, A + sav);

a--;

sav--;

}

a = sav + 1;

sav = i;

//b가 r+1보다 작은 동안 중복된 값을 중앙으로 옮긴다.

while (b < r + 1) {

swap(A + b, A + sav);

b++;

sav++;

}

b = sav - 1;

q.a = a;

q.b = b;

return q;

}

//제자리 정렬 함수

void inPlaceQuickSort(int \*A, int l, int r) {

quick q;

int k;

//왼쪽 경계가 오른쪽 경계보다 작으면 끝남

if (l >= r) {

return;

}

k = r;//마지막 원소를 기준으로 잡음

//제자리 분할 함수 호출하여 경계점 반환

q = inPlacePartition(A, l, r, k);

//재귀를 통한 정렬진행

inPlaceQuickSort(A, l, q.a - 1);

inPlaceQuickSort(A, q.b + 1, r);

}

//거의 역정렬된 리스트 만드는 함수

void almostInPlaceQuickSort(int \*A, int l, int r) {

quick q;

//limit까지 역정렬해 거의 역정렬된 리스트를 만든다.

if ((r - l) > Limit) {

q = inPlacePartition(A, l, r, 0);

almostInPlaceQuickSort(A, l, q.a - 1);

almostInPlaceQuickSort(A, q.b + 1, r);

}

}

//거의 역정렬 제자리 분할

quick inverseInPlacePartition(int \*A, int l, int r, int k) {

int p, i, j;

int a, b;

int sav;

quick q;

p = A[k];//기준원소 pivot을 저장한다.

i = l;

j = r - 1;

a = l;

b = r;

//역정렬을 위해 i가 j를 넘어가지 않는 동안만 swap하는 과정

while (i <= j) {

while (i <= j&&A[i] >= p) {//기준원소보다 크거나 같은 동안 진행

if (A[i] == p) {//기준원소랑 같으면 swap

swap(A + a, A + i);

a++;

}

i = i + 1;

}

//기준원소보다 작거나 같은동안 진행

while (j >= i&&A[j] <= p) {

//기준원소랑 같으면 swap

if (A[j] == p) {

b--;

swap(A + j, A + b);

}

j = j - 1;

}

//i가 j보다 작으면 swap

if (i < j) {

swap(A + i, A + j);

}

}

//아래는 inplacesort와 같다.

//중복되는 원소의 자리를 찾아주는 작업

sav=i-1;

while (a > i) {

swap(A + a, A + sav);

a--;

sav--;

}

a = sav + 1;

sav = i;

while (b < r + 1) {

swap(A + b, A + sav);

b++;

sav++;

}

b = sav - 1;

q.a = a;

q.b = b;

return q;

}

//거의 역정렬 리스트를 만드는 함수

void inverseInPlaceQuickSort(int \*A, int l, int r) {

quick q;

//limit을 두어 어느정도까지만 역정렬을 만든다.

if ((r - l) > Limit) {

q = inverseInPlacePartition(A, l, r, 0);

inverseInPlaceQuickSort(A, l, q.a - 1);

inverseInPlaceQuickSort(A, q.b + 1, r);

}

}

//배열의 리스트를 만드는 함수

Arrays createArrays() {

Arrays arr;

int i;

//N크기로 동적할당

arr.A = (int\*)malloc(sizeof(int)\*N);

arr.B = (int\*)malloc(sizeof(int)\*N);

arr.C = (int\*)malloc(sizeof(int)\*N);

arr.A1 = (int\*)malloc(sizeof(int)\*N);

arr.B1 = (int\*)malloc(sizeof(int)\*N);

arr.C1 = (int\*)malloc(sizeof(int)\*N);

//A리스트에 랜덤한 수 대입

srand((unsigned)time(NULL));

for (i = 0; i < N; i++) {

arr.A[i] = rand() % N + 1;

}

//B와 C에도 A의 랜덤한 수 대입

for (i = 0; i < N; i++) {

arr.B[i] = arr.A[i];

arr.C[i] = arr.A[i];

}

//거의 정렬된 리스트 생성

almostInPlaceQuickSort(arr.B, 0, N - 1);

//거의 역정렬된 리스트 생성

inverseInPlaceQuickSort(arr.C, 0, N - 1);

//A1, B1, C1에 각각 대입

for (i = 0; i < N; i++) {

arr.A1[i] = arr.A[i];

arr.B1[i] = arr.B[i];

arr.C1[i] = arr.C[i];

}

//구조체 반환

return arr;

}

void main() {

Arrays arr;//배열구조체 생성

LARGE\_INTEGER Start, End, Freq;//실행시간 재는 변수

double run;//실행시간을 재는 변수

//배열의 구조체에 리스트를 대입

arr = createArrays();

//무작위한 리스트

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

inPlaceQuickSort(arr.A, 0, N - 1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행 시간 계산

printf("제자리퀵정렬(무작위)의 CPU time = %0.10lf\n", run);//실행시간 출력

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

mergeSort(arr.A1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행시간 계산

printf("합병정렬(무작위)의 CPI time = %0.10lf\n", run);//실행시간 출력

//거의 정렬된 리스트

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

inPlaceQuickSort(arr.B, 0, N - 1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행 시간 계산

printf("제자리퀵정렬(거의정렬)의 CPU time = %0.10lf\n", run);//실행시간 출력

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

mergeSort(arr.B1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행시간 계산

printf("합병정렬(거의정렬)의 CPI time = %0.10lf\n", run);//실행시간 출력

//거의 역정렬된 리스트

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

inPlaceQuickSort(arr.C, 0, N - 1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행 시간 계산

printf("제자리퀵정렬(거의 역정렬)의 CPU time = %0.10lf\n", run);//실행시간 출력

QueryPerformanceFrequency(&Freq);

QueryPerformanceCounter(&Start);

mergeSort(arr.C1);

QueryPerformanceCounter(&End);

run = (double)(End.QuadPart - Start.QuadPart) / (double)Freq.QuadPart;//실행시간 계산

printf("합병정렬(거의 역정렬)의 CPI time = %0.10lf\n", run);//실행시간 출력

}

**캡쳐 이미지 :**

