PHW3

Problem 1 code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BUCKET\_SIZE 33

typedef struct hashTable {

char value[10];

struct hashTable\* next;

}Hash;

Hash table[BUCKET\_SIZE];

//description : produces new node using malloc to connect to overflow key in hash table.

//input : key value to do overflow chaining.

//output : returns a new node's address including initialized member value.

Hash\* newNode(char\* key) {

Hash\* node;

node = (Hash\*)malloc(sizeof(Hash));

node->next = NULL;

strcpy(node->value, key);

return node;

}

//description : plus ASCII code value of each alphabet for condition of hash function(sum % 33).

//input : a string to plus each alphabet.

//output : returns sum of ASCII code value of each alphabet.

int ASCIISum(char\* key) {

int sum = 0;

int i = 0;

while (key[i] != NULL) {

sum += key[i];

i++;

}

return sum;

}

//description : search a specific key from user in hash table.

//input : key to search

//output : if key found, prints search key, otherwise prints Not found.

void SearchHash(char\* key) {

for (int i = 0; i < BUCKET\_SIZE; i++) {

if (table[i].next == NULL) {

if (strcmp(" ", table[i].value) == 0)continue;

if (strcmp(key, table[i].value) == 0) {

printf("[Search] : Found!\n");

return;

}

}

else {

Hash\* prev = NULL;

Hash\* ptr = table[i].next;

while (ptr != NULL) {

prev = table[i].next;

if (strcmp(key, ptr->value) == 0) {

printf("[Search] : Found!\n");

return;

}

ptr = ptr->next;

}

}

}

printf("[Search] : Not found!\n");

}

//description : insert a specific key from user in hash table

//input : key to insert

//output : if same key is already existed, prints exist! ,otherwise, prints inserted.

void insertHash(char\* key) {

int index = ASCIISum(key) % BUCKET\_SIZE;

if (table[index].next == NULL) {

if (strcmp(key, table[index].value) == 0) {

printf("[Insert] : exists!\n");

return;

}

if (strcmp(" ", table[index].value) == 0) {

strcpy(table[index].value, key);

printf("[Insert] : inserted!\n");

return;

}

}

else {

int flag = 0;

Hash\* prev = NULL;

Hash\* ptr = table[index].next;

while (ptr != NULL) {

if (strcmp(key, table[index].next->value) == 0) {

printf("[Insert] : exists!\n");

return;

}

prev = ptr;

ptr = ptr->next;

if (ptr == NULL) {

ptr = prev;

flag = 1;

ptr->next = newNode(key);

printf("[Insert] : inserted!\n");

break;

}

}

if (!flag) {

table[index].next = newNode(key);

printf("[Insert] : inserted!\n");

flag = 0;

}

}

}

//description : delete a specific key from user in the hash table

//input : key to delete

//output : if key found, prints delete, otherwise prints Not found.

void deleteHash(char\* key) {

for (int i = 0; i < BUCKET\_SIZE; i++) {

if (table[i].next == NULL) {

if (strcmp(" ", table[i].value) == 0) continue;

if (strcmp(key, table[i].value) == 0) {

if (table[i].next == NULL) {

printf("[Delete] : deleted!\n");

strcpy(table[i].value, " ");

return;

}

}

}

else {

Hash\* prev = NULL;

Hash\* ptr = table[i].next;

while (ptr != NULL) {

prev = ptr;

if (strcmp(key, ptr->value) == 0) {

printf("[Delete] : deleted!\n");

if (ptr->next != NULL) {

table[i].next = ptr->next;

ptr->next = NULL;

}

else {

ptr = NULL;

}

free(ptr);

return;

}

ptr = ptr->next;

}

}

}

printf("[Delete] : Not found!\n");

}

//description : prints structurally all values in hash table.

//input : none

//output : prints struct of hash table including overflow.

void printTable() {

printf("------------------------------------\n");

printf("| index | bucket | overflow Chaining => \n");

for (int i = 0; i < BUCKET\_SIZE; i++) {

printf(" %d ", i);

if (strcmp(" ", table[i].value) == 0) printf("(null)");

else printf("%s ", table[i].value);

if (table[i].next == NULL) printf("");

else printf("%s ", table[i].next->value);

if (table[i].next == NULL) printf("");

else printf("%s ", table[i].next->next->value);

if (table[i].next == NULL || table[i].next->next == NULL) printf("");

else printf("%s ", table[i].next->next->next->value);

printf("\n");

}

printf("------------------------------------\n");

}

int main() {

char buff[20][10];

FILE\* fp;

if ((fp = fopen("keyinput.txt", "r")) == NULL) {

printf("Error : read file failure\n");

return 0;

}

for (int i = 0; i < 20; i++) fscanf(fp, "%s", buff[i]);

for (int i = 0; i < BUCKET\_SIZE; i++) {

table[i].next = NULL;

strcpy(table[i].value, " ");

}

int index = 0;

for (int i = 0; i < 20; i++) {

index = ASCIISum(buff[i]) % BUCKET\_SIZE;

if (strcmp(" ", table[index].value) == 0) {

strcpy(table[index].value, buff[i]);

}

else {

int flag = 0;

Hash\* prev = NULL;

while (table[index].next != NULL) {

prev = table[index].next;

table[index].next = table[index].next->next;

if (table[index].next == NULL) {

table[index].next = prev;

flag = 1;

table[index].next->next = newNode(buff[i]);

break;

}

}

if (!flag) {

table[index].next = newNode(buff[i]);

flag = 0;

}

}

}

printf("\n");

printTable();

printf("\n");

SearchHash("Blue");

SearchHash("black");

SearchHash("Purple");

printf("\n");

deleteHash("Purple");

deleteHash("Blue");

deleteHash("Green");

printf("\n");

insertHash("Green");

insertHash("White");

insertHash("Golden");

insertHash("nedloG");

printf("\n");

SearchHash("Blue");

SearchHash("nedloG");

SearchHash("Yellow");

SearchHash("Green");

printf("\n");

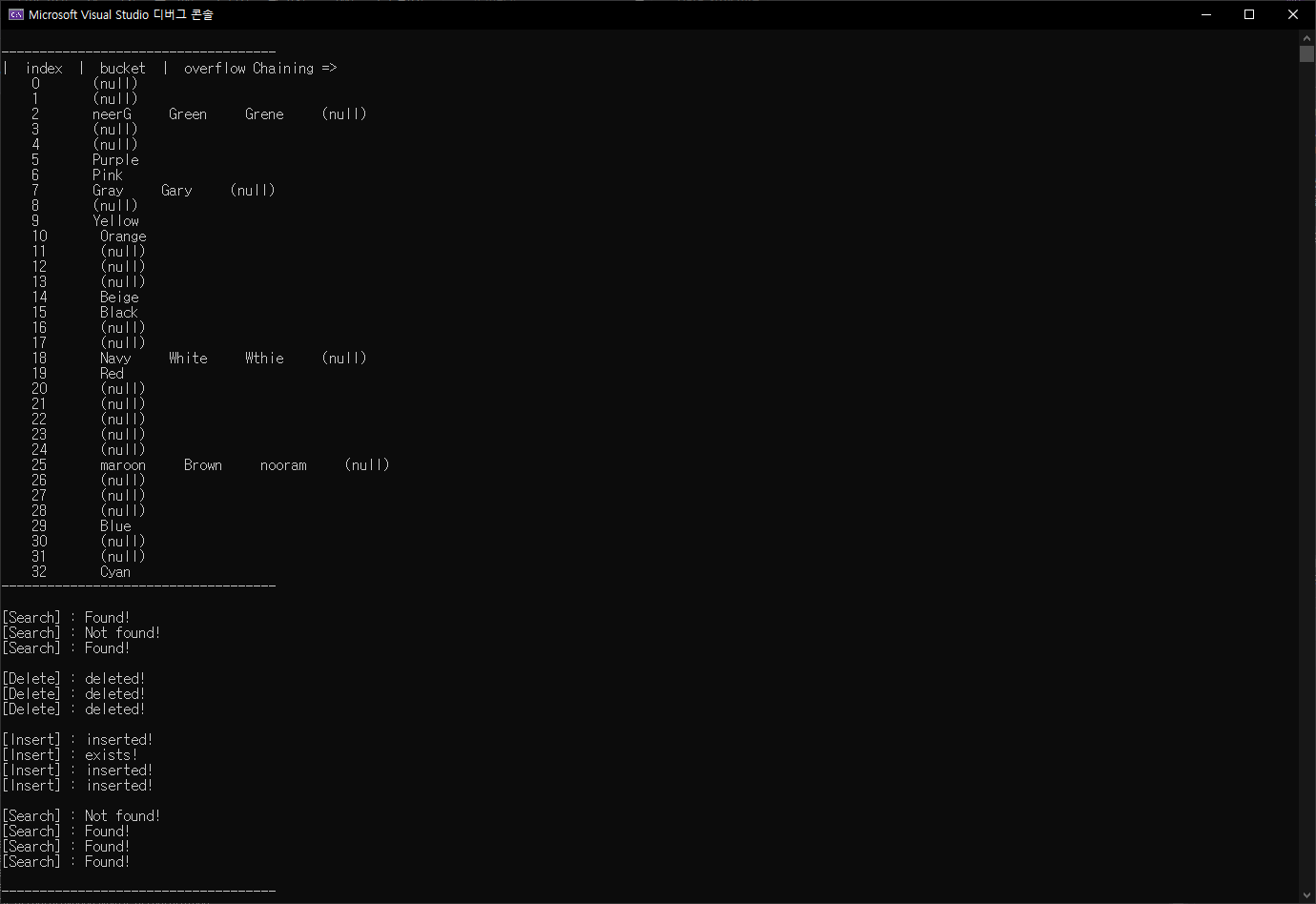
printTable();

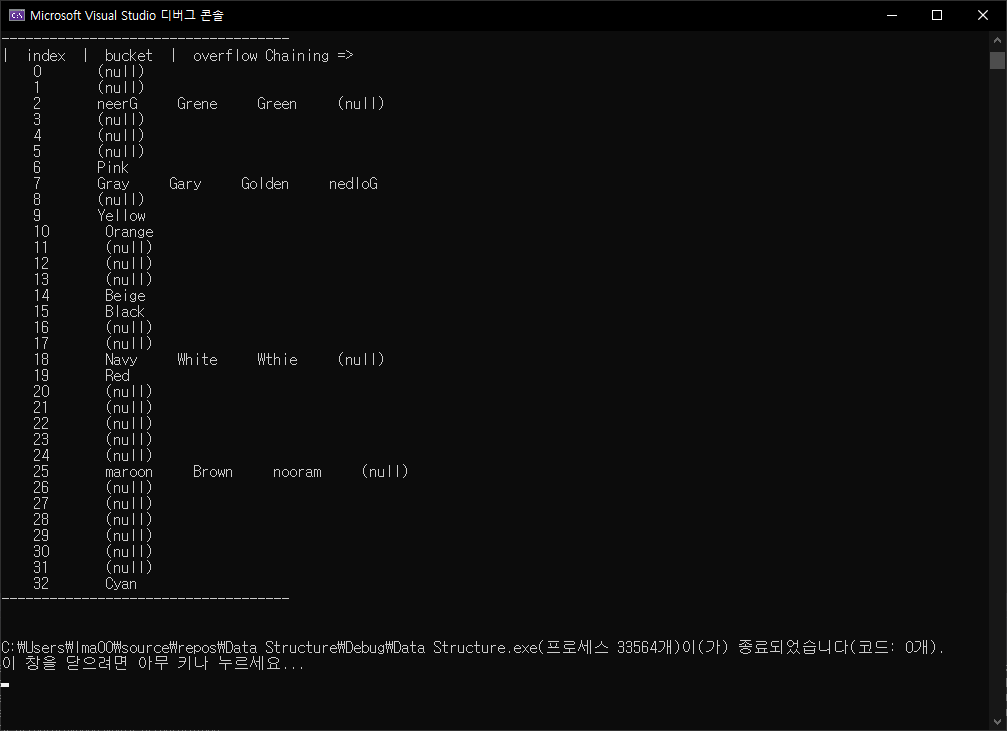
printf("\n");

fclose(fp);

}

Problem 1 result





Problem 2 part1&part2-1 code

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void dupSelSort(int low, int high, int arr[][3], int col);

int arr1[15][3];

int arr2[10][3];

// duplex selection sort function

void dupSelSort(int low, int high, int arr[][3], int col)

{

int lTarget = low;

int rTarget = high;

for (int i = low; i <= high; ++i)

if (arr[lTarget][col] > arr[i][col])

lTarget = i;

for (int i = 0; i < 3; i++) {

int swapTemp = arr[lTarget][i];

arr[lTarget][i] = arr[low][i];

arr[low][i] = swapTemp;

}

for (int i = low; i <= high; ++i)

if (arr[rTarget][col] < arr[i][col])

rTarget = i;

for (int i = 0; i < 3; i++) {

int swapTemp = arr[rTarget][i];

arr[rTarget][i] = arr[high][i];

arr[high][i] = swapTemp;

}

}

int main() {

// create random number(range of 1~100)\_table1

srand(time(NULL));

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 3; j++) {

arr1[i][j] = rand() % 100 + 1;

}

}

// create random number(range of 1~100)\_table2

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 3; j++) {

arr2[i][j] = rand() % 100 + 1;

}

}

// print part1

printf("<Part 1>\n");

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

// update the tables

arr1[9][2] = 55;

arr1[13][2] = 55;

arr2[4][0] = 55;

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

for (int i = 0; i < 15 / 2; ++i) {

dupSelSort(i, 15 - 1 - i, arr1, 2);

}

for (int i = 0; i < 10 / 2; ++i) {

dupSelSort(i, 10 - 1 - i, arr2, 0);

}

// print part 2-1

printf("<Part 2-1>\n");

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

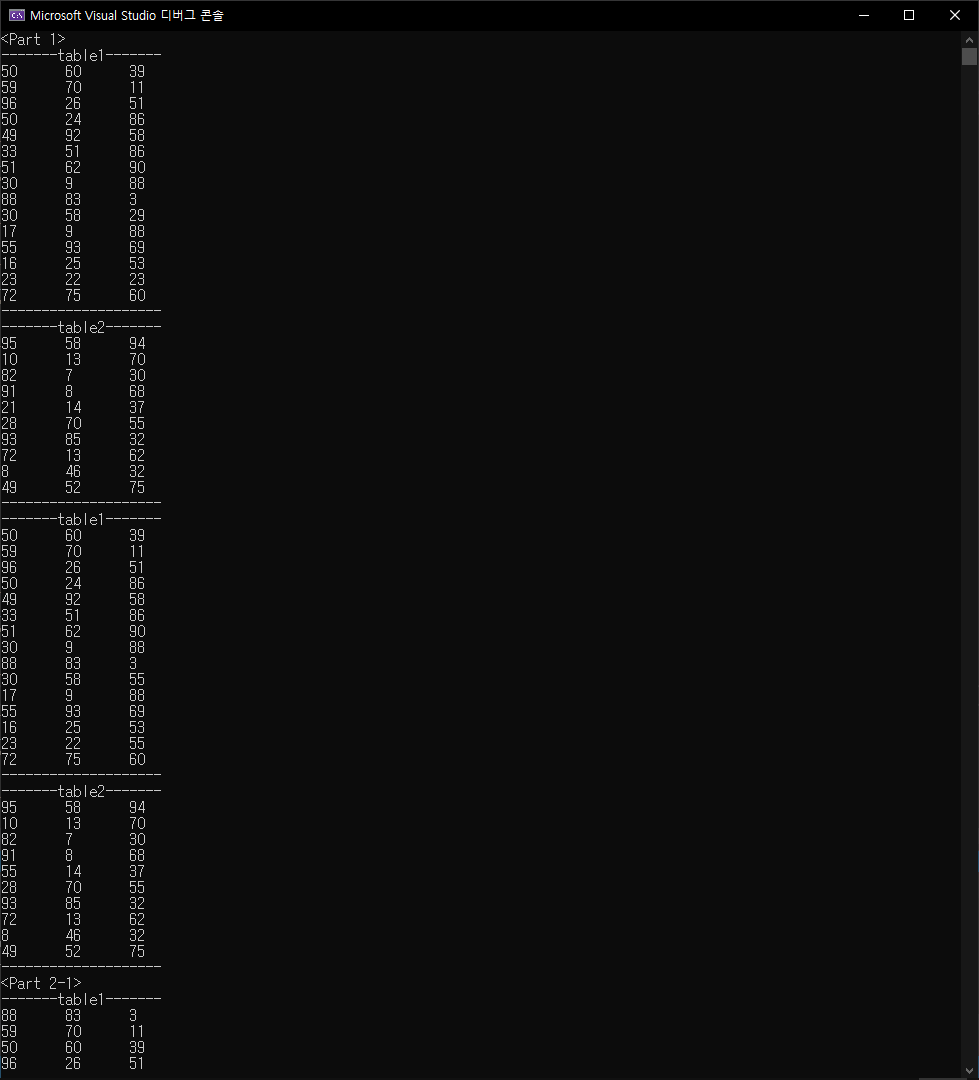
printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

}

Problem 2 part1&part2-1 result





Problem 2 part2-2 code

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int arr1[15][3];

int arr2[10][3];

// quick sort function

void quicksort(int arr[][3], int first, int last, int col) {

int i, j, pivot, temp;

if (first < last) {

pivot = first;

i = first;

j = last;

while (i < j) {

while (arr[i][col] <= arr[pivot][col] && i < last)

i++;

while (arr[j][col] > arr[pivot][col])

j--;

for (int k = 0; k < 3; k++) {

if (i < j) {

temp = arr[i][k];

arr[i][k] = arr[j][k];

arr[j][k] = temp;

}

}

}

for (int k = 0; k < 3; k++) {

temp = arr[pivot][k];

arr[pivot][k] = arr[j][k];

arr[j][k] = temp;

}

quicksort(arr, first, j - 1, col);

quicksort(arr, j + 1, last, col);

}

}

int main() {

// create random number(range of 1~100)\_table1

srand(time(NULL));

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 3; j++) {

arr1[i][j] = rand() % 100 + 1;

}

}

// create random number(range of 1~100)\_table2

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 3; j++) {

arr2[i][j] = rand() % 100 + 1;

}

}

// update the tables

arr1[9][2] = 55;

arr1[13][2] = 55;

arr2[4][0] = 55;

// print part1

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

// quicksort table1

quicksort(arr1, 0, 14, 2);

// quicksort table2

quicksort(arr2, 0, 9, 0);

// print part2-2

printf("<Part 2-2>\n");

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

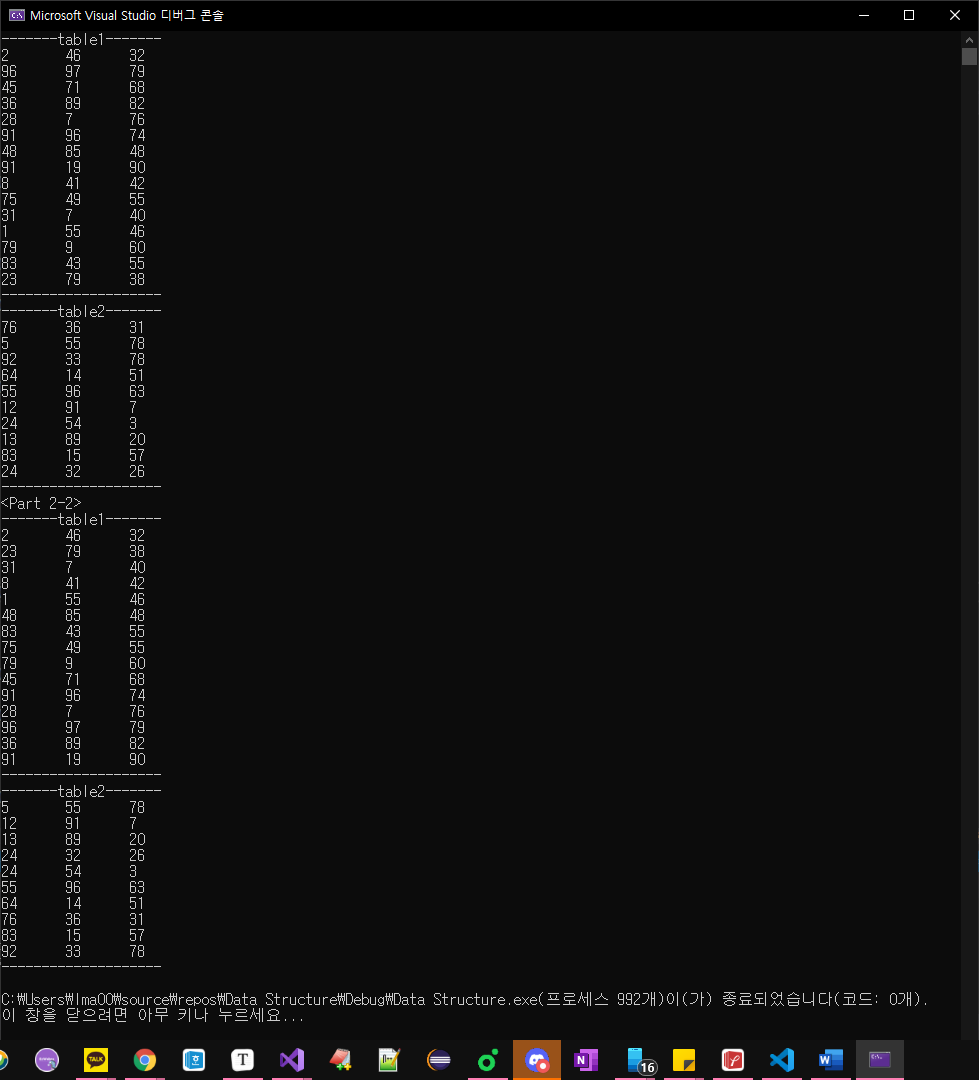
printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

}

Problem 2 part 2-2 result



Problem 2 part 3 code

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int arr1[15][3];

int arr2[10][3];

// quick sort function

void quicksort(int arr[][3], int first, int last, int col) {

int i, j, pivot, temp;

if (first < last) {

pivot = first;

i = first;

j = last;

while (i < j) {

while (arr[i][col] <= arr[pivot][col] && i < last)

i++;

while (arr[j][col] > arr[pivot][col])

j--;

for (int k = 0; k < 3; k++) {

if (i < j) {

temp = arr[i][k];

arr[i][k] = arr[j][k];

arr[j][k] = temp;

}

}

}

for (int k = 0; k < 3; k++) {

temp = arr[pivot][k];

arr[pivot][k] = arr[j][k];

arr[j][k] = temp;

}

quicksort(arr, first, j - 1, col);

quicksort(arr, j + 1, last, col);

}

}

// mergesort function

void mergesort(int x[15][3], int y[10][3], int z[25][3]) {

int temp1=0, temp2=0;

while (temp1 < 15 && temp2 < 10) {

if (x[temp1][2] < y[temp2][0]) {

for (int i = 0;i < 3;i++) {

z[temp1 + temp2][i] = x[temp1][i];

}

temp1 += 1;

}

else {

for (int i = 0;i < 3;i++) {

z[temp1 + temp2][i] = y[temp2][i];

}

temp2 += 1;

}

}

if (temp1 == 15) {

while (temp2 < 10) {

for (int i = 0;i < 3;i++) {

z[temp1 + temp2][i] = y[temp2][i];

}

temp2 += 1;

}

}

else {

while (temp1 < 15) {

for (int i = 0;i < 3;i++) {

z[temp1 + temp2][i] = x[temp1][i];

}

temp1 += 1;

}

}

}

int main() {

// create random number(range of 1~100)\_table1

srand(time(NULL));

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 3; j++) {

arr1[i][j] = rand() % 100 + 1;

}

}

// create random number(range of 1~100)\_table2

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 3; j++) {

arr2[i][j] = rand() % 100 + 1;

}

}

// update the tables

arr1[9][2] = 55;

arr1[13][2] = 55;

arr2[4][0] = 55;

// print part 1

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

//print part 2-2

printf("<Part 2-2>\n");

printf("-------table1-------\n");

for (int i = 0; i < 15; i++) {

printf("%d %d %d\n", arr1[i][0], arr1[i][1], arr1[i][2]);

}

printf("--------------------\n");

printf("-------table2-------\n");

for (int i = 0; i < 10; i++) {

printf("%d %d %d\n", arr2[i][0], arr2[i][1], arr2[i][2]);

}

printf("--------------------\n");

int arr3[25][3];

mergesort(arr1, arr2, arr3);

// print part 3

printf("----merge table-----\n");

for (int i = 0; i < 25; i++) {

printf("%d %d %d\n", arr3[i][0], arr3[i][1], arr3[i][2]);

}

printf("--------------------\n");

}

Problem 2 part 3 result

