1. **소스코드 및 주석**

#include <iostream>

#include <random>

#include <limits.h>

using namespace std;

int RandomNumber(int start, int end) {

random\_device rd;

mt19937\_64 rng(rd());

uniform\_int\_distribution<\_\_int64> dist(start, end);

int number = dist(rng);

return number;

}

void SwapArray(int\* arr, int index1ToChange, int index2ToChange) {

int temp = arr[index1ToChange];

arr[index1ToChange] = arr[index2ToChange];

arr[index2ToChange] = temp;

}

int Selection(int\* arr, int left, int right, int k, int\* p\_comparisonCount)

{

if (left == right)

return arr[left];

int pivot = (left + right) / 2;

SwapArray(arr, pivot, left);

int low = left + 1;

int high = right;

while (low <= high) {

while (low <= right && arr[low] <= arr[left]) {

\*p\_comparisonCount += 1;

low++;

}

while (high >= left && arr[high] > arr[left]) {

\*p\_comparisonCount += 1;

high--;

}

if (low < high) {

SwapArray(arr, low, high);

low++;

high--;

}

}

SwapArray(arr, left, low - 1);

int pivotRank = low - left;

if (k == pivotRank)

return arr[low - 1];

else if (k < pivotRank)

return Selection(arr, left, low - 2, k, p\_comparisonCount);

else

return Selection(arr, low, right, k - pivotRank, p\_comparisonCount);

}

void MedianQuickSort(int\* arr, int start, int end, int\* p\_comparisonCount) {

if (start >= end) return;

int pivotIndex = (start + end) / 2;

int pivotVal = arr[pivotIndex];

int left = start;

int i = start;

int right = end;

while (i <= right) {

(\*p\_comparisonCount)++;

if (arr[i] < pivotVal) {

SwapArray(arr, left, i);

left++;

i++;

}

else if (arr[i] > pivotVal) {

SwapArray(arr, i, right);

right--;

}

else {

i++;

}

}

MedianQuickSort(arr, start, left - 1, p\_comparisonCount);

MedianQuickSort(arr, right + 1, end, p\_comparisonCount);

}

int FindMinimum\_k\_times(int\* arr, int length, int k, long long\* p\_comparisonCount3) {

int\* visited = new int[length]();

int result = -1;

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

int minVal = INT\_MAX;

int minIndex = -1;

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

\*p\_comparisonCount3 += 1;

if (!visited[i] && arr[i] < minVal) {

minVal = arr[i];

minIndex = i;

}

}

if (minIndex != -1) {

visited[minIndex] = 1;

result = minVal;

}

else {

break;

}

}

delete[] visited;

return result;

}

int main() {

int n = RandomNumber(500, 1000);

int rows = n, cols = n;

int\*\* arr\_2d = new int\* [rows];

int\*\* arr2\_2d = new int\* [rows];

int\*\* arr3\_2d = new int\* [rows];

int randomData = 0;

int temp = 0;

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

arr\_2d[i] = new int[cols];

arr2\_2d[i] = new int[cols];

arr3\_2d[i] = new int[cols];

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

randomData = RandomNumber(1, 100000);

arr\_2d[i][j] = randomData;

arr2\_2d[i][j] = randomData;

arr3\_2d[i][j] = randomData;

}

}

cout << "n : " << n << " | 데이터 수(n\*n) : " << n \* n << "개 | n \* n 배열 3개 초기화 완료됨" << endl;

int arr\_length = cols \* rows;

int arr2\_length = cols \* rows;

int arr3\_length = cols \* rows;

int\* arr = new int[arr\_length]();

int\* arr2 = new int[arr2\_length]();

int\* arr3 = new int[arr3\_length]();

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

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

int index = (i \* cols) + j;

arr[index] = arr\_2d[i][j];

arr2[index] = arr2\_2d[i][j];

arr3[index] = arr3\_2d[i][j];

}

}

cout << "2차원 배열을 1차원 배열로 변환완료했습니다" << endl;

int k = RandomNumber(1, 100000);

int kIndex = k - 1;

cout << "k : " << k << endl;

int kRow = kIndex / cols;

int kCol = kIndex % cols;

cout << "--------------------------------" << endl;

cout << "비교 횟수를 셀때, 조건문은 제외하였음" << endl;

cout << "--------------------------------" << endl << endl;

int comparisonCount = 0, comparisonCount2 = 0;

long long comparisonCount3 = 0;

MedianQuickSort(arr2, 0, arr2\_length -1, &comparisonCount2);

int kth\_smallest\_value, kth\_smallest\_value2, kth\_smallest\_value3;

kth\_smallest\_value = Selection(arr, 0, arr\_length - 1, k, &comparisonCount);

cout << "선택 문제 결과 : Selection Problem" << endl;

cout << "k번째 작은값 : " << kth\_smallest\_value << endl;

cout << "비교 횟수: " << comparisonCount << endl;

cout << "--------------------------------" << endl << endl;

kth\_smallest\_value2 = arr2[k - 1];

cout << "정렬 후 k번째 작은값 찾는 방법 결과 : MedianQuickSort 정렬 후 Search" << endl;

cout << "k번째 작은값 " << "arr2[" << kRow << "][" << kCol << "] : " << kth\_smallest\_value2 << " " << endl;

cout << "비교 횟수: " << comparisonCount2 << endl;

cout << "--------------------------------" << endl << endl;

cout << "MIN값부터 차례로 k번째까지 반복하여 찾는 방법" << endl;

kth\_smallest\_value3 = FindMinimum\_k\_times(arr3, arr3\_length, k, &comparisonCount3);

cout << "k번째 작은값 : " << kth\_smallest\_value3 << endl;

cout << "비교 횟수: " << comparisonCount3 << endl;

cout << "--------------------------------" << endl;

cout << endl << endl;

for (int i = 1; i < arr2\_length; i++) {

if (arr2[i - 1] > arr2[i]) {

cout << "정렬 안 됨: arr2[" << i - 1 << "] = " << arr2[i - 1]

<< ", arr2[" << i << "] = " << arr2[i] << endl;

break;

}

}

cout << "[결론]" << endl;

if(comparisonCount < comparisonCount2 && comparisonCount < comparisonCount3)

cout << "선택 문제 방법이 더 효율적입니다." << endl;

else if (comparisonCount2 < comparisonCount && comparisonCount2 < comparisonCount3)

cout << "정렬 후 k번째 작은값 찾는 방법이 더 효율적입니다." << endl;

else if (comparisonCount3 < comparisonCount && comparisonCount3 < comparisonCount2)

cout << "MIN값부터 차례로 k번째까지 반복하여 찾는 방법이 더 효율적입니다." << endl;

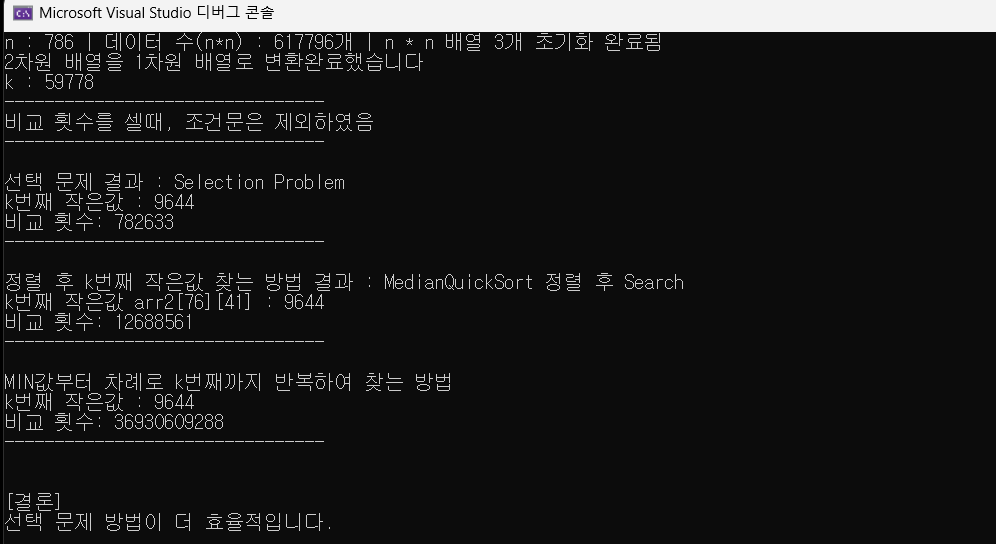
else

cout << "같은 비교횟수가 있거나 비교할 수 없습니다." << endl;

return 0;

}

1. **실행화면 캡처**



* MIN값부터 k번째 작은값 찾는 방법은 실행하는데 시간이 어느정도 걸리기에 기다려야합니다

1. **고찰**

* 입력 데이터가 아주 적을때는, MIN부터 k번째까지 반복하여 찾는 방법이 효율적일순 있으나 입력 데이터가 늘어날 수록 매우 매우 비효율적임 int에서 오버플로우가 생길 정도였다
* 선택 문제 방법은 입력데이터가 매우 클때, 매우 효율적인 시간복잡도를 가지고있었음
* 실제로, 비교횟수를 출력해보니 평균 시간복잡도와 비슷하게 나와 신기하였음
* 2차원 배열을 1차원 배열로 바꿈으로써 복잡도, 가독성을 매우 개선시킬 수 있음을 깨달음