**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант № 2**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Русанова К.В. |
| Преподаватель |  | Тутуева А. В. |

Санкт-Петербург

2020

### Постановка задачи. Описание реализуемых алгоритмов

Реализовать:

1. Двоичный поиск (BinarySearch)

2. Быстрая сортировка (QuickSort)

3. Сортировка пузырьком (BubbleSort)

4. Глупая сортировка (BogoSort)

5. Сортировка подсчётом (CountingSort) для типа char

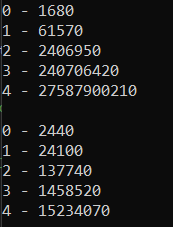
### Оценка временной сложности

Табл.

|  |  |
| --- | --- |
| **Название метода** | **Сложность** |
| BinarySearch | O(log(n)) |
| QuickSort | O(n\*log(n)) |
| BubbleSort | O(n2) |
| BogoSort | O(n\*n!) |
| CountingSort | O(n+k) |

### Сравнение временной сложности алгоритмов

Рис. — Полученные в результате измерений данные в нс (первые данные для пузырьковой сортировки, вторые для быстрой)

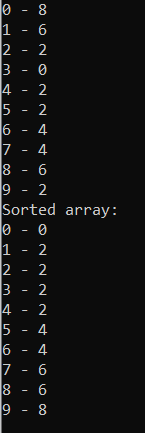


### Реализованные unit-тесты

1. testBinarySearch — проверка бинарного поиска
2. testBinarySearch2 — проверка бинарного поиска
3. testBinarySearch3 — проверка бинарного поиска
4. testBinarySearch4 — проверка бинарного поиска
5. testBinarySearchNotFound — проверка бинарного поиска при отсутствии элемента
6. testBinarySearchNotFound2 — проверка бинарного поиска при отсутствии элемента
7. testBubleSort — проверка пузырьковой сортировки
8. testBubleSortwithemptyplace — проверка пузырьковой сортировки с не идущими подряд элементами
9. testBogoSort — проверка глупой сортировки с не идущими подряд элементами
10. testBogoSortwithemptyplace — проверка глупой сортировки с не идущими подряд элементами
11. testCountingSort — проверка сортировки подсчетом
12. testQuickSort — проверка быстрой сортировки
13. testQuickSortwithemptyplace — проверка быстрой сортировки с не идущими подряд элементами

### Пример работы

Рис. — Массив до и после быстрой сортировки



Лист. — код, показывающий пример работы программы

/\*to demonstrate performance\*/

for (long int i = 0; i < N; i++)

arr1[i] = RANDOM(0, N);

for (size\_t i = 0; i < N; i++)

cout << i << " - " << arr1[i] << endl;

QuickSort(0, N - 1, arr1);

cout << "Sorted array:" << endl;

for (size\_t i = 0; i < N; i++)

cout << i << " - " << arr1[i] << endl;

### Листинг

Лист. — заголовочный файл

#pragma once

void BubleSort(long int N,long int\* arr);//overload for large values

void QuickSort(long int L, long int R, long int\* arr);//overload for large values

int BinarySearch(int\* arr, int N, int search);

void BubleSort(int N, int\* arr);

void BogoSort(int N, int\* arr);

void CountingSort(int N, char\* arr);

void QuickSort(int L, int R, int\* arr);

Лист. 3 — файл программы

#include <iostream>

#include <chrono>

#include <math.h>

#include "lab2.h"

using namespace std;

void QuickSort(long int L, long int R, long int\* arr)

{

long int x = arr[(R + L) / 2]; //choose the middle element as a pivot

long int swapbuf;

long int i = L, j = R;

do

{

while (arr[i] < x)//looking for an element smaller than the pivot

i++;

while (arr[j] > x)//looking for an element bigger than the pivot

j--;

if (i <= j) // swap

{

swapbuf = arr[i];

arr[i] = arr[j];

arr[j] = swapbuf;

i++;

j--;

}

} while (i < j);

if (i < R) //if they did not go beyond the bounds of the array

QuickSort(i, R, arr);

if (j > L)

QuickSort(L, j, arr);

}

void BubleSort(long int N, long int\* arr)

{

long int swapbuf;

for (long int i = 0; i < N - 1; i++) //сортировка пузырьком

{

for (long int j = 0; j < N - i - 1; j++)

{

if (arr[j + 1] < arr[j])

{

swapbuf = arr[j + 1]; //обмен переменных местами

arr[j + 1] = arr[j];

arr[j] = swapbuf;

}

}

}

}

long int RANDOM(long int min, long int max)

{

long int seed = 7678;

return ((rand() \* seed) % (max - min) + min);

}

int main()

{

/\*to collect statistics\*/

const int N = 10; //count test

const int SizeT = 5; //count Time

const long long NN = 100000; //maxSize

long int arr1[NN]; //array int for sort

long int arr2[NN];

long long timeB[SizeT]; //time array

long long timeQ[SizeT];

long long SB = 0, SQ = 0,NowN;

auto begin = std::chrono::steady\_clock::now(); // startTime

auto end = std::chrono::steady\_clock::now(); // endTime

auto elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);

for (unsigned k = 0; k < SizeT; k++) //5 situation

{

NowN = pow(N, k+1);

for (size\_t l = 0; l < N; l++) //10 test

{

for (long int i = 0; i < NowN; i++)

{

arr1[i] = RANDOM(0, NowN);

arr2[i] = arr1[i];

}

begin = std::chrono::steady\_clock::now();

QuickSort(0, NowN - 1, arr1);

end = std::chrono::steady\_clock::now();

elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);

SQ += elapsed\_ns.count();

begin = std::chrono::steady\_clock::now();

BubleSort(NowN, arr2);

end = std::chrono::steady\_clock::now();

elapsed\_ns = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin);

SB += elapsed\_ns.count();

}

timeB[k] = SB / N;

timeQ[k] = SQ / N;

SB = 0;

SQ = 0;

}

for (size\_t i = 0; i < SizeT; i++)

cout << i << " - " << timeB[i] << endl;

cout << endl;

for (size\_t i = 0; i < SizeT; i++)

cout << i << " - " << timeQ[i] << endl;

cout << endl;

/\*to demonstrate performance\*/

for (long int i = 0; i < N; i++)

arr1[i] = RANDOM(0, N);

for (size\_t i = 0; i < N; i++)

cout << i << " - " << arr1[i] << endl;

QuickSort(0, N - 1, arr1);

cout << "Sorted array:" << endl;

for (size\_t i = 0; i < N; i++)

cout << i << " - " << arr1[i] << endl;

}

int BinarySearch(int\* arr, int N, int search)

{

int right = N - 1, left = 0, mid = right / 2; // start position

while ((arr[mid] != search) && (left < right))

{

if (arr[mid] > search)

right = mid - 1;

else

left = mid + 1;

mid = (right + left) / 2; //change mid position

}

if (arr[mid] == search)

return mid;

else

return -1; // if not found

}

void BubleSort(int N, int\* arr)

{

int swapbuf;

for (int i = 0; i < N - 1; i++) //сортировка пузырьком

{

for (int j = 0; j < N - i - 1; j++)

{

if (arr[j + 1] < arr[j])

{

swapbuf = arr[j + 1]; //обмен переменных местами

arr[j + 1] = arr[j];

arr[j] = swapbuf;

}

}

}

}

void BogoSort(int N, int\* arr)

{

int swapbuf;

int randIndex1, randIndex2;

bool IfSorted = false;

int seed = 64531;

int count;

while (!IfSorted)

{

randIndex1 = (rand() \* seed) % N;

randIndex2 = (rand() \* seed) % N;

while (randIndex2 == randIndex1)

randIndex2 = (rand() \* seed) % N;

swapbuf = arr[randIndex1]; //обмен переменных местами

arr[randIndex1] = arr[randIndex2];

arr[randIndex2] = swapbuf;

count = 0;

for (size\_t i = 1; i < N; i++)

if (arr[i - 1] < arr[i])

count++;

if (count == 9)

{

IfSorted = true;

}

}

}

void CountingSort(int N, char\* arr)

{

char min = CHAR\_MAX,max= CHAR\_MIN;

unsigned int dist,controlIndex=0;

for (size\_t i = 0; i < N; i++) //find min and max

{

if (arr[i] < min)

min = arr[i];

if (arr[i] > max)

max = arr[i];

}

dist = max - min+1;

unsigned\* count = new unsigned[dist]; // create new massive

for (size\_t i = 0; i < dist; i++) //initialize zero

count[i] = 0;

for (size\_t i = 0; i < N; i++)

count[arr[i] - min]++;

for (size\_t i = 0; i < dist; i++)

{

for (size\_t j = 0; j < count[i]; j++)

{

arr[controlIndex] = (char)(i + min);

controlIndex++;

}

}

}

void QuickSort(int L, int R, int\* arr)

{

int x = arr[(R + L) / 2]; //choose the middle element as a pivot

int swapbuf;

int i = L, j = R;

do

{

while (arr[i] < x)//looking for an element smaller than the pivot

i++;

while (arr[j] > x)//looking for an element bigger than the pivot

j--;

if (i <= j) // swap

{

swapbuf = arr[i];

arr[i] = arr[j];

arr[j] = swapbuf;

i++;

j--;

}

} while (i<j);

if (i < R) //if they did not go beyond the bounds of the array

QuickSort(i,R,arr);

if (j > L)

QuickSort(L,j,arr);

}