## Федеральное государственное автономное образовательное учреждение высшего образования

## **Санкт-Петербургский национальный исследовательский университет**

# информационных технологий, механики и оптики

# Кафедра ИПМ

# Лабораторная работа №5

1 курс

**Бинарный поиск и бинарное дерево поиска**

# Выполнил

Студент 2 курса

# Группы P3218

# Гхази Даниэль

# Принял:

К.т.н.

Старший преподаватель

Симоненко Зинаида

Григорьевна

Санкт-Петербург

2016

**Цель работы:**

Для выполнения лабораторной работы необходимо сгенерировать тестовые файлы (используя генераторы случайных чисел), содержащие целые числа, в количестве от 2 6 до 2 20 (можно и больше), при этом количество элементов в следующем файле в два раза больше чем в предыдущем, реализовать алгоритмы используя один из следующих языков программирования: C++, C#, C, Python, для каждого тестового файла из набора выполнить сортировку данных, вычислить среднее время сортировки по одному файлу, построить график зависимости времени сортировки от количества элементов в файле, выполнить сравнение алгоритмов

**Код генератора исходных данных:**

int main()

{

srand(time(NULL));

FILE \*file;

if ((file = fopen("mas.txt", "wt")) == NULL)

return 1;

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

fprintf(file, "%i", rand() % ARR\_SIZE);

fclose(file);

return 0;

}

**Коды сортировок:**

**Бинарный поиск**

int first = 0;

int last = ARR\_SIZE;

if (mas[0] > elemToFind)

return -1;

else if (mas[ARR\_SIZE - 1] < elemToFind)

return -2;

while (first < last)

{

int mid = first + (last - first) / 2;

if (elemToFind <= mas[mid])

last = mid;

else

first = mid + 1;

}

if (mas[last] == elemToFind)

return last;

else

return -3;

**Бинарное дерево поиска**

int main()

{

for (int curElemIndex = 0; curElemIndex < ARR\_SIZE; curElemIndex++)

binarySearchTreeCreation(curElemIndex, 1);

}

int binarySearchTreeCreation(int curElemIndex, int curPlaceToTakeIndex)

{

if (binarySearchTree[curPlaceToTakeIndex] == 0)

{

binarySearchTree[curPlaceToTakeIndex] = mas[curElemIndex];

return 0;

}

else if (mas[curElemIndex] >= binarySearchTree[curPlaceToTakeIndex])

{

curPlaceToTakeIndex = curPlaceToTakeIndex \* 2 + 1;

}

else

curPlaceToTakeIndex \*= 2;

binarySearchTreeCreation(curElemIndex, curPlaceToTakeIndex);

return -1;

}

int binarySearchTreeSearch(int elemToFind)

{

int flag = 0;

while ((binarySearchTree[j] != elemToFind) && (flag != 1))

{

if (elemToFind >= binarySearchTree[j])

if (binarySearchTree[j \* 2 + 1] != 0)

j = j \* 2 + 1;

else

flag = 1;

else

if (binarySearchTree[j \* 2] != 0)

j = j \* 2;

else

flag = 1;

}

if (flag == 1)

return -1;

else

return 1;

}

**Результаты:**

|  |  |  |
| --- | --- | --- |
|  | Время поиска (в секундах) | |
| Кол-во эл-в | Бинарный поиск | Бинарное дерево |
| 1800 | 0,00107 | 1,514 |
| 3600 | 0,00111 | 3,752 |
| 7200 | 0,00142 | 9,713 |
| 14400 | 0,00169 | 23,676 |

**Сравнительный анализ:**

**Вывод:**

Сравнив алгоритм бинарного поиска и бинарного дерева, мы выяснили, что алгоритм бинарного слабо зависит от количество входных элементов, в то время как алгоритм бинарного дерева замедляется более чем в два раза при увеличении числа элементов для поиска.