## Федеральное государственное автономное образовательное учреждение высшего образования

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

# информационных технологий, механики и оптики

# Кафедра ИПМ

# Лабораторная работа №3

1 курс

**Сортировки за O(n\*logn)**

# 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 26 до 220 , 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;

}

**Сортировки:**

**Бинарной кучей**

int main()

{

buildHeap(&currSize);

for (int i = ARR\_SIZE - 1; i > 0; i--)

{

heap[i] = extractMax(&currSize);

}

}

int extractMax(int \*currSize)

{

int maxElem = heap[1];

heap[1] = heap[\*currSize];

\*currSize -= 1;

maxHeapify(1, currSize);

return maxElem;

}

void maxHeapify(int index, int \*currSize)

{

int curElem = index;

int l = 2 \* index;

int r = 2 \* index + 1;

if ((l <= \*currSize) && (heap[l] > heap[curElem]))

curElem = l;

if ((r <= \*currSize) && (heap[r] > heap[curElem]))

curElem = r;

if (curElem != index)

{

int temp = heap[index];

heap[index] = heap[curElem];

heap[curElem] = temp;

maxHeapify(curElem, currSize);

}

}

void buildHeap(int \*currSize)

{

for (int i = \*currSize / 2; i > 0; i--)

maxHeapify(i, currSize);

}

**Слиянием**

void mergeSort(int\* mas, int lb, int ub)

{

int split;

if (lb < ub)

{

split = (lb + ub) / 2;

mergeSort(mas, lb, split);

mergeSort(mas, split + 1, ub);

merge(mas, lb, split, ub);

}

}

void merge(int\* mas, int lb, int split, int ub)

{

int pos1 = lb;

int pos2 = split + 1;

int pos3 = 0;

int \*bufMas = (int\*)calloc(ub - lb + 1, sizeof(int));

while ((pos1 <= split) && (pos2 <= ub))

if (mas[pos1] < mas[pos2])

bufMas[pos3++] = mas[pos1++];

else

bufMas[pos3++] = mas[pos2++];

while (pos2 <= ub)

bufMas[pos3++] = mas[pos2++];

while (pos1 <= split)

bufMas[pos3++] = mas[pos1++];

for (pos3 = 0; pos3 < ub - lb + 1; pos3++)

mas[lb + pos3] = bufMas[pos3];

free(bufMas);

}

**Результаты:**

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Время выполнения алгоритма | |
| Сортировка кучей, мс | Сортировка слиянием, мс |
| 16000 | 14 | 2 |
| 32000 | 15 | 3 |
| 64000 | 29 | 13 |
| 128000 | 47 | 45 |
| 256000 | 163 | 72 |
| 512000 | 284 | 149 |

**Выводы:**

Я выполнил сортировку данных, построил график зависимости времени сортировки от количества элементов в файле. В результате выяснилось, что сортировка слиянием в несколько раз эффективнее сортировки кучей, при количестве элементов меньшем 100.000 и большем 140.000. В диапазоне от 100.000 до 140.000 элементов, их эффективность приблизительно равна.