## Федеральное государственное автономное образовательное учреждение высшего образования

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

# информационных технологий, механики и оптики

# Кафедра ИПМ

# Лабораторная работа №4

1 курс

**Сортировки левосторонней кучей и многопутевым слиянием**

# 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:**

**Левосторонняя куча**

int main()

{

buildHeap(&currSize);

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

{

heap[i] = extractMax(&currSize);

}

void buildHeap(int \*currSize)

{

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

maxHeapify(i, currSize);

}

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);

}

}

int extractMax(int \*currSize)

{

int maxElem = heap[1];

creationOfTwoTrees(currSize);

for (int i = 1; i <= rightCurrSize; i++)

insert(i);

\*currSize -= 1;

return maxElem;

}

void creationOfTwoTrees(int\* currSize)

{

int currSizeTemp = \*currSize;

int numberOfFloors = 0;

while (currSizeTemp != 1)

{

currSizeTemp /= 2;

numberOfFloors++;

}

int numberOfLeftLeaves = \*currSize - pow(2, numberOfFloors) + 1;

if (numberOfLeftLeaves > pow(2, numberOfFloors - 1))

numberOfLeftLeaves = pow(2, numberOfFloors - 1);

leftCurrSize = pow(2, numberOfFloors - 1) - 1 + numberOfLeftLeaves;

rightCurrSize = \*currSize - leftCurrSize - 1;

int r = 3;

int k = 0;

for (int i = 1; i <= rightCurrSize; i++)

{

heapRight[i] = heap[r + k];

if (k < r / 3 - 1)

k++;

else

{

r \*= 2;

k = 0;

}

}

r = 2;

k = 0;

for (int i = 1; i <= leftCurrSize; i++)

{

heap[i] = heap[r + k];

if (k < r / 2 - 1)

k++;

else

{

r \*= 2;

k = 0;

}

}

}

void insert(int i)

{

leftCurrSize++;

int index = leftCurrSize;

heap[leftCurrSize] = heapRight[i];

int par = index / 2;

int flag = 0;

while (flag != 1)

{

if (index == 1)

flag = 1;

else if (heap[par] > heapRight[i])

flag = 1;

else

{

int temp = heap[par];

heap[par] = heap[index];

heap[index] = temp;

index = par;

par = index / 2;

}

}

}

**Многопутевое слияние**

public void Sort(T [] items)

{

this.items = items;

Sort(0, items.Length - 1);

}

void Sort (int start, int end)

{

if (start < end)

{

int length = (int)Math.Ceiling((double)(end - start + 1) / Ways);

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

{

int l, r;

l = start + length \* i;

r = start + length \* i + length - 1;

Sort(l, r);

}

Sort(start + length \* (Ways - 1), end);

Merge(start, end, length);

}

}

void Merge(int start, int end, int length)

{

T[] temp = new T[end - start + 1];

// Creating lists for merging

List<List<T>> merging = new List<List<T>>(Ways);

List<T> tmp;

int l, r;

int j;

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

{

l = start + length \* i;

r = start + length \* i + length - 1;

tmp = new List<T>(length);

for (j = l; j < r + 1; j++)

tmp.Add(items[j]);

merging.Add(tmp);

}

if ((r = start + length \* (Ways - 1)) <= end)

{

tmp = new List<T>(end - r + 1);

for (j = r; j < end + 1; j++)

tmp.Add(items[j]);

merging.Add(tmp);

}

j = 0;

while (merging.Count != 0)

{

int min = 0;

for (int i = 1; i < merging.Count; i++)

{

if (merging[i][0].CompareTo(merging[min][0]) < 0)

{

min = i;

}

}

temp[j] = merging[min][0];

merging[min].RemoveAt(0);

if (merging[min].Count == 0)

merging.RemoveAt(min);

j++;

}

j = 0;

for (int i = start; i < end + 1; i++)

{

items[i] = temp[j];

j++;

}

}

**Результаты:**

|  |  |  |
| --- | --- | --- |
| Кол-во эл-в | Время работы сортировок, секунды | |
| Левосторонняя куча | Многопутевое слияние |
| 16000 | 0,05901 | 0,09599 |
| 32000 | 0,11198 | 0,25151 |
| 64000 | 0,32371 | 0,54992 |
| 128000 | 0,78541 | 1,19022 |
| 256000 | 1,22343 | 1,98436 |

**Сравнительный анализ:**

**Выводы:**

Сравнив две вышепредставленные сортировки, мы приходим к выводу, что сортировка левосторонней кучей в среднем в 1.5 раза эффективней сортировки многопутевыми слияниями.