Пирамидальная сортировка

Heap Sort

Пирамидальная сортировка

- Heapsort
- Сортировка кучей
- Сортировка с помощью кучи

- Куча – пирамида – дерево – список – массив

Определения (разговорный язык)

- Heap:
  - an untidy pile of something
    - a collection of things thrown one on another
    - a large, disorganized collection of things
  - [usually plural] heap (of something) (informal) a lot of something

- Idiom: at the top/bottom of the heap
  - high up/low down in the structure of an organization or a society

Определения (информатика)

- *perplexity.ai*
- act as an experienced linguist and information science professor what are definitions of "heap". explain in simple words common definitions and specialized meanings in computer science
- can we compare a binary heap to a pyramid? is there any clue to this meaning when using the idiom "at the top/bottom of the heap"

- 1. **Heap as a Data Structure**: In computer science, a "heap" is a specialized tree-based data structure that satisfies the heap property. There are two types of heaps: a max-heap and a min-heap. In a max-heap, the highest key is at the root, and in a min-heap, the lowest key is at the root. Heaps are commonly used to implement priority queues, and they have various operations such as insertion, deletion, and peeking.
- 2. **Heap as Memory**: In certain programming languages, including C and Pascal, a "heap" also refers to a region of memory that is allocated for dynamic memory allocation. This is distinct from the stack, which is used for static memory allocation. The management of the heap can be done by the application itself or by the operating system.

«Куча» = «пирамида»

- perplexity.ai
- can we compare a binary heap to a pyramid? is there any clue to this meaning when using the idiom "at the top/bottom of the heap“

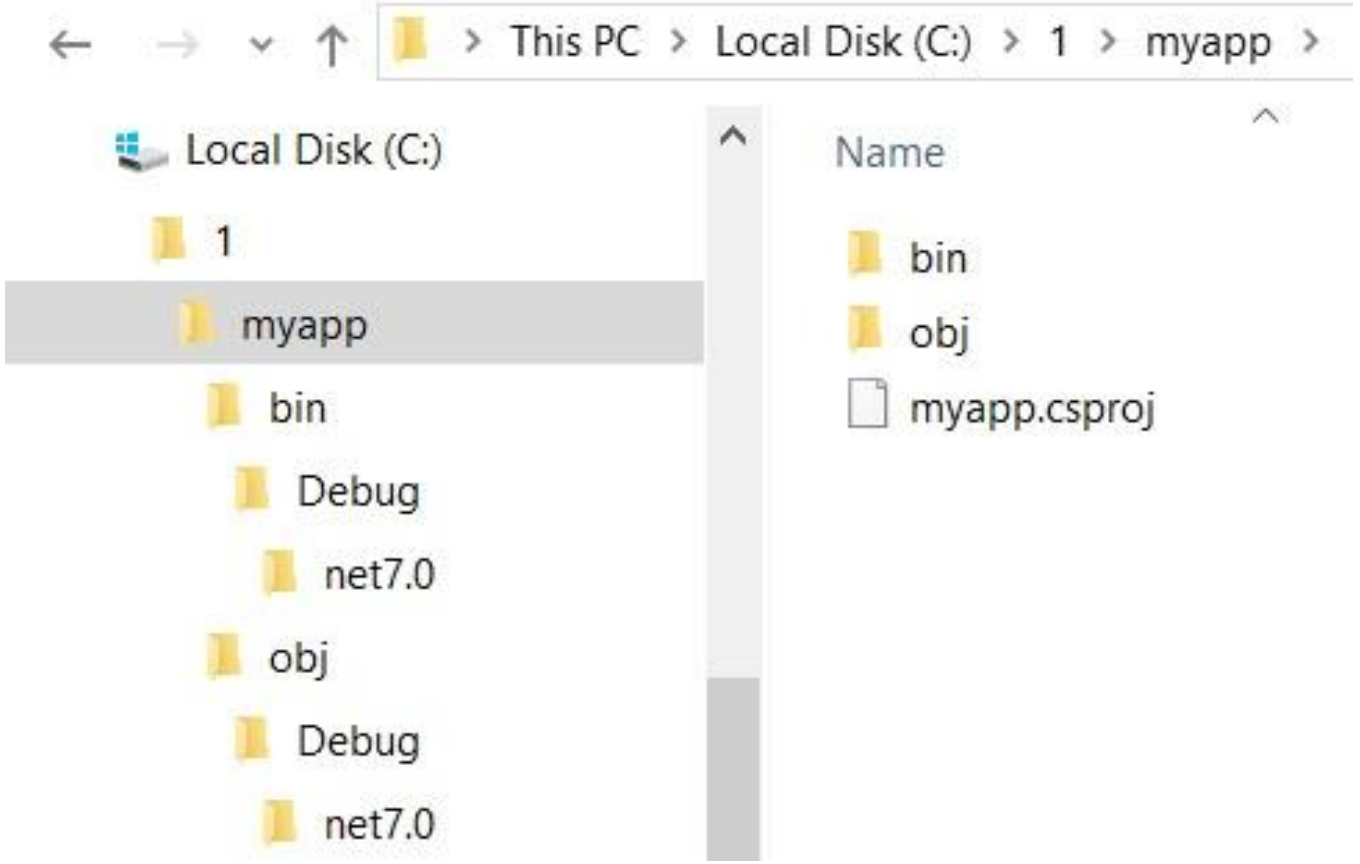
- Yes, a binary heap can be compared to a pyramid. In computer science, a binary heap is a specialized tree-based data structure that satisfies the heap property. It is a pyramidal structure of nodes whose nodes can stack upward with rows of decreasing values toward a minimum (minimum binary heap) or a maximum (maximum binary heap).
- The term "at the top/bottom of the heap" is an idiom that means being in a position of power or being in a position of low status, respectively. This idiom is not directly related to the computer science definition of a heap, but it may have originated from the idea of a heap as a pile or a large, disorganized collection of things.

Сортирующее дерево

- Двоичное (бинарное) дерево
  - *Binary tree*
  - Представление массива
- Сортировка выбором
  - *Selection sort*
  - Отбор максимального значения в выходной массив
- Сортировка методом пузырька
  - *Bubble sort*
  - Поиск максимального значения

Файловое дерево

```
C:\1>tree
Folder PATH listing
Volume serial number is 08E0-0916
C:.
└───myapp
    ├───bin
    │   └───Debug
    │       └───net7.0
    └───obj
        └───Debug
            └───net7.0
                ├───ref
                └───refint
```



Ссылки

- https://en.wikipedia.org/wiki/Heapsort
- https://ru.wikipedia.org/wiki/Пирамидальная_сортировка
- https://prog-cpp.ru/data-heap/
- https://mathworld.wolfram.com/Heapsort.html
- [Кормен Алгоритмы] 7 Сортировка с помощью кучи
- [Кнут том 3]

Дерево с корнем

- Корневое дерево – Rooted tree
- Корень – Root
- Вершина / узел – Vertex / Node
- Внутренняя вершина – Internal node
- Лист – Leaf / External node
- Ребро – Edge
- Глубина – Depth
- Высота – Height
- Родитель – Parent
- Ребенок – Child
- Левый ребенок – Left child
- Правый ребенок – Right child
- Братья – Siblings
- Предок – Ancestor
- Потомок – Descendant
- Полное дерево – Complete tree

Дерево – Tree

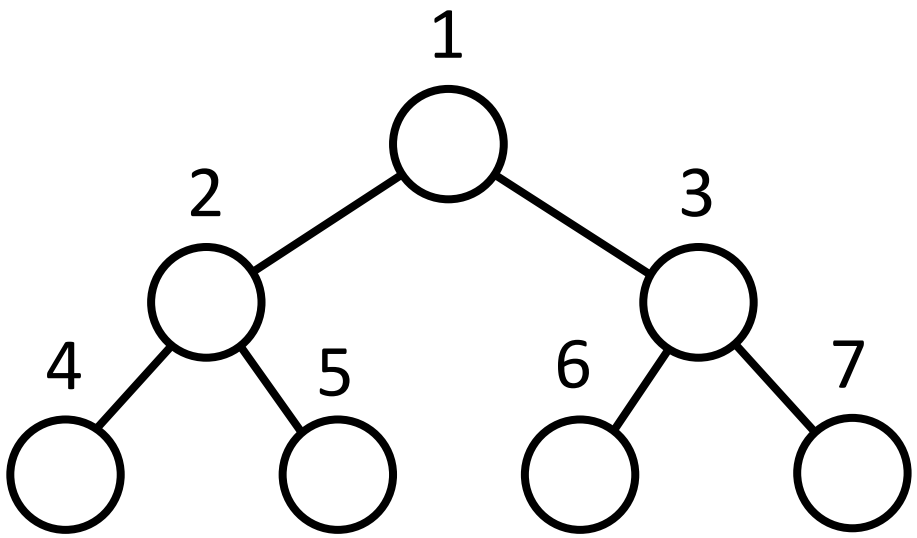
- Связный ациклический неориентированный граф – нарисовать
  - Connected acyclic undirected graph

- Дерево без выделенного корня
- Дерево с корнем

- Число вершин = n
- Число ребер - ?

Родитель-ребенок

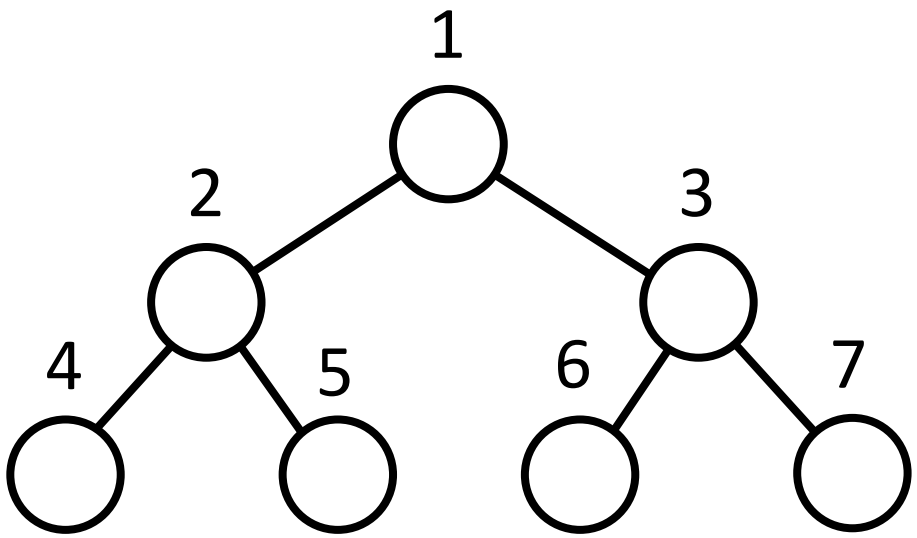
- Корень дерева – первый элемент (нулевой индекс)
- Массив x: [1, 2, 3, 4, 5, 6, 7]




```
i = 2
parent(i), left_child(i), right_child(i)
```

```
(1, 4, 5)
```

Родитель-ребенок



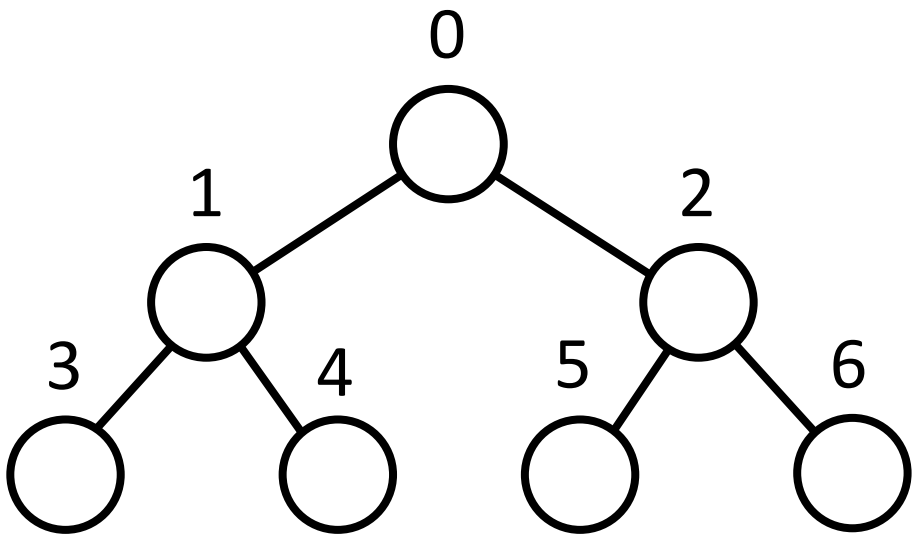

```
i = 2
parent(i), left_child(i), right_child(i)
```

```
(1, 4, 5)
```

```
# Binary tree: (x[1], x[2], x[3], ...)
# 1-indexed array

def parent(i):
    # the parent of the node at index i
    return i // 2

def left_child(i):
    # the left child of the node at index i
    return 2 * i

def right_child(i):
    # the right child of the node at index i
    return 2 * i + 1
```

Родитель-ребенок

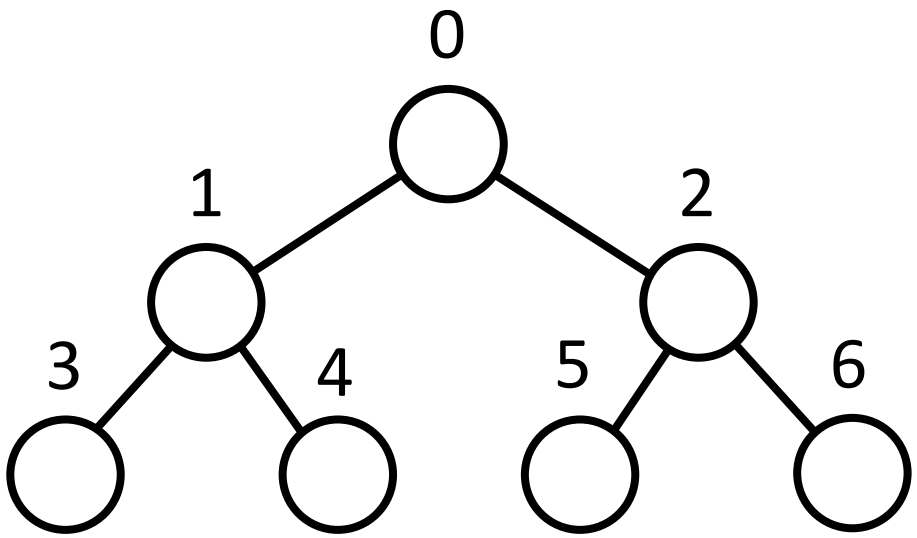
- Корень дерева – элемент с номером 0
- Массив x: [0, 1, 2, 3, 4, 5, 6]




```
i = 2
parent(i), left_child(i), right_child(i)
```

```
(0, 5, 6)
```

Родитель-ребенок



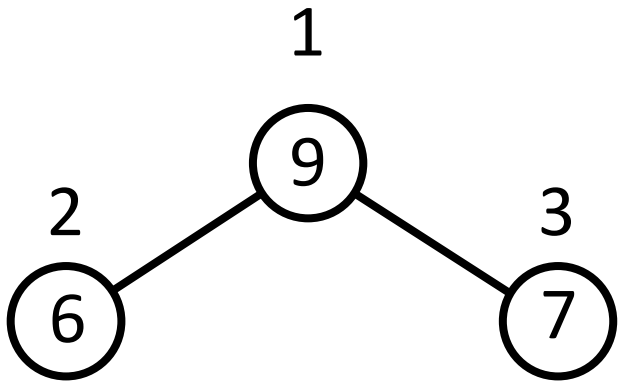

```
i = 2
parent(i), left_child(i), right_child(i)
```

```
(0, 5, 6)
```

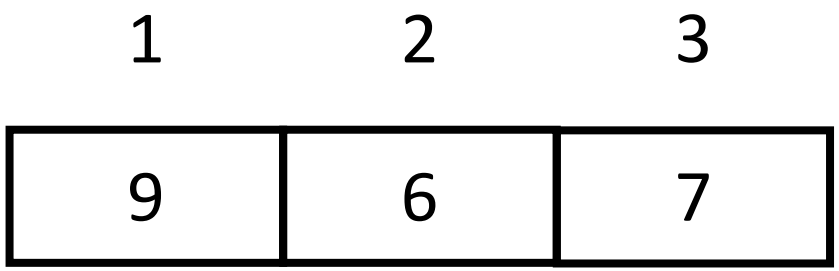
```
# Binary tree: (x[0], x[1], x[2], ...)
# 0-indexed array

def parent(i):
    return (i - 1) // 2

def left_child(i):
    return 2 * i + 1

def right_child(i):
    return 2 * i + 2
```

Двоичная куча – Binary heap

- «Куча на максимум» – Max-куча – max-heap
- Дерево



Массив



- Основное свойство кучи – Heap property:
- Значение потомка не превосходит значения предка
- $x[\,parent(i)\,] \geq x[\,i\,]$

Операции над кучей

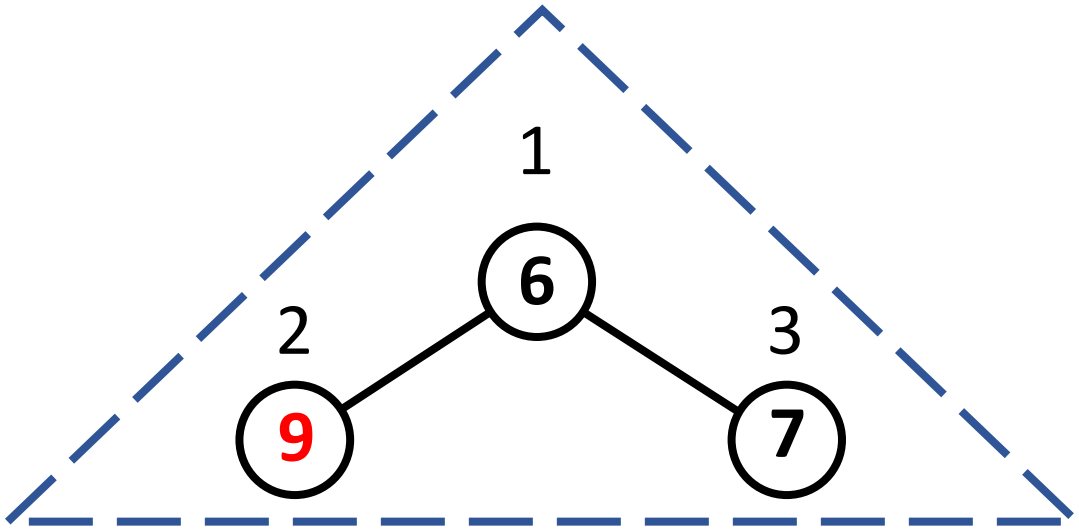
- heapify
  - поддерживает основное свойство кучи
- build_heap
  - строит кучу из неотсортированного массива
- heapsort
  - сортирует массив с помощью двоичной кучи

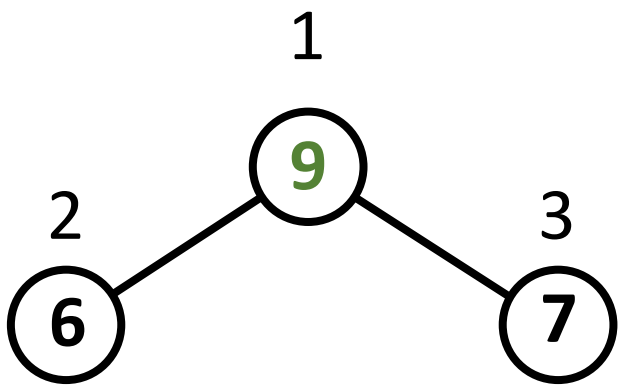
heapify(i , heap_size)

- Сохранение основного свойства кучи
  - x: массив
  - i: индекс вершины поддерева
  - n: длина входного массива
  - heap_size: размер кучи
- Если для вершины x[i] не выполняется основное свойство, переставить значение x[i] с бóльшим из детей – почему?
- Рекурсивно «погружаем» значение в нужное место – «пузырек»

Восстановление свойства кучи

x = [6, 9, 7] ➔ x = [9, 6, 7]





Рисуем и реализуем

```
x = [2,9,8,3,7,2,5]
heap_size = len(x)
print(*x, "<--")
heapify(0, heap_size)
print(*x, "-->")
```

```
C:\1>python heap_heapify.py
2 9 8 3 7 2 5 <--
9 2 8 3 7 2 5
9 7 8 3 2 2 5
9 7 8 3 2 2 5 -->
```

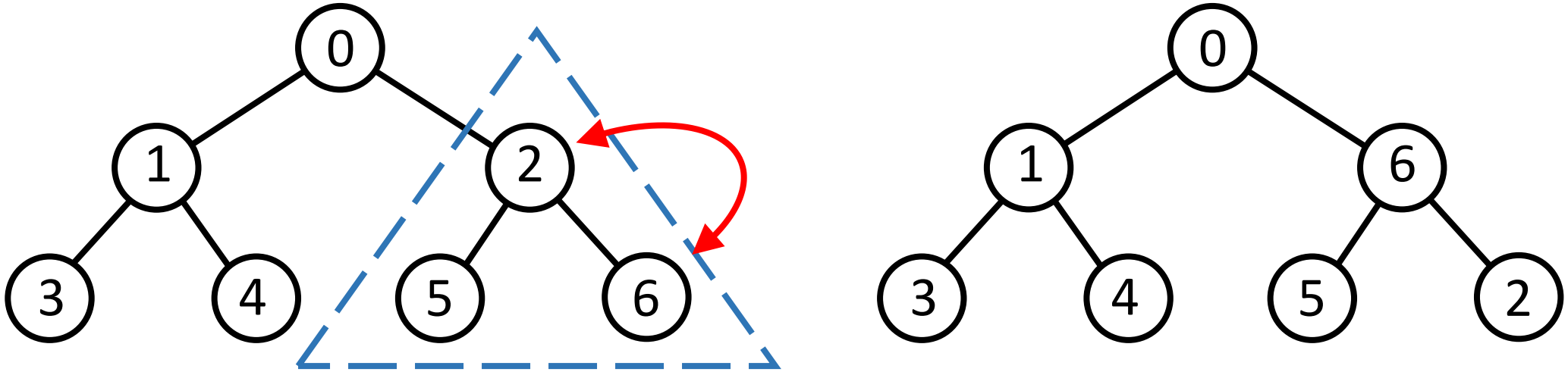
```
1  def heapify(i, heap_size):
2      left = 2 * i + 1
3      right = 2 * i + 2
4      largest = i
5
6      if left < heap_size and x[left] > x[largest]:
7          largest = left
8
9      if right < heap_size and x[right] > x[largest]:
10         largest = right
11
12     if largest != i:
13         x[i], x[largest] = x[largest], x[i]
14         print(*x)
15         heapify(largest, heap_size)
```

build_heap()

- Построение кучи из массива
- x: исходный неотсортированный массив
- Применяем heapify к вершинам, начиная с нижних – почему?
- С какого индекса начинаются листья для корня в 0 и в 1?

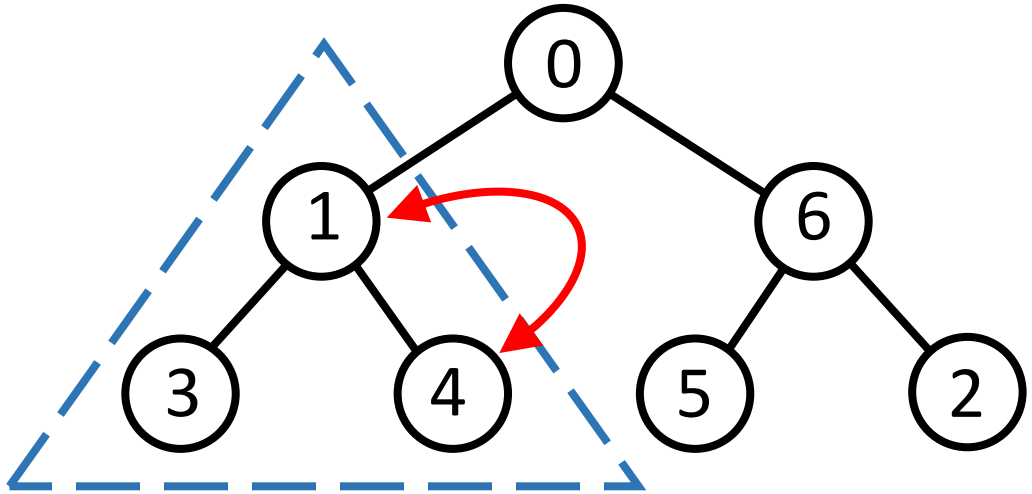
Моделируем шаг 1: i = 2

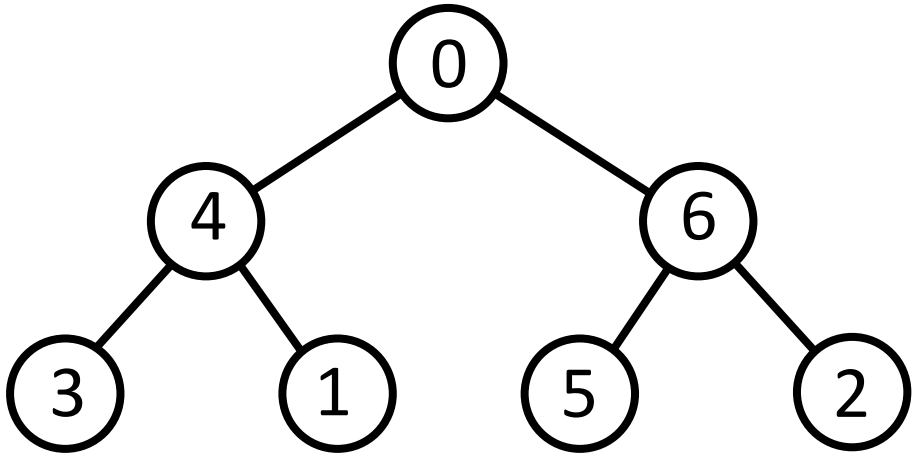
[0, 1, 2, 3, 4, 5, 6] ➔ [0, 1, 6, 3, 4, 5, 2]



Моделируем шаг 2: i = 1

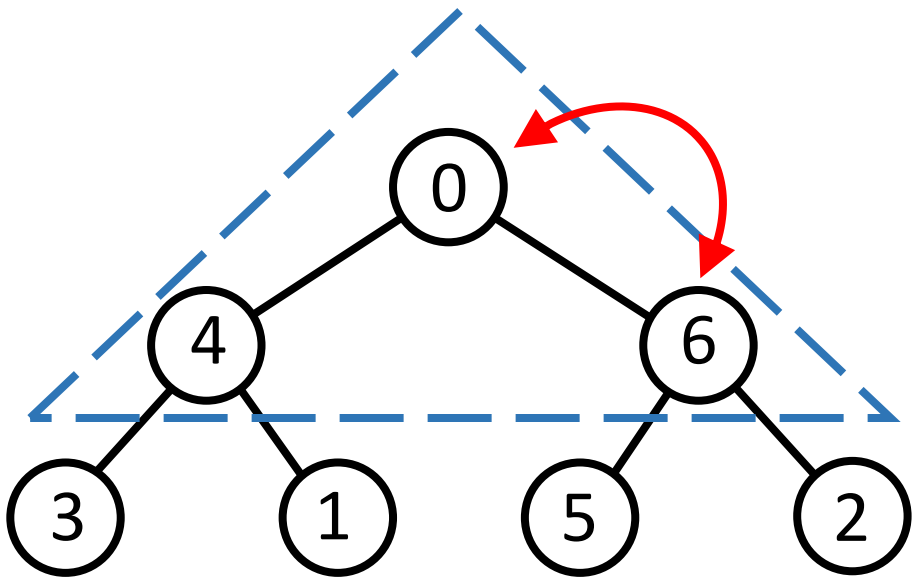
[0, 1, 6, 3, 4, 5, 2] ➔ [0, 4, 6, 3, 1, 5, 2]





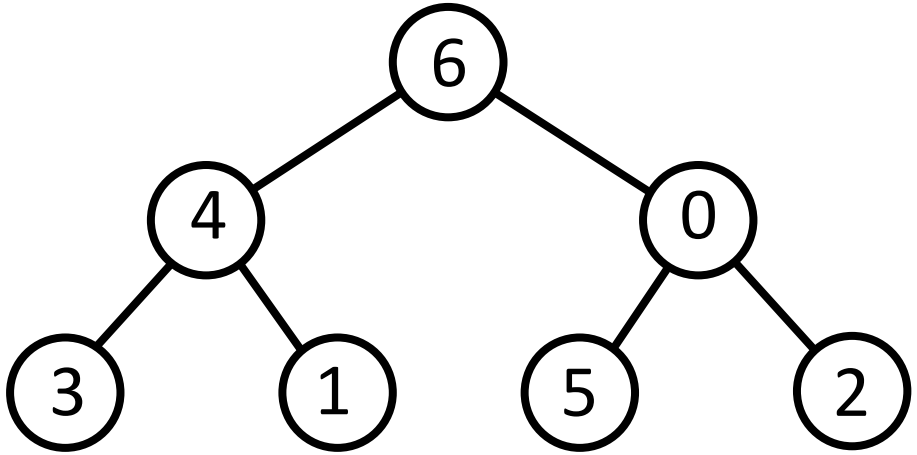
Моделируем шаг 3: i = 0

[0, 4, 6, 3, 1, 5, 2]



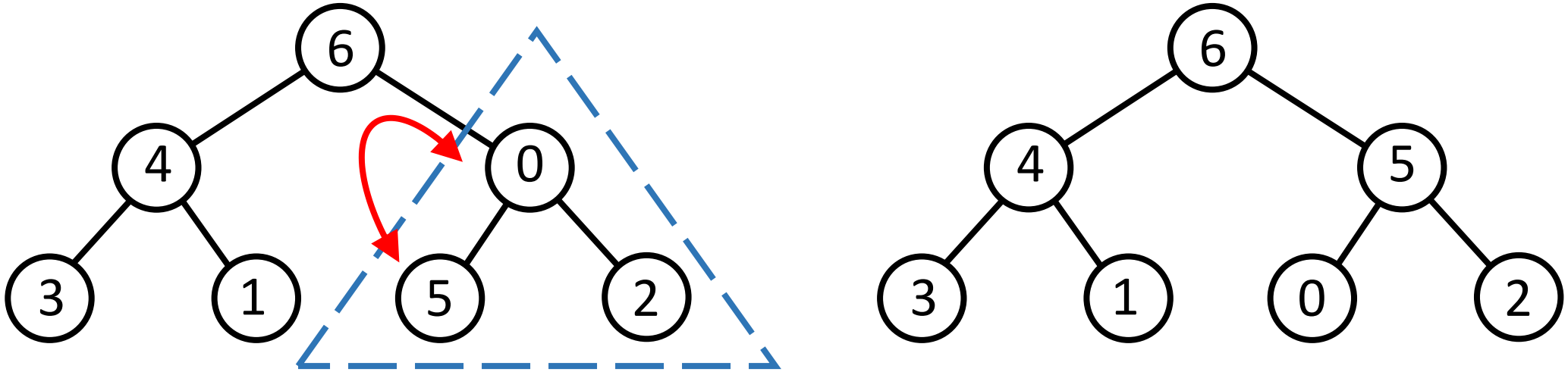
➔

[6, 4, 0, 3, 1, 5, 2]



Моделируем шаг 4: i = 2

[0, 4, 6, 3, 1, 5, 2] ➔ [6, 4, 5, 3, 1, 0, 2]



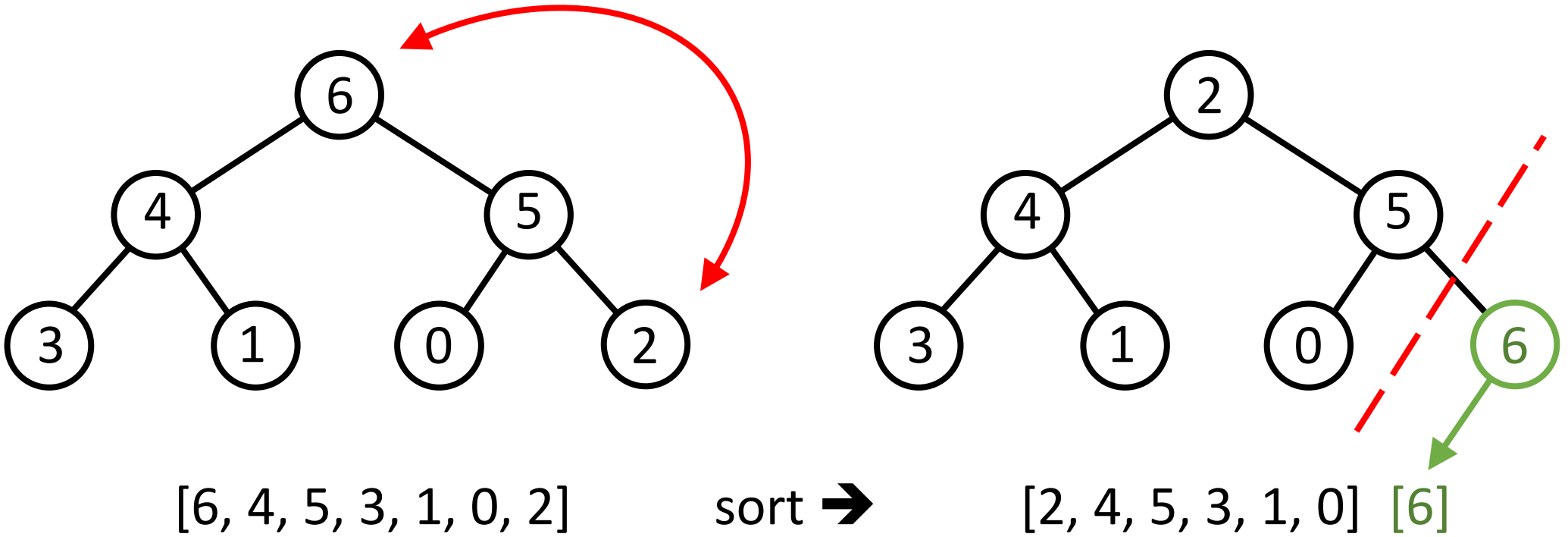
```python
def build_heap():
    heap_size = len(x)
    for i in range ( (heap_size - 1) // 2, -1, -1 ):
        heapify(i, heap_size)

x = list(range(7))
print(*x, "<--")
build_heap()
print(*x, "-->")
```

```
C:\1>python heap_build.py
0 1 2 3 4 5 6 <--
0 1 6 3 4 5 2
0 4 6 3 1 5 2
6 4 0 3 1 5 2
6 4 5 3 1 0 2
6 4 5 3 1 0 2 -->
```

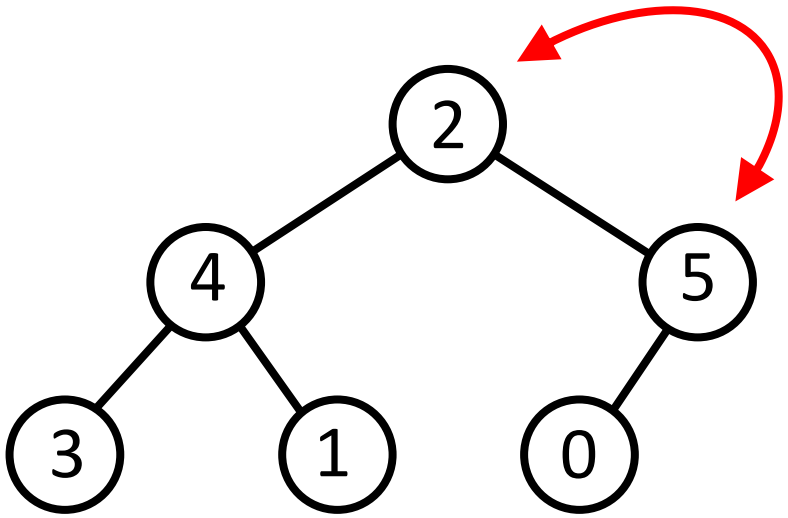
heapsort()

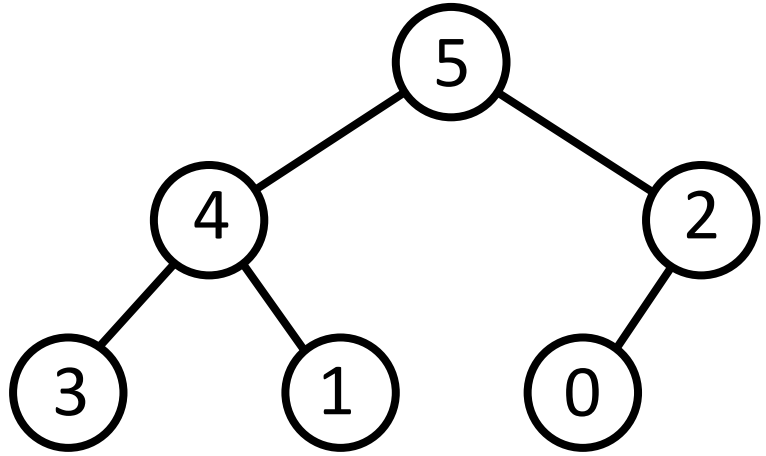
- Сортировка с помощью кучи
- build_heap: Построение кучи из исходного массива
- x[0]: максимальный элемент – обмен значений с концом массива
- x[i]: конец неотсортированного массива
- heap_size – 1: уменьшаем размер кучи
- heapify: восстановить основное свойство кучи

(1) Сортировка: heap_size = 7 → 6



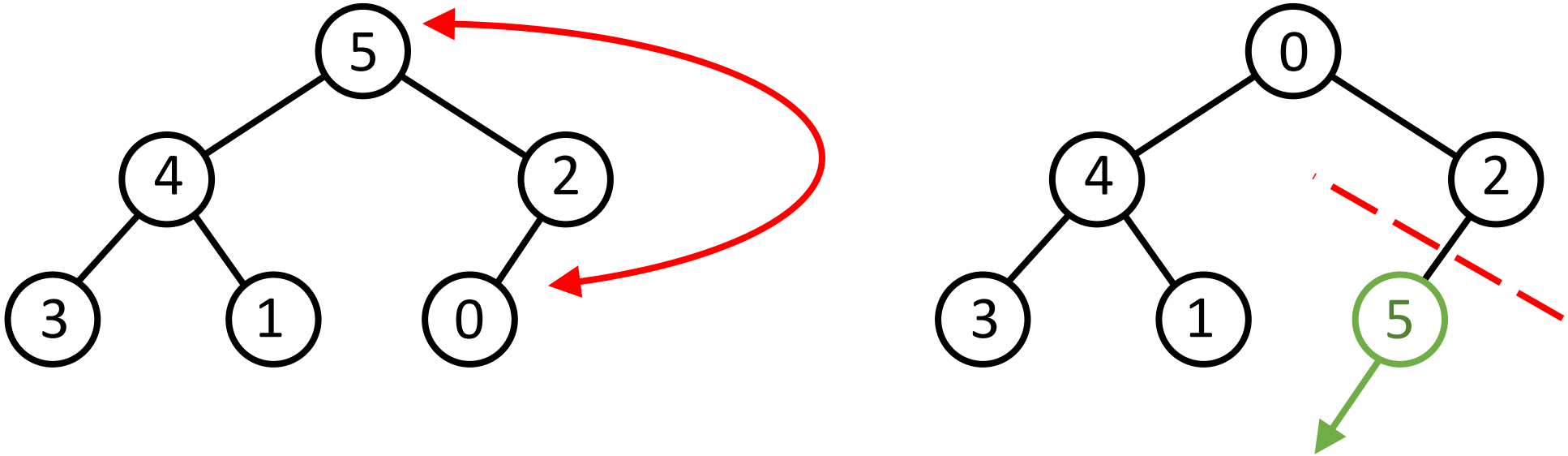
(2) Восстанавливаем кучу: heap_size = 6





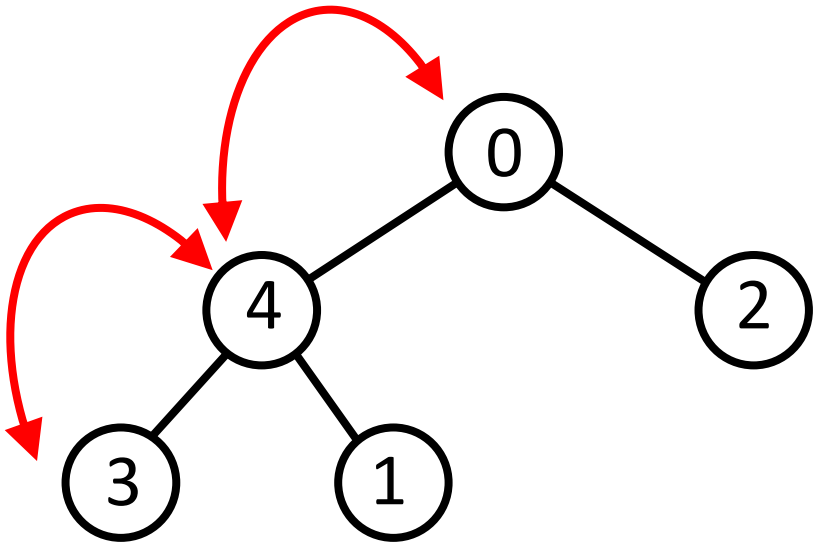
[2, 4, 5, 3, 1, 0] [6] heapify ➔ [5, 4, 2, 3, 1, 0] [6]

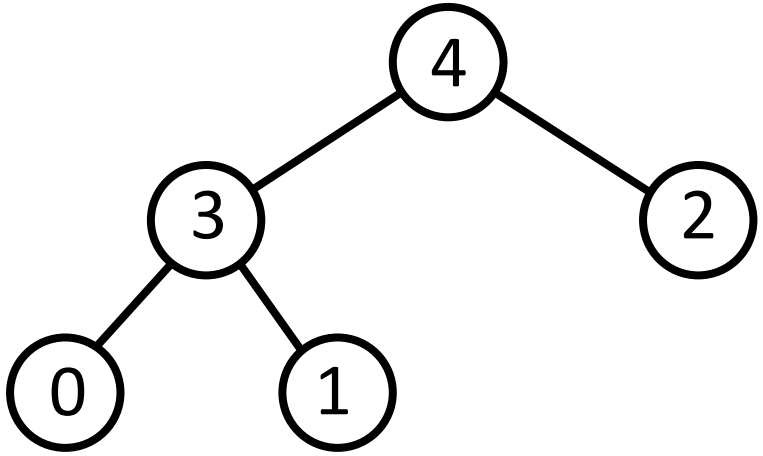
(3) Сортировка: heap_size = 6 → 5



[5, 4, 2, 3, 1, 0] [6] sort → [0, 4, 2, 3, 1] [5, 6]

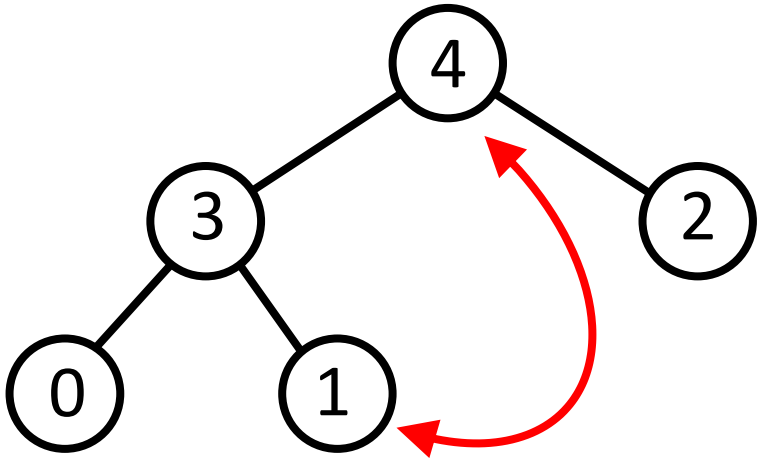
(4) Восстанавливаем кучу: heap_size = 5

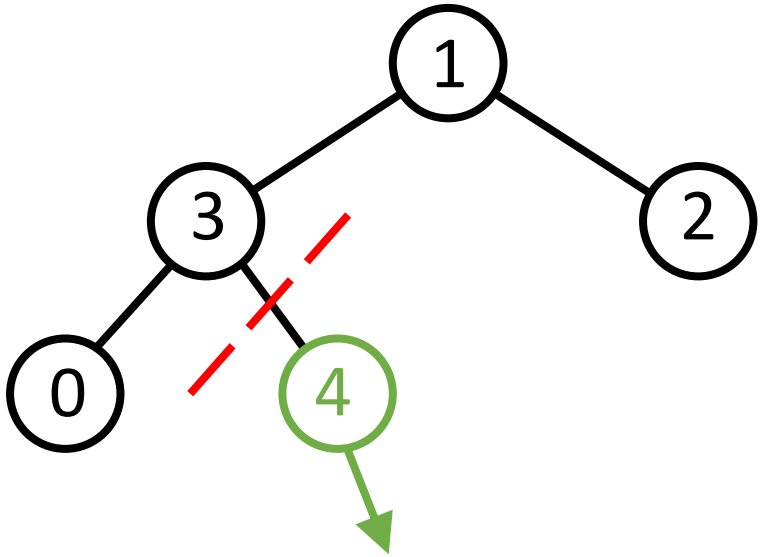




[0, 4, 2, 3, 1] [5, 6] heapify ➔ [4, 3, 2, 0, 1] [5, 6]

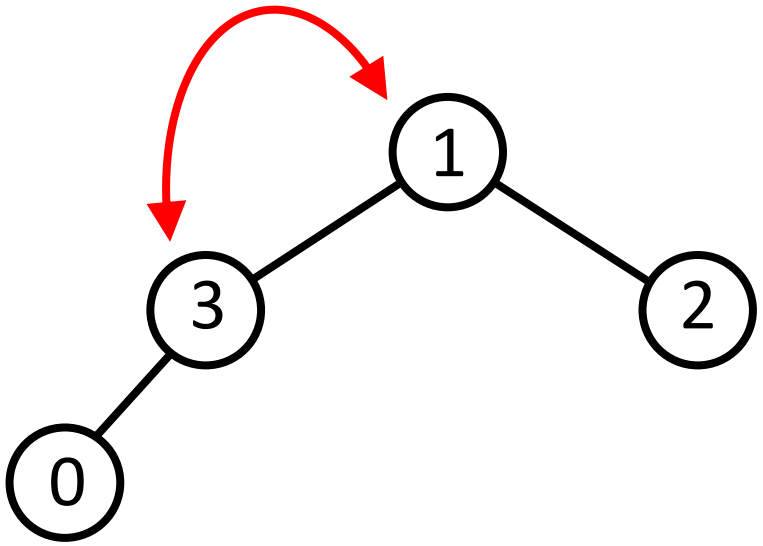
(5) Сортировка: heap_size = 5 → 4

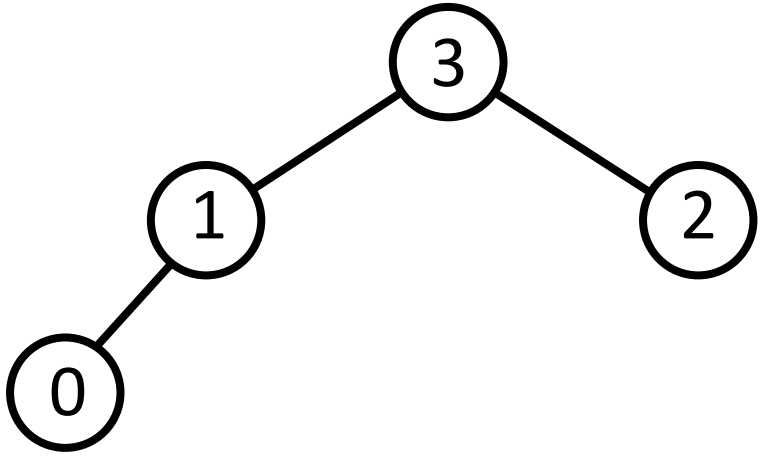




[5, 4, 2, 3, 1, 0] [5, 6] sort → [1, 3, 2, 0] [4, 5, 6]

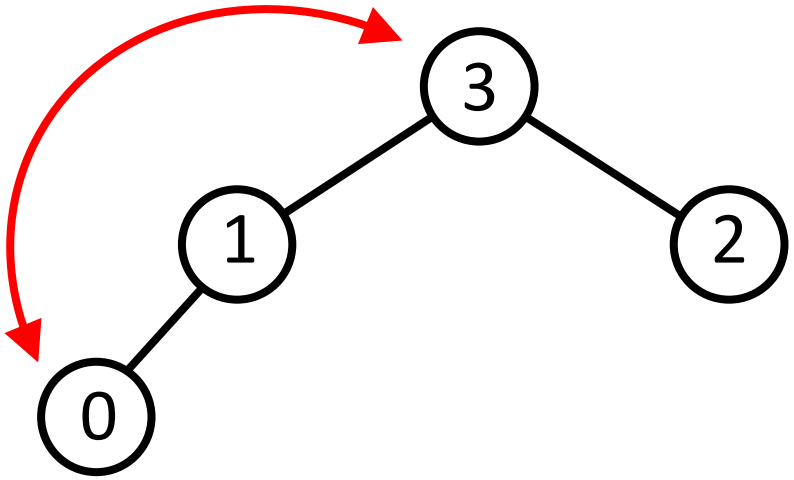
(6) Восстанавливаем кучу: heap_size = 4

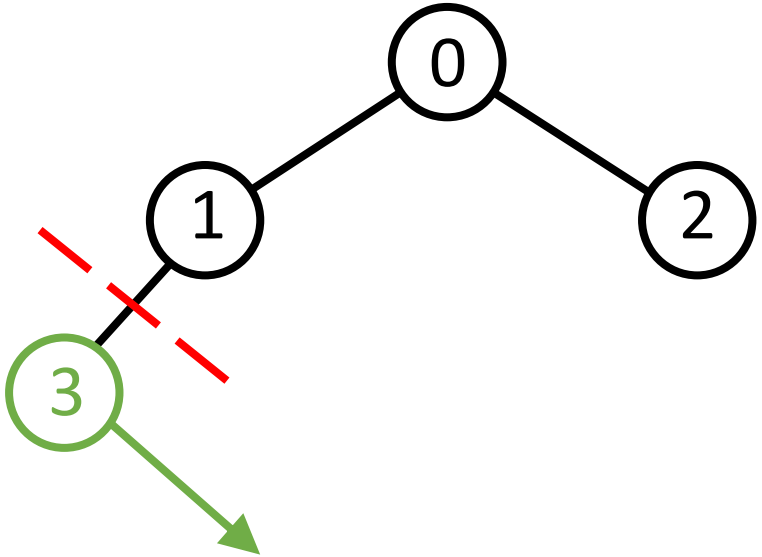




[1, 3, 2, 0] [4, 5, 6] heapify ➔ [3, 1, 2, 0] [4, 5, 6]

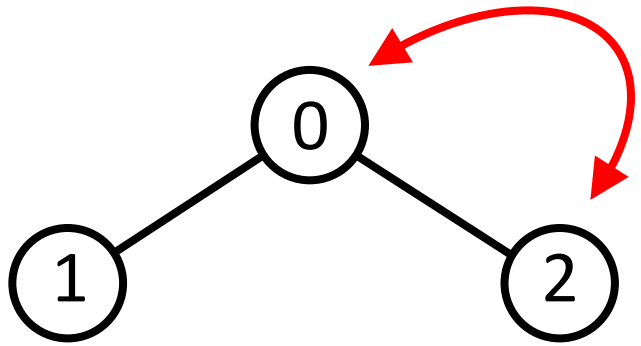
(7) Сортировка: heap_size = 4 → 3

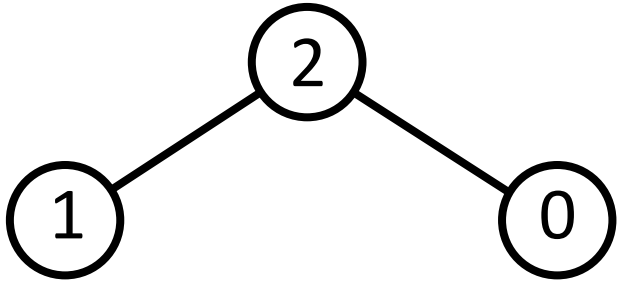




[3, 1, 2, 1] [4, 5, 6] sort → [0, 1, 2] [3, 4, 5, 6]

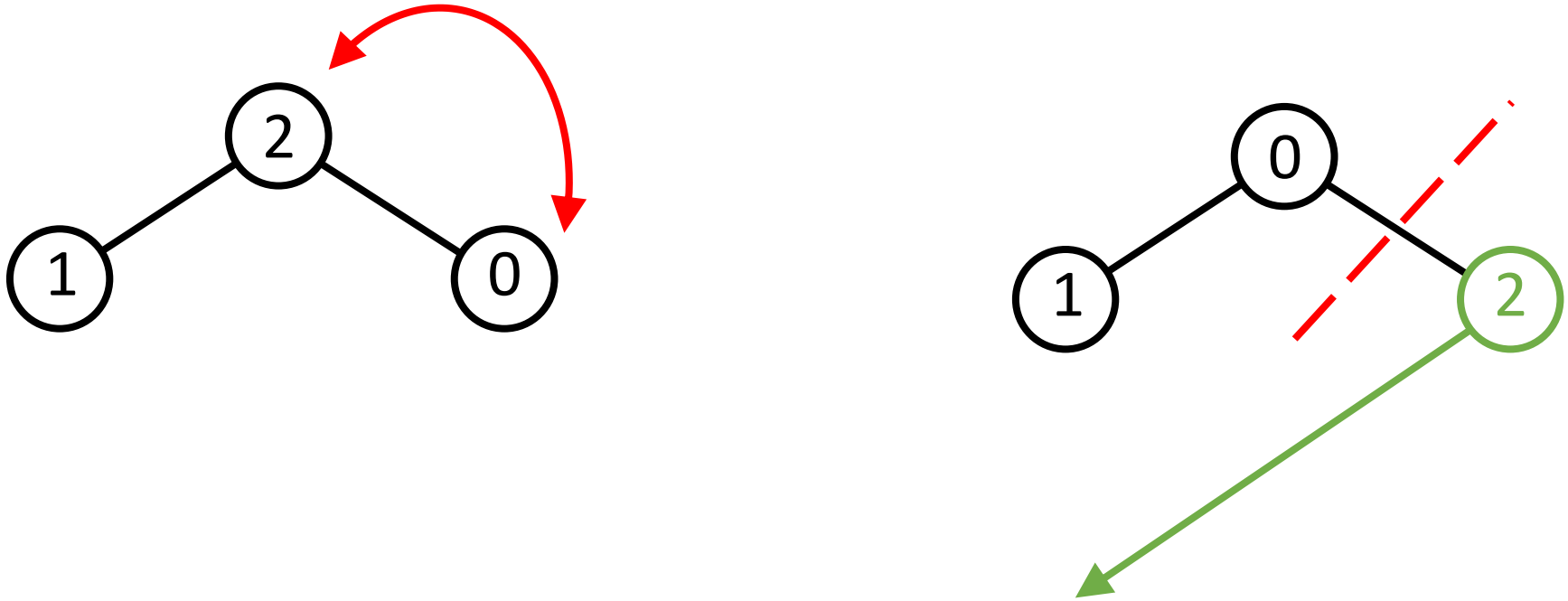
(8) Восстанавливаем кучу: heap_size = 3





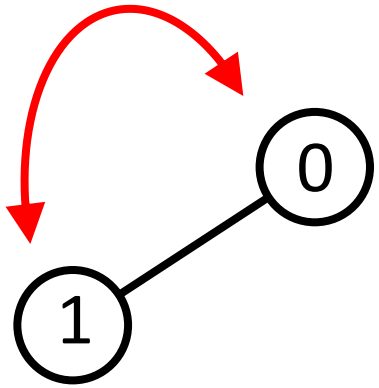
[0, 1, 2] [3, 4, 5, 6] heapify ➔ [2, 1, 0] [3, 4, 5, 6]

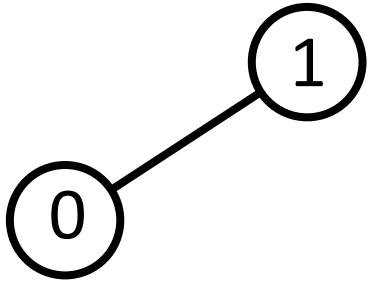
(9) Сортировка: heap_size = 3 → 2



[2, 1, 0] [3, 4, 5, 6] sort → [0, 1] [2, 3, 4, 5, 6]

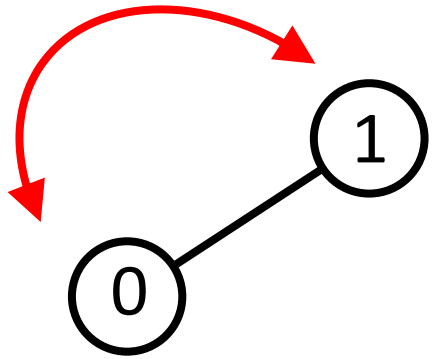
(10) Восстанавливаем кучу: heap_size = 2

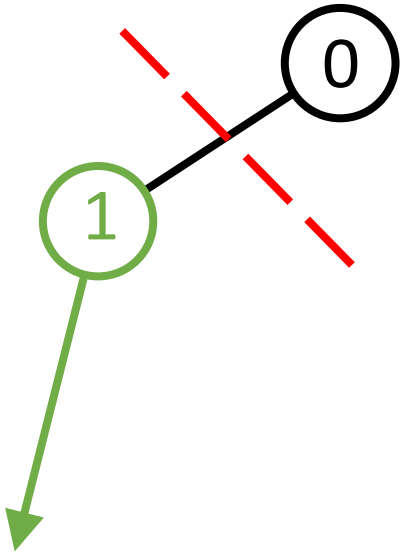




[0, 1] [2, 3, 4, 5, 6] heapify ➔ [1, 0] [2, 3, 4, 5, 6]

(11) Сортировка: heap_size = 2 → 1





[1, 0] [2, 3, 4, 5, 6] sort → [0] [1, 2, 3, 4, 5, 6]

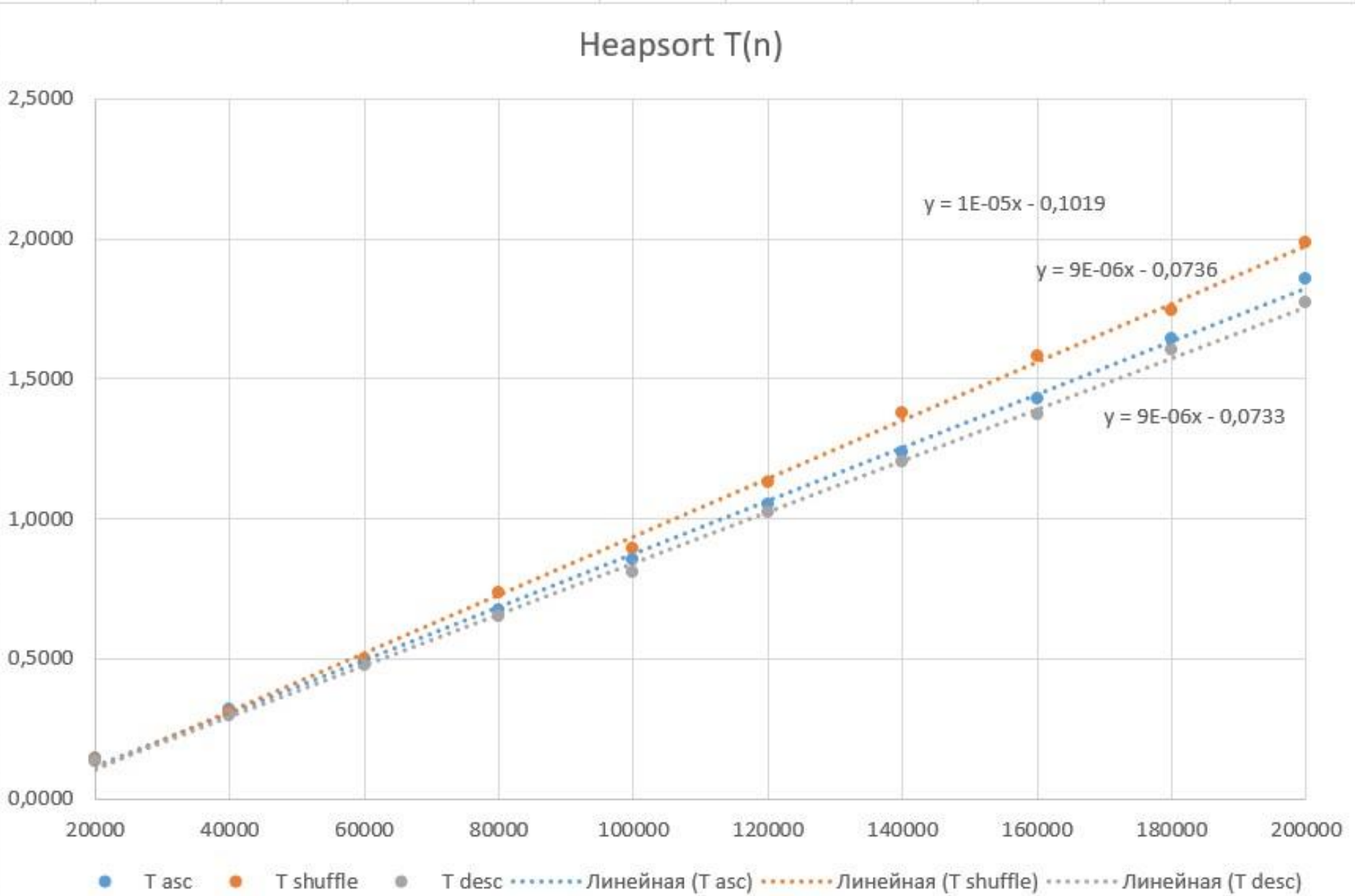
```
22  def heapsort():
23      build_heap()
24      heap_size = len(x)
25      for i in range (len(x) - 1, 0, -1):
26          x[0], x[i] = x[i], x[0]
27          print(f"i: {i} h: {heap_size} x: {x}")
28          heap_size -= 1
29          heapify(0, heap_size)
30
31  x = [2, 9, 8, 3, 7, 2, 5]
32  print(*x, "<--")
33  heapsort()
34  print(*x, "-->")
```

```
2 9 8 3 7 2 5 <--
i: 6 h: 7 x: [5, 7, 8, 3, 2, 2, 9]
i: 5 h: 6 x: [2, 7, 5, 3, 2, 8, 9]
i: 4 h: 5 x: [2, 3, 5, 2, 7, 8, 9]
i: 3 h: 4 x: [2, 3, 2, 5, 7, 8, 9]
i: 2 h: 3 x: [2, 2, 3, 5, 7, 8, 9]
i: 1 h: 2 x: [2, 2, 3, 5, 7, 8, 9]
2 2 3 5 7 8 9 -->
```

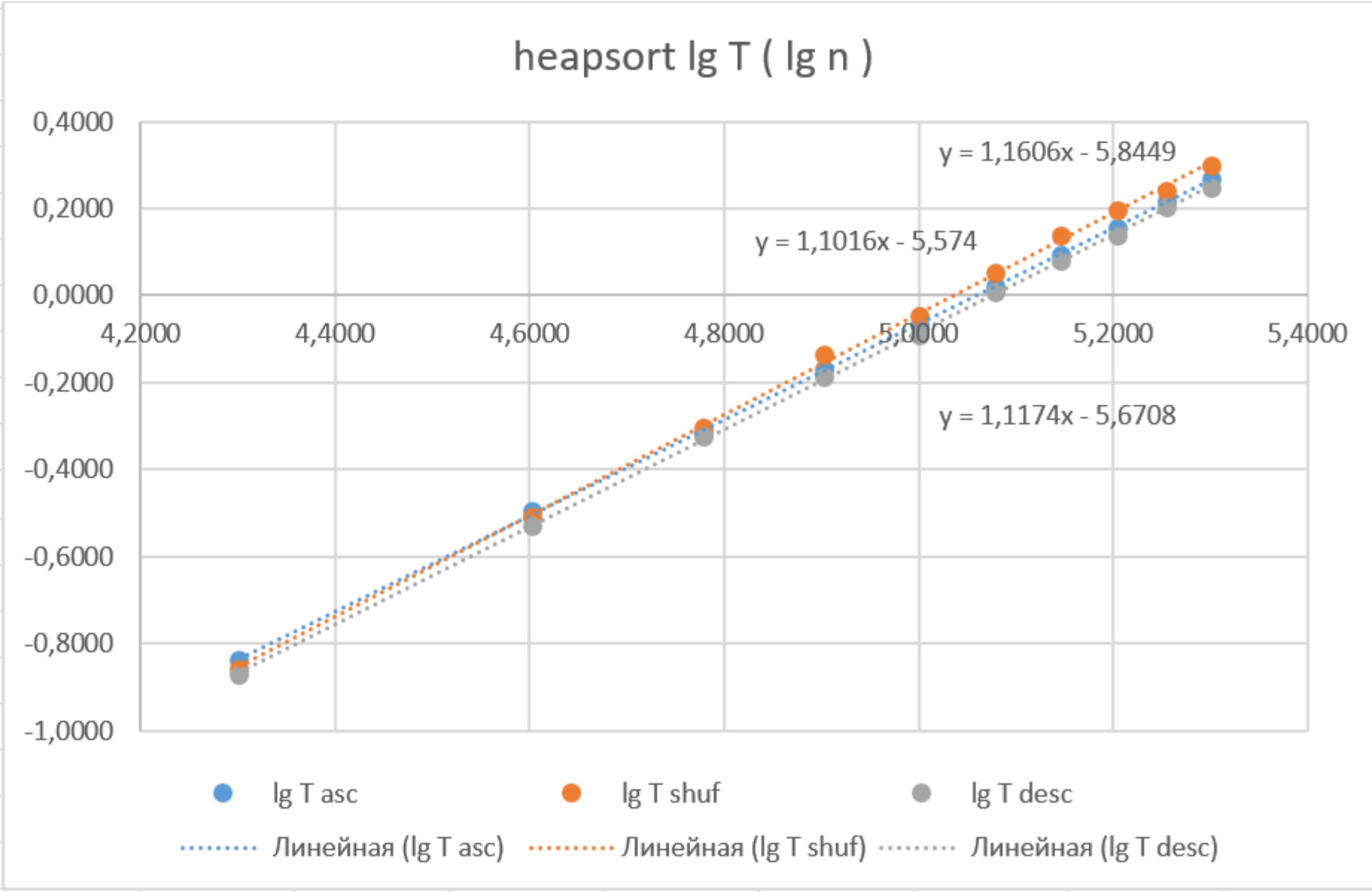
Подбираем сложность задачи

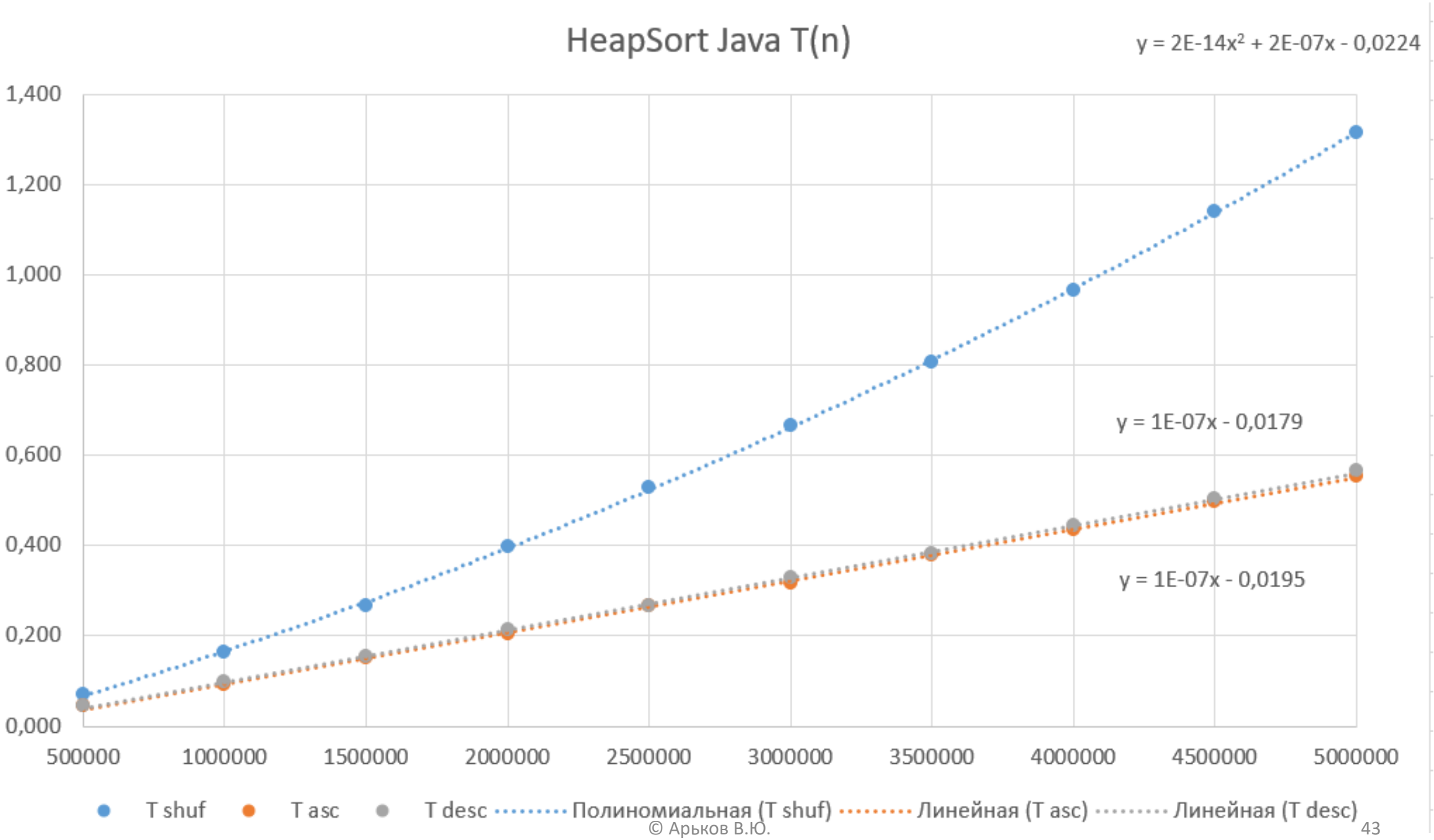
```
30 import random
31 n = 200000
32 x = list(range(n))
33 random.shuffle(x)
34 import time
35 start = time.time()
36 heapsort()
37 end = time.time()
38 tn = end - start
39 print(f"{n}; {tn:.10f}".replace(".", ","))
40
41 #  20000; 0,1334862709
42 # 200000; 1,9561014175
```

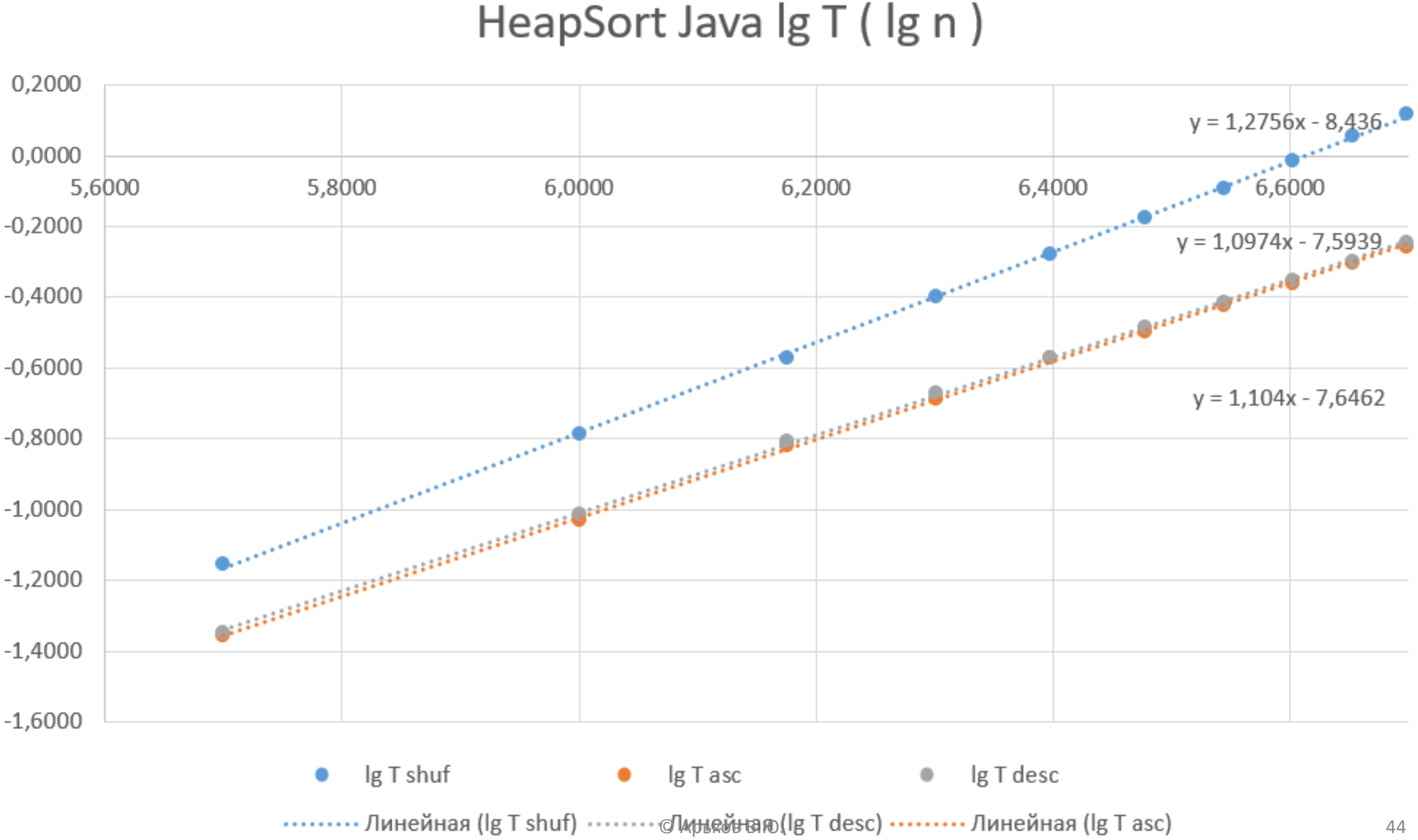
n	T asc	T shuffle	T desc
20000	0,1450	0,1390	0,1350
40000	0,3190	0,3100	0,2950
60000	0,4820	0,4990	0,4760
80000	0,6770	0,7350	0,6510
100000	0,8570	0,8960	0,8110
120000	1,0510	1,1290	1,0220
140000	1,2380	1,3810	1,2040
160000	1,4310	1,5790	1,3720
180000	1,6440	1,7440	1,6020
200000	1,8580	1,9890	1,7700

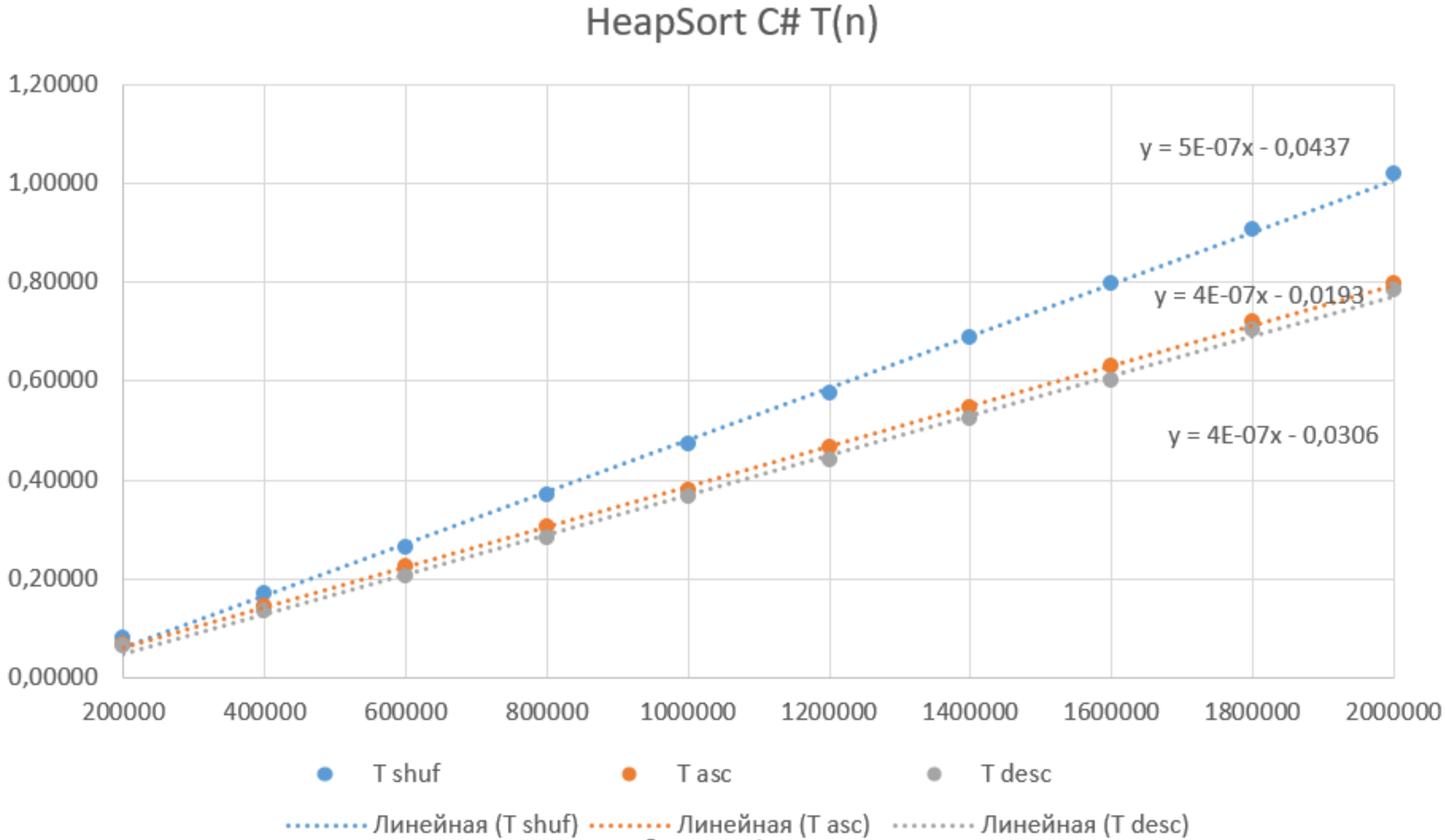


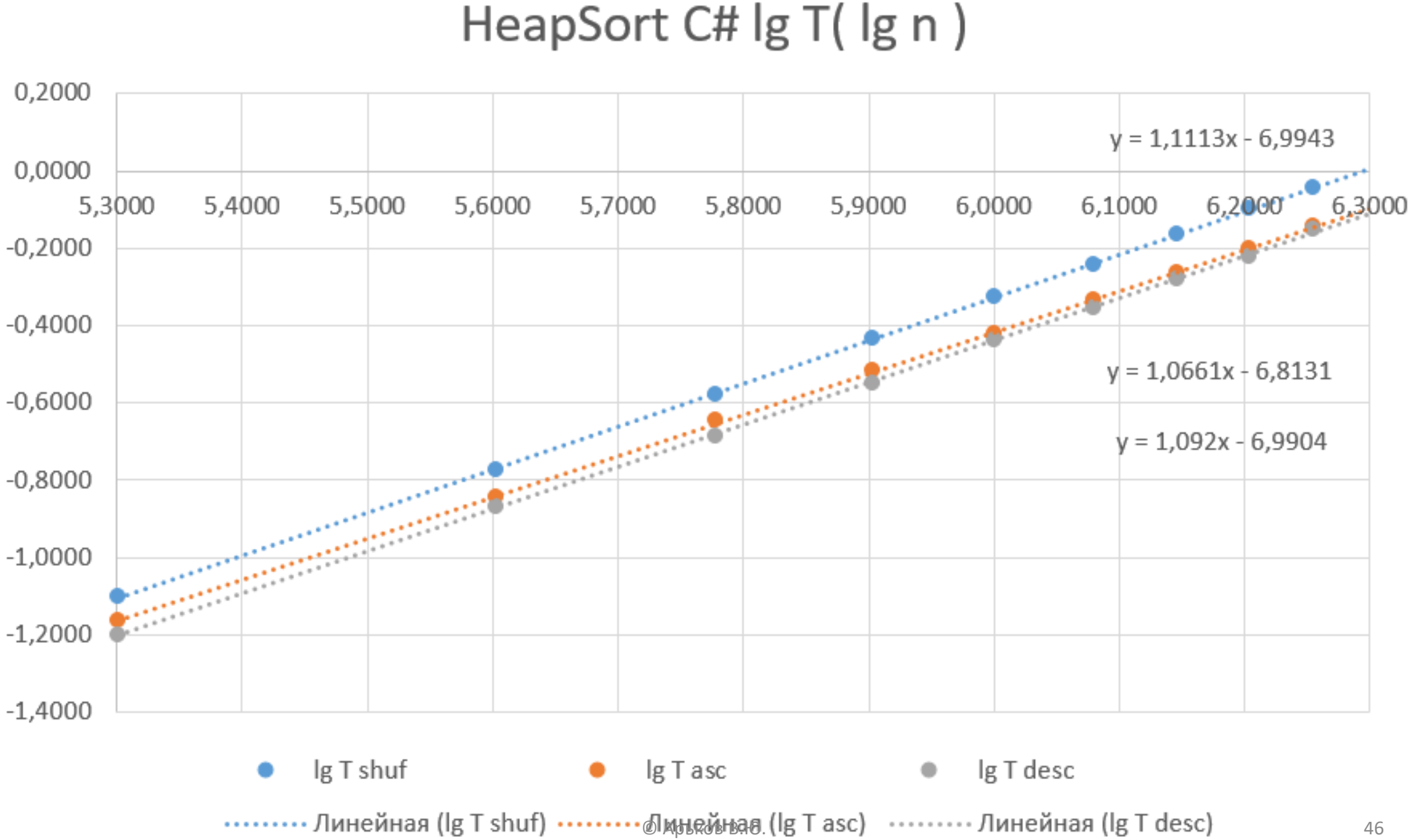
lg n	lg T asc	lg T shuf	lg T desc
4,3010	-0,8386	-0,8570	-0,8697
4,6021	-0,4962	-0,5086	-0,5302
4,7782	-0,3170	-0,3019	-0,3224
4,9031	-0,1694	-0,1337	-0,1864
5,0000	-0,0670	-0,0477	-0,0910
5,0792	0,0216	0,0527	0,0094
5,1461	0,0927	0,1402	0,0806
5,2041	0,1556	0,1984	0,1374
5,2553	0,2159	0,2415	0,2047
5,3010	0,2690	0,2986	0,2480




HeapSort Java T(n)
y = 2E-14x² + 2E-07x - 0,0224
y = 1E-07x - 0,0179
y = 1E-07x - 0,0195
1,400
1,200
1,000
0,800
0,600
0,400
0,200
0,000
500000
1000000
1500000
2000000
2500000
3000000
3500000
4000000
4500000
5000000
T shuf
T asc
T desc
Полиномиальная (T shuf)
Линейная (T asc)
Линейная (T desc)


HeapSort Java lg T (lg n)
0,2000
0,0000
-0,2000
-0,4000
-0,6000
-0,8000
-1,0000
-1,2000
-1,4000
-1,6000
5,6000
5,8000
6,0000
6,2000
6,4000
6,6000
y = 1,2756x - 8,436
y = 1,0974x - 7,5939
y = 1,104x - 7,6462
lg T shuf
lg T asc
lg T desc
Линейная (lg T shuf)
Линейная (lg T desc)
Линейная (lg T asc)


HeapSort C# T(n)
1,20000
1,00000
0,80000
0,60000
0,40000
0,20000
0,00000
200000
400000
600000
800000
1000000
1200000
1400000
1600000
1800000
2000000
y = 5E-07x - 0,0437
y = 4E-07x - 0,0193
y = 4E-07x - 0,0306
T shuf
T asc
T desc
Линейная (T shuf)
Линейная (T asc)
Линейная (T desc)


HeapSort C# lg T(lg n)
0,2000
0,0000
-0,2000
-0,4000
-0,6000
-0,8000
-1,0000
-1,2000
-1,4000
5,3000
5,4000
5,5000
5,6000
5,7000
5,8000
5,9000
6,0000
6,1000
6,2000
6,3000
y = 1,1113x - 6,9943
y = 1,0661x - 6,8131
y = 1,092x - 6,9904
lg T shuf
lg T asc
lg T desc
Линейная (lg T shuf)
Линейная (lg T asc)
Линейная (lg T desc)