к

. .
( )ь
(. . .)

к ь

к ь

3

" "

. .

7-56

() .

. ., . .

— 2022 .

. () .
(, , , , ..) . , .
. :

1) , ;

2) , ;

3) , , .

: : , , .
:

– 3 : , , ;

– ;

– ;

– .

1

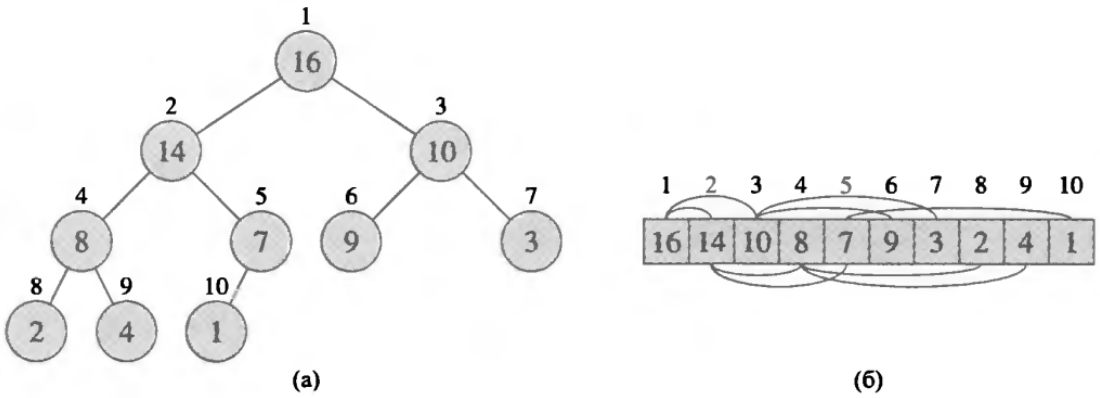
— , , .

1.1

(BubbleSort) [1] — , . , , . $N-1$, , , , ( ). , .

1.2

(HeapSort) [2] — , (heap).
() — , -, , . 1.1. , , , , , .



1.1 – () ()

: .

(max-heap)
$A[Parent(i)] \geqslant A[i]$], .. . , .

(min-heap) $A[Parent(i)] \leqslant A[i]$. , .

, , . i, $2i+1$, $2i+2$.

:

1) ;

2) ( ) m, m — ;

3) $m-1$;

4) 2, 3 - $m-1$, - 0.

, , , .

heapify , .

.. , [(n/2 + 1)...n] (n -) , (-) . [(n/4 + 1)...n/2], , , -.

heapify $i \in [0...n/2]$ A[i], A[Left(i)], A[Right(i)], A - , Left(i) Right(i) - , largest. A[i], A[i] A[largest] . , i . A[i] largest, largest . , heapify .

1.3

(Bucket sort) [3] — , , ; O(n).

[a, b] (a b -) n , (buckets), n . ( ) [a, b], , . , , .

, . , .

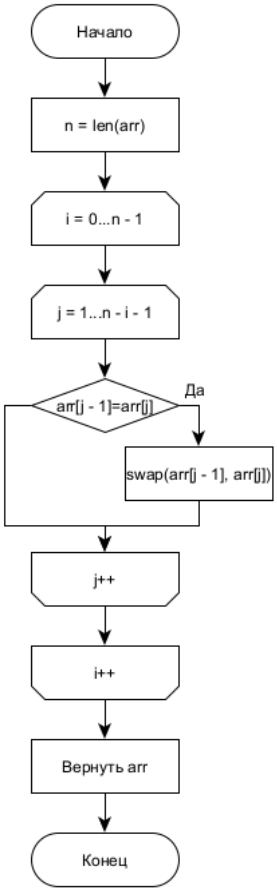
— , , .

2

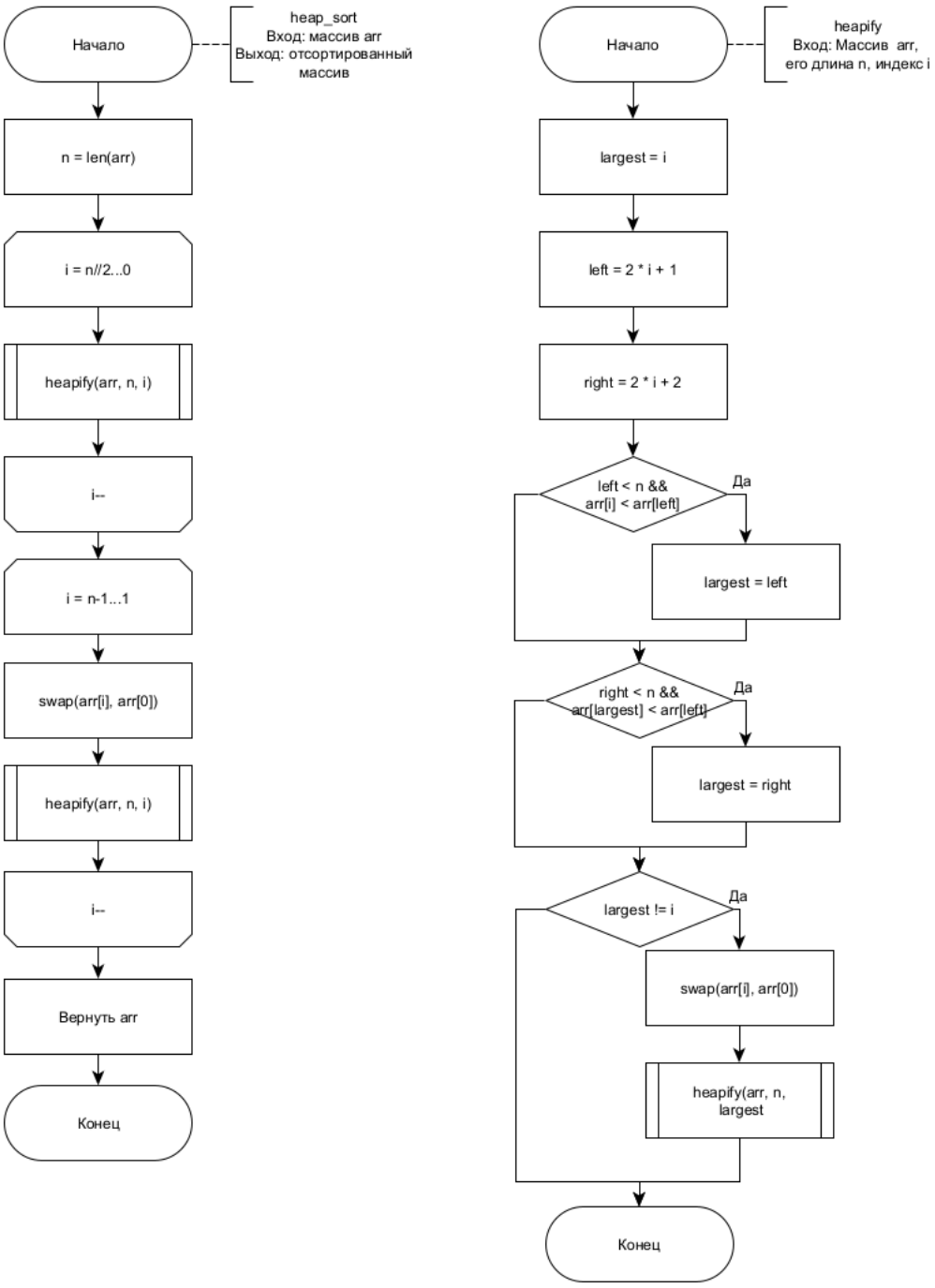
, .

2.1

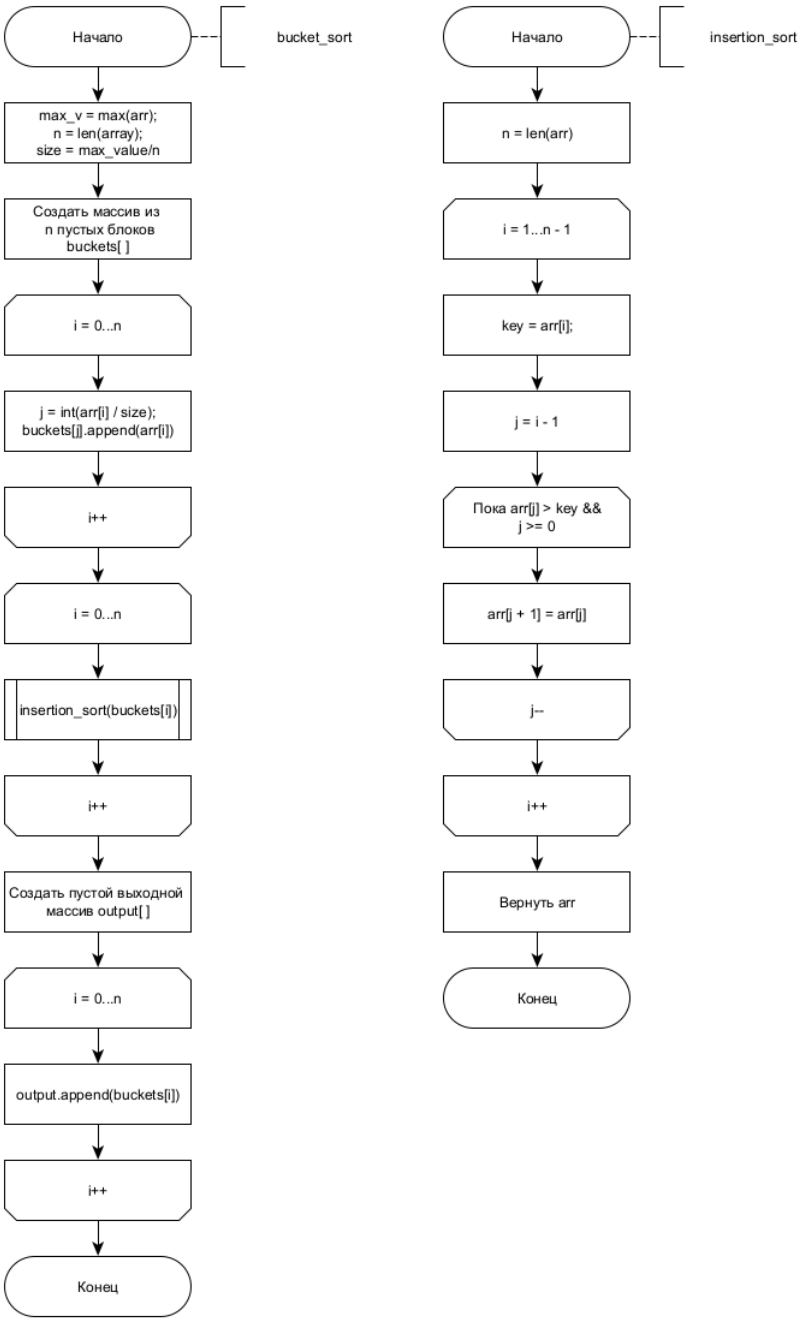
: (2.1), (2.2), (2.3).



2.1 –


Начало
heap_sort
Вход: массив arr
Выход: отсортированный массив
n = len(arr)
i = n//2...0
heapify(arr, n, i)
i--
i = n-1...1
swap(arr[i], arr[0])
heapify(arr, n, i)
i--
Вернуть arr
Конец
Начало
heapify
Вход: Массив arr, его длина n, индекс i
largest = i
left = 2 * i + 1
right = 2 * i + 2
left < n && arr[i] < arr[left]
Да
largest = left
right < n && arr[largest] < arr[left]
Да
largest = right
largest != i
Да
swap(arr[i], arr[0])
heapify(arr, n, largest
Конец

2.2 – () c heapify ()


Начало
bucket_sort
max_v = max(arr);
n = len(array);
size = max_value/n
Создать массив из
n пустых блоков
buckets[ ]
i = 0...n
j = int(arr[i] / size);
buckets[j].append(arr[i])
i++
i = 0...n
insertion_sort(buckets[i])
i++
Создать пустой выходной
массив output[ ]
i = 0...n
output.append(buckets[i])
i++
Конец
Начало
insertion_sort
n = len(arr)
i = 1...n - 1
key = arr[i];
j = i - 1
Пока arr[j] > key &&
j >= 0
arr[j + 1] = arr[j]
j--
i++
Вернуть arr
Конец

2.3 – () , ()

2.2

[4]:

1) (2.1) 1.

$$+, -, /, \%, ==, != , <, >, <=, >=, [], ++, -- \tag{2.1}$$

2) if then A else B , (2.2).

$$f_{if} = f + \begin{cases} f_A, & , \\ f_B, & . \end{cases} \tag{2.2}$$

3) , (2.3).

$$f_{for} = f + f + N(f + f + f) \tag{2.3}$$

4) 0.

2.3

N.

2.3.1

:

– $i \in [1..N)$ (2.4).

$$f_i = 2 + 2(N - 1) \tag{2.4}$$

– , $[1..N - 1]$, (2.5).

$$f_j = 3(N - 1) + \frac{N \cdot (N - 1)}{2} \cdot (3 + f_{if}) \tag{2.5}$$

– (2.6).

$$f_{if} = 4 + \begin{cases} 0, \\ 9, \end{cases} \tag{2.6}$$

(2.7).

$$f_{best} = \frac{7}{2}N^2 + \frac{3}{2}N - 3 \approx \frac{7}{2}N^2 = O(N^2) \tag{2.7}$$

(2.8).

$$f_{worst} = 8N^2 - 8N - 3 \approx 8N^2 = O(N^2) \tag{2.8}$$

2.3.2

, heapify, .

, n- $\lfloor \lg n \rfloor$. heapify n $\lfloor \lg n \rfloor$ . .

heapify (2.9).

$$f_{best} = 16 = O(1) \tag{2.9}$$

heapify (2.10).

$$f_{worst} = 16 + 9 + 25\lfloor \lg n \rfloor \approx 25\lfloor \lg n \rfloor = O(\lfloor \lg n \rfloor) \tag{2.10}$$

:

– $i \in [N/2...0]$ (2.11).

$$f_i = 3 + 2(N/2) \tag{2.11}$$

– (heapify)
$j \in [N-1...0)$ (2.12).

$$f_j = 3 + 2(N-1) \tag{2.12}$$

– heapify $i \in [N/2...0]$ $j \in [N-1...0)$ (2.13).

$$f_{summ_{heapify}} = 16 + (N/2 + N - 1) f_{heapify}, \tag{2.13}$$

$f_{heapify}$ (2.14).

$$f_{heapify} = \begin{cases} 16, \\ 25 + 25\lfloor \lg n \rfloor, \end{cases} \tag{2.14}$$

(2.15).

$$f_{best} = 22 + 16(N/2 + N - 1) + 2(N - 1 + N/2) \approx 27N = O(N) \tag{2.15}$$

(2.16).

$$\begin{aligned} f_{worst} &= 22 + (N/2 + N - 1)(25 + 25\lfloor \lg n \rfloor) + 2(N - 1 + N/2) \approx \\ &\approx \frac{75}{2} N \lfloor \lg n \rfloor = O(N \lg n) \end{aligned} \tag{2.16}$$

2.3.3

, , .

.

– $i \in [1..N)$ ( )(2.17):

$$f_i = 2 + 2(N-1) + 7(N-1) \tag{2.17}$$

– , $[1..N-1]$ (2.18):

$$f_{while} = \begin{cases} (N-1) \cdot 4, \\ (N-1) \cdot 10(N-1), \end{cases} \tag{2.18}$$

(2.19).

$$f_{best} = 9N - 3 + 4N - 4 \approx 13N = O(N) \tag{2.19}$$

(2.20).

$$f_{worst} = 10N^2 - 20N - 10 + 9N - 3 \approx 10N^2 = O(N^2) \tag{2.20}$$

:

– (N)$i \in [0...N)$ (2.21).

$$f_1 = 2 + 2N + N \tag{2.21}$$

– ( )$i \in [0...N)$ (2.22).

$$f_2 = 2 + 2N + 5N \tag{2.22}$$

– ( )$i \in [0...N)$. , i- N, $n_i \leq N$, i- (2.22):

$$f_3 = 2 + 2N + \sum k = 0^{N-1} f_{ins_k} \tag{2.23}$$

, f_{ins_k} — k- , n_k.

, -, "— . , N , N, , 1 (O(1)). , N
. .

(2.24).

$$f_{best} = 13N + 6 + (13N - 7) \approx 26N = O(N) \tag{2.24}$$

(2.25).

$$f_{worst} = 13N + 6 + (10N^2 - 11N - 10 - 3) \approx 10N^2 = O(n^2) \tag{2.25}$$

, , .

3

, .

3.1

Python [5]. . .

3.2

3.1 — 3.6 — , , .

3.1 –

```
1 def bubble_sort(array):
2     n = len(array)
3     for i in range(n):
4         for j in range(1, n - i):
5             if array[j - 1] > array[j]:
6                 t = array[j-1]
7                 array[j-1] = array[j]
8                 array[j] = t
9     return array
```

3.2 –

```
def heap_sort(array):
    n = len(array)
    for i in range(n//2, -1, -1):
        heapify(array, n, i)
    for i in range(n-1, 0, -1):
        t = array[i]
        array[i] = array[0]
        array[0] = t
        heapify(array, i, 0)
    return array
```

3.3 – heapify

```
def heapify(array, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and array[i] < array[l]:
        largest = l
    if r < n and array[largest] < array[r]:
        largest = r
    if largest != i:
        t = array[i]
        array[i] = array[largest]
        array[largest] = t
        heapify(array, n, largest)
```

3.4 –

```
def bucket_sort(array):
    max_value = max(array)
    min_value = min(array)
    n = len(array)
    if n == 0:
        return array
    size = (max_value - min_value)/n
    if size == 0:
        return array

    buckets_list= []
    for i in range(n):
```

3.5 –

```
        buckets_list.append([])

    for i in range(n):
        j = int ((array[i] - min_value)/ size)
        if j != n:
            buckets_list[j].append(array[i])
        else:
            buckets_list[-1].append(array[i])

    for z in range(n):
        insertion_sort(buckets_list[z])

    final_output = []
    for x in range(n):
        final_output = final_output + buckets_list[x]
    return final_output
```

3.6 –

```
def insertion_sort(array):
    for i in range(1,len(array)):
        key = array[i]
        j = i-1
        while array[j] > key and j >= 0:
            array[j+1] = array[j]
            j -= 1
        array[j+1] = key
    return array
```

3.3

. 3.1 , . .

3.1 –

$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$
$[55, 45, 35, 25, 15]$	$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$
$[-10, -20, -30, -25]$	$[-30, -25, -20, -10]$	$[-30, -25, -20, -10]$
$[40, -10, 20, -30, 75]$	$[-30, -10, 20, 40, 75]$	$[-30, -10, 20, 40, 75]$
$[100]$	$[100]$	$[100]$
$[-20]$	$[-20]$	$[-20]$
$[\,]$	$[\,]$	$[\,]$

3.2 – ( )

100	0.25	0.2188	0.0781
200	0.9688	0.3438	0.1562
300	2.0312	0.75	0.2656
400	3.6719	1.0156	0.3281
500	5.875	1.3438	0.5469
600	8.5156	1.6094	0.5781
700	12.1562	1.8594	0.8125
800	15.7031	2.3438	1.1094

, . , .

4

, , .

4.1

, :

– Windows 10 21H1 [6] x86_64;

– 8 2133 ;

– Intel Core i5-8300H 2.30 [7].

.

4.2

process_time time Python. float [8].

— , .

, , . $[A_{min}, A_{max}]$, A_{min} A_{max} A. N , $N-1$ [0, 1], 1000000. $N-1$, . , .

4.1 — 4.4.

4.1 – ( )

100	0.25	0.2188	0.0781
200	0.9688	0.3438	0.1562
300	2.0312	0.75	0.2656
400	3.6719	1.0156	0.3281
500	5.875	1.3438	0.5469
600	8.5156	1.6094	0.5781
700	12.1562	1.8594	0.8125
800	15.7031	2.3438	1.1094

4.2 – ( )

100	0.25	0.2188	0.0781
200	0.9688	0.3438	0.1562
300	2.0312	0.75	0.2656
400	3.6719	1.0156	0.3281
500	5.875	1.3438	0.5469
600	8.5156	1.6094	0.5781
700	12.1562	1.8594	0.8125
800	15.7031	2.3438	1.1094

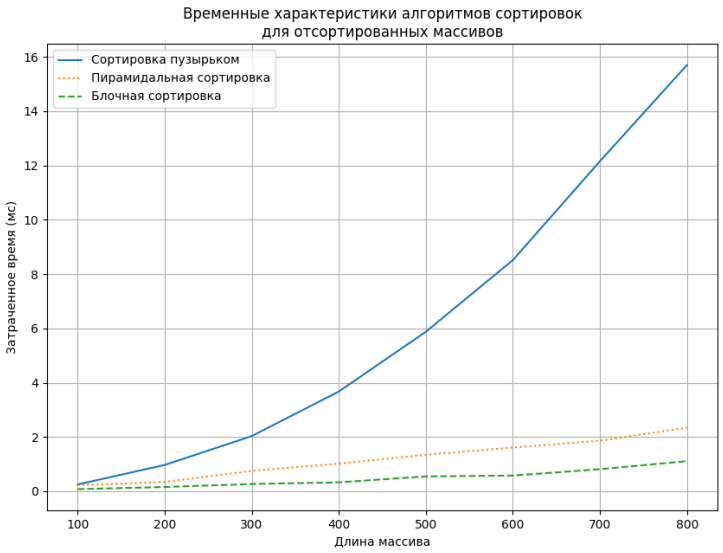
4.3 – , ( )

100	0.4062	0.2188	0.0781
200	1.8438	0.375	0.1406
300	4.0156	0.6094	0.2812
400	7.4062	0.8594	0.3906
500	11.5312	1.2188	0.6094
600	16.7188	1.5312	0.7656
700	23.4844	1.8906	0.8125
800	30.4844	2.0469	1.125

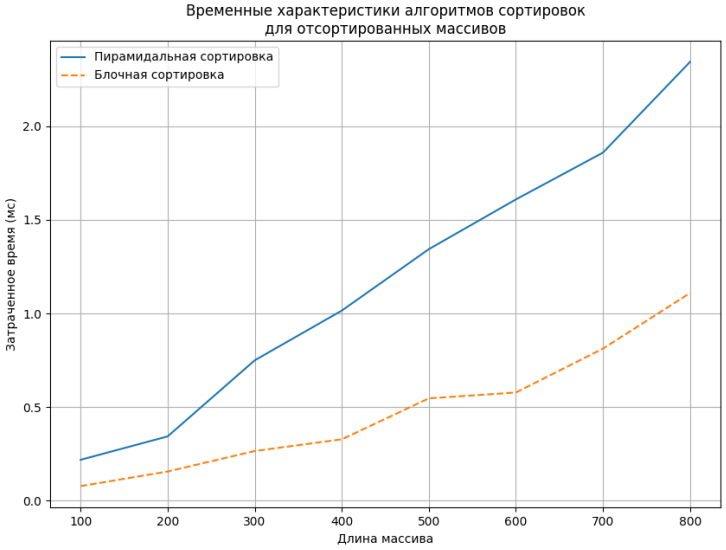
4.4 – c ( )

100	0.4062	0.2188	0.0781
200	1.8438	0.375	0.1406
300	4.0156	0.6094	0.2812
400	7.4062	0.8594	0.3906
500	11.5312	1.2188	0.6094
600	16.7188	1.5312	0.7656
700	23.4844	1.8906	0.8125
800	30.4844	2.0469	1.125

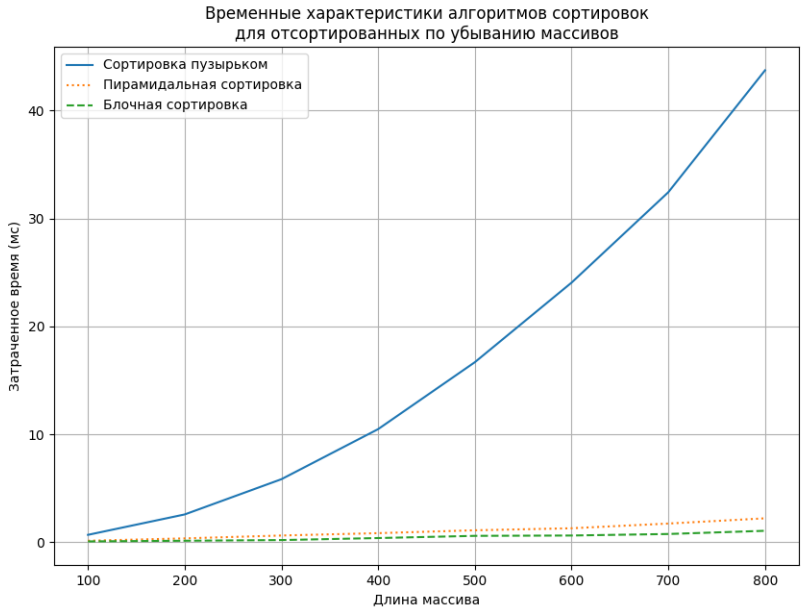
4.1 — 4.6 .


Временные характеристики алгоритмов сортировок
для отсортированных массивов
Сортировка пузырьком
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0
2
4
6
8
10
12
14
16
100
200
300
400
500
600
700
800

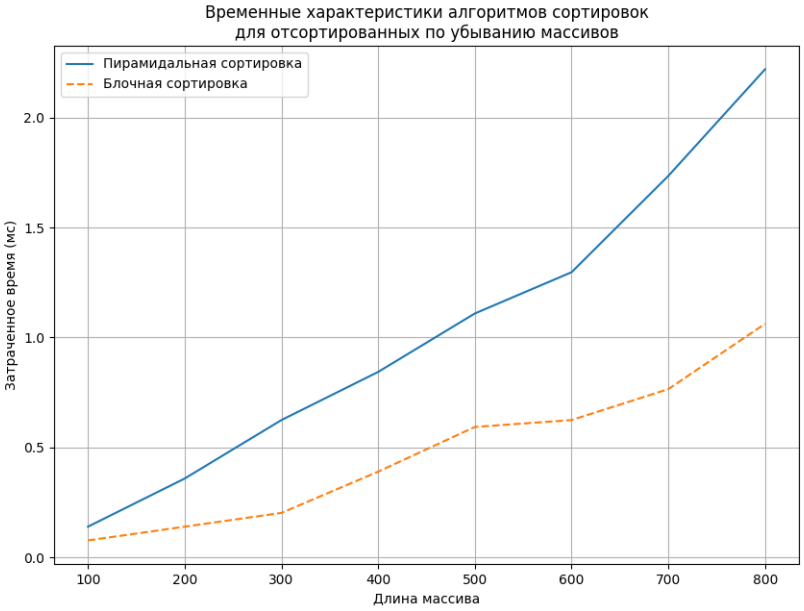
4.1 –


Временные характеристики алгоритмов сортировок
для отсортированных массивов
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0.0
0.5
1.0
1.5
2.0
100
200
300
400
500
600
700
800

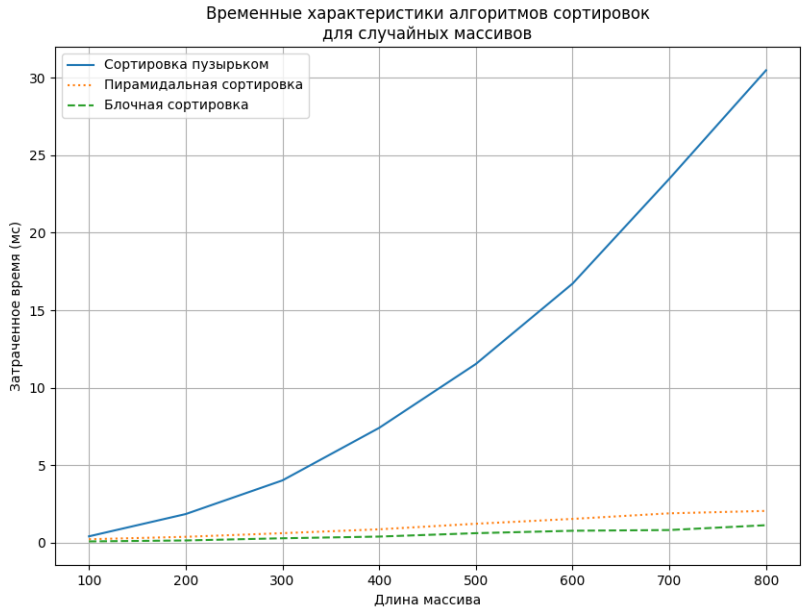
4.2 –


Временные характеристики алгоритмов сортировок
для отсортированных по убыванию массивов
Сортировка пузырьком
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0
10
20
30
40
100
200
300
400
500
600
700
800

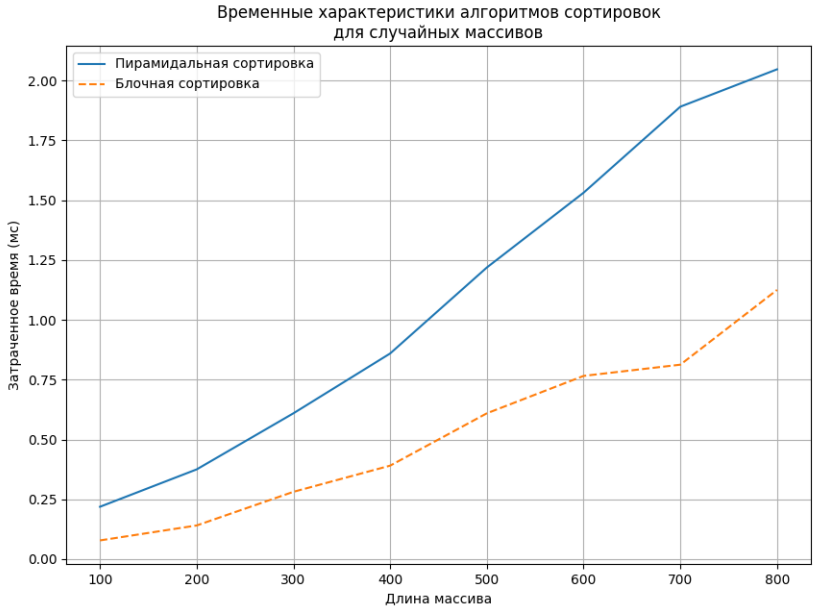
4.3 –


Временные характеристики алгоритмов сортировок
для отсортированных по убыванию массивов
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0.0
0.5
1.0
1.5
2.0
100
200
300
400
500
600
700
800

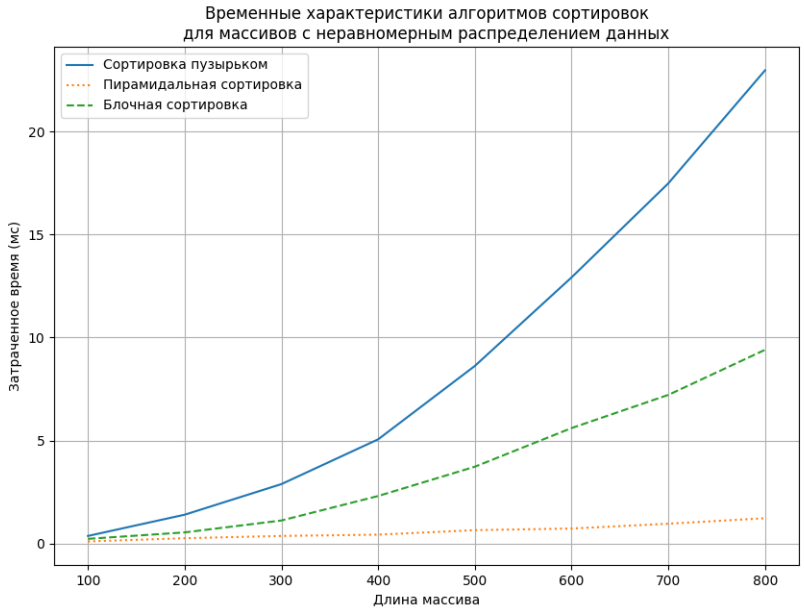
4.4 – ,


Временные характеристики алгоритмов сортировок
для случайных массивов
Сортировка пузырьком
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0
5
10
15
20
25
30
100
200
300
400
500
600
700
800

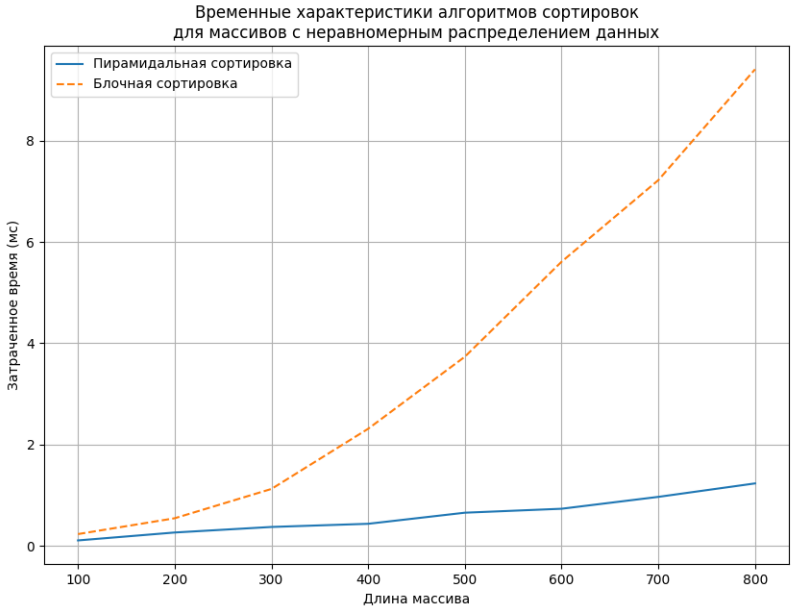
4.5 – ,


Временные характеристики алгоритмов сортировок
для случайных массивов
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
100
200
300
400
500
600
700
800

4.6 –


Временные характеристики алгоритмов сортировок
для массивов с неравномерным распределением данных
Сортировка пузырьком
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0
5
10
15
20
100
200
300
400
500
600
700
800

4.7 –


Временные характеристики алгоритмов сортировок
для массивов с неравномерным распределением данных
Пирамидальная сортировка
Блочная сортировка
Затраченное время (мс)
Длина массива
0
2
4
6
8
100
200
300
400
500
600
700
800

4.8 –

.

, (. 4.2, 4.4, 4.6). , (. 4.7).

, (800 15-20). , . , . .
, , - ( ). , , .

.

, — : , , .

:

– 3 : , , ;

– ;

– ;

– .

, (800 15-20). , . , . .

[1] Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to Algorithms. — MIT Press, Cambridge, 2009. — pp. 40–41.

[2] Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to Algorithms. — MIT Press, Cambridge, 2009. — pp. 151–156.

[3] Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to Algorithms. — MIT Press, Cambridge, 2009. — pp. 200–204.

[4] . . - . . – . , , 2007. . 376.

[5] Welcome to Python [ ]. : https://www.python.org (: 10.10.2022).

[6] Windows [ ]. : https://www.microsoft.com/en-us/windows (: 10.10.2022).

[7] Intel Core i5 [ ]. : https://www.intel.com/processors/core/i5/docs (: 10.10.2022).

[8] time Time access and conversions [ ]. : https://docs.python.org/3/library/time.html (: 10.10.2022).