**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Новиков Г.В. |
| Преподаватель |  | Родионова Е.А. |

Санкт-Петербург

2022

Постановка задачи

Реализовать следующие алгоритмы сортировки:

1. Сортировка вставками (Insertion sort)

2. Сортировка выбором (Selection sort)

3. Пузырьковая сортировка (Bubble sort)

4. Сортировка слиянием (Merge sort)

5. Сортировка Шелла (Shell sort)

6. Быстрая сортировка (Quick sort)

Для каждого алгоритма сортировки указать временную асимптотическую

сложность для лучшего, худшего случая и среднего случая, а также

пространственную сложность, подкрепив это логическими построениями (в

меру своих сил). Свести получившиеся результаты в таблицу. Построить

график зависимости времени выполнения от размера входных данных и

определить временную асимптотическую сложность для лучшего, худшего,

среднего случая практически. Определить наиболее быстрый алгоритм

сортировки. Сравнить скорость его выполнения с одним из алгоритмов

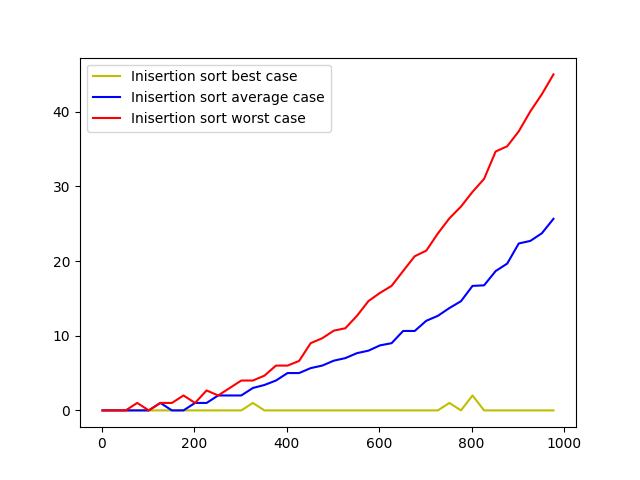
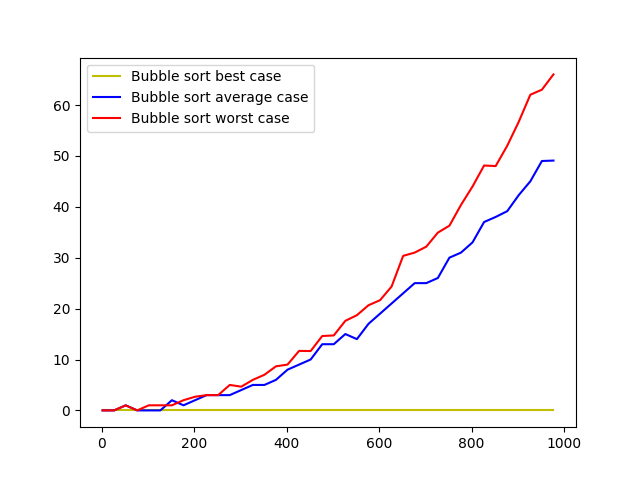
сортировки из базовых библиотек языка.

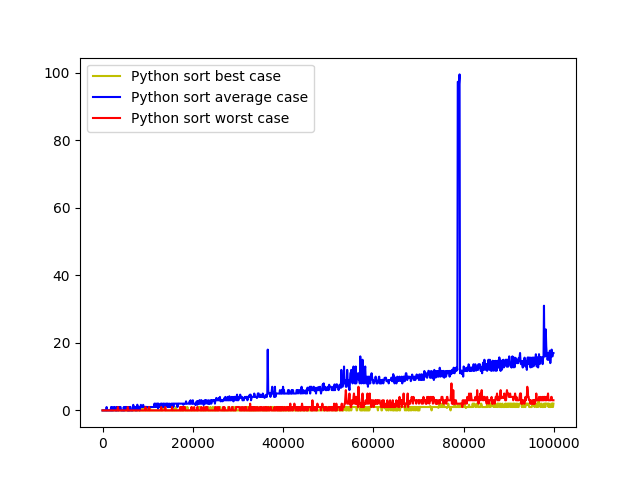
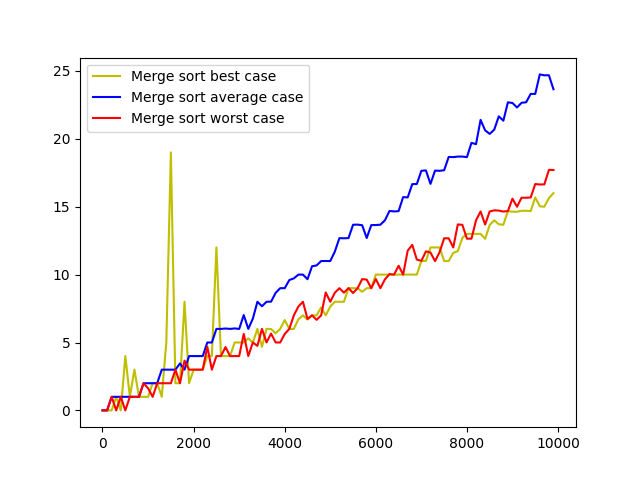
Оценка сложности

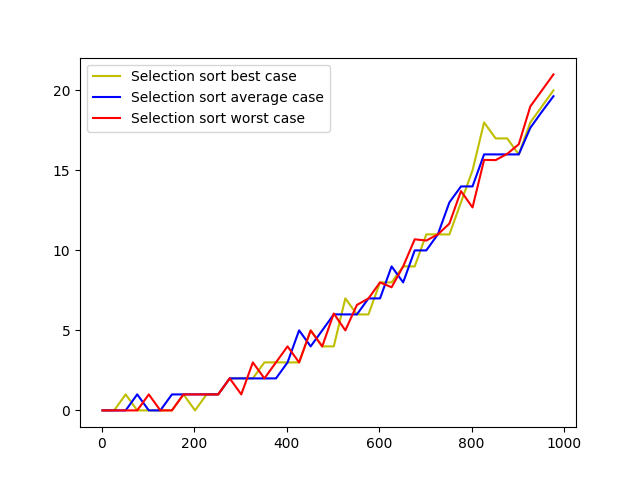
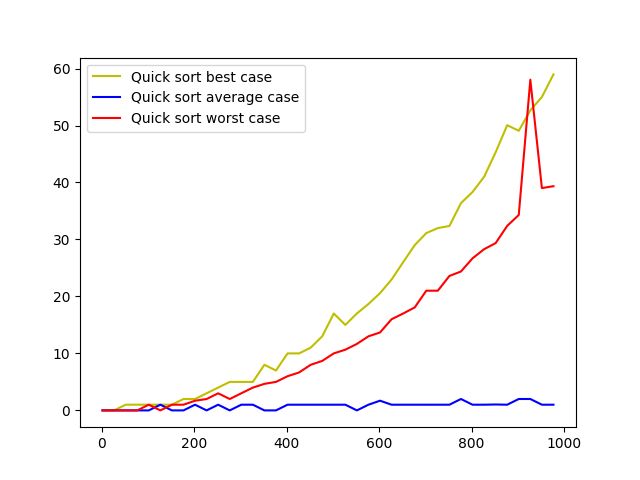
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сортировка** | **Временная сложность** | | | **Пространственная сложность** |
| **Best** | **Average** | **Worst** |
| insertion | n | n^2 | n^2 | 1 |
| selection | n^2 | n^2 | n^2 | 1 |
| bubble | n | n^2 | n^2 | 1 |
| merge | nlogn | nlogn | nlogn | n |
| shell | nlogn | nlogn | n^2 | 1 |
| quick | nlogn | nlogn | n^2 | logn |

**Графики**

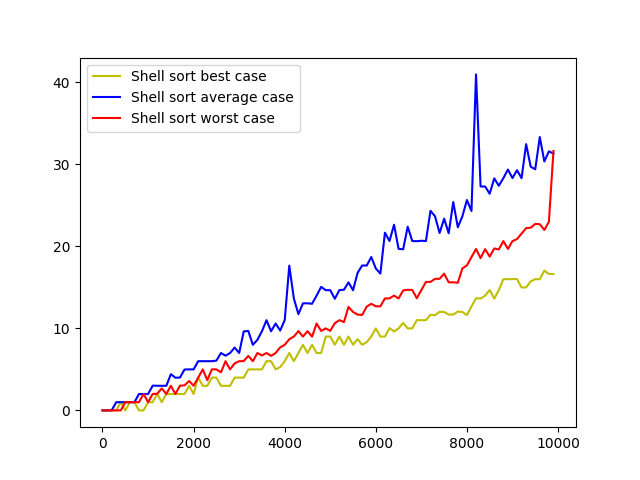
Bubble: Insertion:



Merge: Python sorted():

Quick: Selection: 

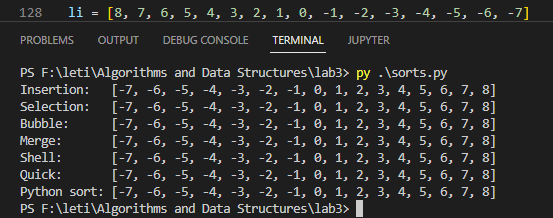
Shell:



Наиболее быстрый алгоритм из реализованных – Merge sort

Имеет такую же временную сложность, как и функция sorted() стандартной библиотеки python – во всех случаях ϴ(nlogn), но при одном размере входных данных sorted работает в несколько раз быстрее(видно на графике). Вероятно, это связано с тем, что стандартные функции в python написаны на C.

Пример работы



Листинг

def insertion\_sort(li: list, n: int) -> list:

# Worst: n^2

# Average: n^2

# Best: n

# Memory: 1

for i in range(1, n):

el = li[i]

j = i - 1

while j >= 0 and li[j] > el:

li[j + 1] = li[j]

j -= 1

li[j + 1] = el

return li

def selection\_sort(li: list, n: int) -> list:

# Worst: n^2

# Average: n^2

# Best: n^2

# Memory: 1

for i in range(n):

min\_el\_i = i

for j in range(i + 1, n):

if li[j] < li[min\_el\_i]:

min\_el\_i = j

li[i], li[min\_el\_i] = li[min\_el\_i], li[i]

return li

def bubble\_sort(li: list, n: int) -> list:

# Optimized

# Worst: n^2

# Average: n^2

# Best: n

# Memory: 1

for i in range(n - 1):

swapped = False

for j in range(n - 1 - i):

if li[j] > li[j + 1]:

li[j], li[j + 1] = li[j + 1], li[j]

swapped = True

if not swapped:

break

return li

def merge(left: list, left\_len: int, right: list, right\_len: int):

# O(n)

# Memory: n

res\_li = []

r, l = 0, 0

for i in range(left\_len + right\_len):

if l == left\_len:

res\_li.extend(right[r:])

break

elif r == right\_len:

res\_li.extend(left[l:])

break

elif left[l] < right[r]:

res\_li.append(left[l])

l += 1

else:

res\_li.append(right[r])

r += 1

return res\_li

def merge\_sort(li: list, n: int) -> list:

# Worst: nlogn

# Average: nlogn

# Best: nlogn

# Memory: n

if n == 1:

return li

mid\_i = n // 2

left = merge\_sort(li[:mid\_i], mid\_i)

right = merge\_sort(li[mid\_i:], n - mid\_i)

li = merge(left, mid\_i, right, n - mid\_i)

return li

def shell\_sort(li: list, n: int) -> list:

# Worst: n^2

# Average: nlogn

# Best: nlogn

# Memory: 1

gap = n // 2

while gap > 0:

for i in range(gap, n):

el = li[i]

j = i - gap

while j >= 0 and li[j] > el:

li[j + gap] = li[j]

j -= gap

li[j + gap] = el

gap //= 2

return li

def partition(li: list, n: int) -> int:

# O(n)

# Memory: 1

pivot = li[-1]

wall = 0 # left to the wall - els <= pivot, right(or equal) to the wall - > pivot

for i in range(n - 1):

if li[i] <= pivot:

li[i], li[wall] = li[wall], li[i]

wall += 1

li[-1], li[wall] = li[wall], li[-1]

return li, wall

def quick\_sort(li: list, n: int) -> list:

# Worst: n^2

# Average: nlogn

# Best: nlogn

# Memory: logn

if n <= 0:

return li

li, wall = partition(li, n)

li = quick\_sort(li[:wall], wall) + [li[wall]] + quick\_sort(li[wall+1:], n - wall - 1)

return li

li = [8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7]

print("Insertion: ", insertion\_sort(li, len(li)))

print("Selection: ", selection\_sort(li, len(li)))

print("Bubble: ", " " \* 3, bubble\_sort(li, len(li)))

print("Merge: ", " " \* 4, merge\_sort(li, len(li)))

print("Shell: ", " " \* 4, shell\_sort(li, len(li)))

print("Quick: ", " " \* 4, quick\_sort(li, len(li)))

print("Python sort:", sorted(li))

Ссылка на код

[GitHub](https://github.com/Gregory-hub/leti/blob/master/Algorithms%20and%20Data%20Structures/lab3/sorts.py)