**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №2**

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

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

Санкт-Петербург

2022

Постановка задачи

Решить задачу с использованием принципа «Разделяй и властвуй».

Определить теоретическую асимптотическую сложность решения.

Эмпирически оценить временную сложность решения для «average-case».

О влюбленных улитках. На квадрате земли 1 км 2 располагаются улитки-гермафродиты. В момент времени каждая из улиток с постоянной

скоростью 1 cм/с ползет к улитке, являющейся ближайшей к ней в момент

времени t = 0, выбрав её в качестве спутника жизни. Определить время,

через которое первая пара улиток достигнет друг друга или наличие

ситуации, приводящей улиток в замешательство, наиболее эффективным

способом

Реализовано на Python 3.9.1

**Функции:**

|  |  |  |
| --- | --- | --- |
| create\_snails(side\_size: int, number\_of\_snails: int) | Создает улиток со случайными координатами | O(n) |
| distance(snail1: list, snail2: list) | Вычисляет расстояние между улитками | O(1) |
| merge\_snails(l\_snails: list, r\_snails: list, by\_y: bool = False) | Сливает улиток, сортируя по x или y | O(n) |
| merge\_sort(snails: list, by\_y: bool = False) | Сортирует слиянием | O(nlogn) |
| min\_in\_mid\_area(l\_snails: list, r\_snails: list) | Ищет наименьшее расстояние в зоне у границы разделения улиток на 2 части | O(n^2) |
| find\_min\_dist(snails: list, confused: bool = False) | Основной алгоритм | O(nlogn) |
| test\_algorithm(side\_size: int, number\_of\_snails: int) | Вычисляет время работы алгоритма с заданными данными | O(nlogn) |

Теоретическая асимптотическая сложность реализованного алгоритма

Воспользуемся идеей “Разделяй-и-властвуй”: разобьём наше множество по x-координате точек на два: P1 и P2, |Pi|=n/2 в каждом, рекурсивно найдём ответ в каждом из них: d1 и d2, а после этого попытаемся найти пару точек p1∈P1 и p2∈P2, таких что ρ(p1, p2)<min(d1, d2)=d.

Будем перебирать только те пары точек из обоих множеств, которые потенциально могут дать результат лучше, чем d. Заметим, что это означает, что нет смысла рассматривать точки, чья x-координата отличается от xx-координаты P(n2) хотя бы на d. Более того, зафиксировав, скажем p1∈P1, нет смысла рассматривать с ней в паре точки p2∈P2, которые отличаются по y-координате также хотя бы на d.

Таким образом, для каждой точки из P1 имеет смысл рассматривать только те точки из P2, которые лежат в квадрате размера 2d×2d с центром в этой точке. В дальнейшем мы докажем, что количество таких точек не превосходит некоторой константы, из чего следует, что число потенциально улучшающих ответ пар (на шаге слияния) не более чем линейно по n.

**Почему количество интересующих нас пар точек линейно по n?**

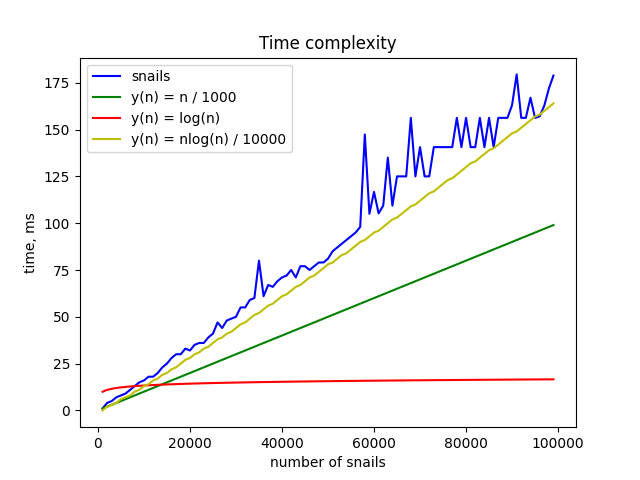
Пусть p1∈P1. Как мы уже поняли, все точки p2∈P2, которые могут образовывать интересную нам пару с p1 лежат в квадрате 2d×2d. Разобьём его на 16 квадратов размера d2 каждый. Заметим, что тогда в каждом из получившихся квадратиков не более одной интересующей нас точки, иначе бы в P2 нашлись 2 точки, расстояние между которыми не больше d/(2√2)<d≤d2, чего быть точно не может, иначе бы рекурсивный вызов в P2 нашёл бы это (меньшее, чем то, что он нашёл) расстояние.

Следовательно, для каждой точки из P1 существует не более чем константа потенциальных пар, а значит общее число потенциальных пар линейно по n.

2T(n/2) + 2O(nlogn) + O(n) = O(nlogn)

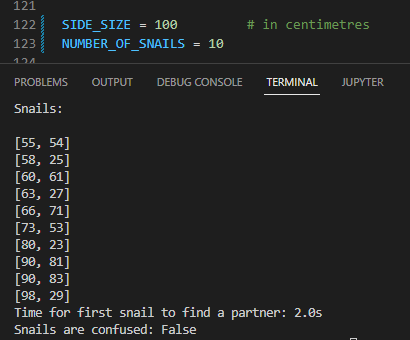
Оценка временной сложности эмпирическим способом

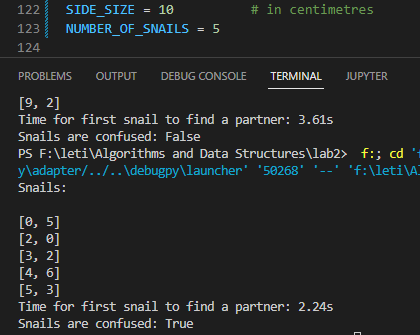
**Average-case:**



Как видим, график улиток примерно совпал с y = cnlogn

Пример работы





Улитка [3, 2] не может выбрать между [2, 0] и [5, 3] и умирает от избыточной работы мозга

Листинг

from random import randint

from math import ceil, log2

from matplotlib import pyplot as plt

from datetime import date, datetime

def create\_snails(side\_size: int, number\_of\_snails: int):

# O(n)

snails = []

for i in range(number\_of\_snails):

snails.append([randint(0, side\_size), randint(0, side\_size)])

return merge\_sort(snails)

def distance(snail1: list, snail2: list):

# O(1)

dist\_x = snail1[0] - snail2[0]

dist\_y = snail1[1] - snail2[1]

return (dist\_x\*\*2 + dist\_y\*\*2)\*\*(1/2)

def merge\_snails(l\_snails: list, r\_snails: list, by\_y: bool = False):

# O(n)

# n = len(l) + len(r)

snails = []

l, r = 0, 0

for i in range(len(l\_snails) + len(r\_snails)):

if l == len(l\_snails):

for j in range(r, len(r\_snails)):

snails.append(r\_snails[j])

break

if r == len(r\_snails):

for j in range(l, len(l\_snails)):

snails.append(l\_snails[j])

break

if by\_y:

main\_i = 1

secondary\_i = 0

else:

main\_i = 0

secondary\_i = 1

if l\_snails[l][main\_i] < r\_snails[r][main\_i] or (l\_snails[l][main\_i] == r\_snails[r][main\_i] and l\_snails[l][secondary\_i] < r\_snails[r][secondary\_i]):

snails.append(l\_snails[l])

l += 1

else:

snails.append(r\_snails[r])

r += 1

return snails

def merge\_sort(snails: list, by\_y: bool = False):

# 2T(n/2) + O(n) = O(nlogn)

if len(snails) <= 1:

return snails

mid\_i = len(snails) // 2

snails\_l = merge\_sort(snails[:mid\_i], by\_y) # T(n/2)

snails\_r = merge\_sort(snails[mid\_i:], by\_y) # T(n/2)

return merge\_snails(snails\_l, snails\_r, by\_y) # O(n)

def min\_in\_mid\_area(l\_snails: list, r\_snails: list):

# O(n\*\*2)

min\_dist = SIDE\_SIZE

for l\_sn in l\_snails:

r\_i = 0

init\_min\_dist = min\_dist

while r\_i < len(r\_snails) and l\_sn[1] - r\_snails[r\_i][1] > init\_min\_dist:

r\_i += 1

while r\_i < len(r\_snails) and l\_sn[1] - r\_snails[r\_i][1] <= init\_min\_dist:

if distance(l\_sn, r\_snails[r\_i]) < min\_dist:

min\_dist = distance(l\_sn, r\_snails[r\_i])

r\_i += 1

return min\_dist

def find\_min\_dist(snails: list, confused: bool = False):

# 2T(n/2) + 2O(nlogn) = O(nlogn)

# snails must be sorted by x first, than by y

if len(snails) <= 1:

return -1, snails, False

l\_min\_dist, l\_snails, l\_confused = find\_min\_dist(snails[:len(snails) // 2]) # T(n/2)

r\_min\_dist, r\_snails, r\_confused = find\_min\_dist(snails[len(snails) // 2:]) # T(n/2)

if l\_min\_dist == -1 and r\_min\_dist == -1: # one snail in each

min\_dist = distance(l\_snails[0], r\_snails[0]) # O(1)

elif l\_min\_dist == -1: # one snail in l

min\_dist = r\_min\_dist

elif r\_min\_dist == -1: # one snail in r

min\_dist = l\_min\_dist

else: # > 1 snails in each

if l\_min\_dist == r\_min\_dist:

confused = True

min\_dist = min(l\_min\_dist, r\_min\_dist)

l\_snails = merge\_sort(l\_snails[ceil(len(l\_snails) - min\_dist):], by\_y=True) # O(nlogn)

r\_snails = merge\_sort(r\_snails[:ceil(min\_dist)], by\_y=True) # O(nlogn)

# from (snails / 2 - min\_dist) to (snails / 2 + min\_dist)

min\_mid = min\_in\_mid\_area(l\_snails, r\_snails) # O(1) because len(l) + len(r) <= const

if min\_dist == min\_mid:

confused = True

elif min\_mid < min\_dist:

min\_dist = min\_mid

confused = False

return min\_dist, snails, confused

def test\_algorithm(side\_size: int, number\_of\_snails: int):

snails = create\_snails(side\_size, number\_of\_snails)

start\_time = datetime.now()

find\_min\_dist(snails)

end\_time = datetime.now()

return (end\_time - start\_time).total\_seconds()

SIDE\_SIZE = 10 # in centimetres

NUMBER\_OF\_SNAILS = 5

snails = create\_snails(SIDE\_SIZE, NUMBER\_OF\_SNAILS)

start\_time = datetime.now()

min\_dist, snails, confused = find\_min\_dist(snails)

end\_time = datetime.now()

print("Snails:\n", \*snails, sep='\n')

print("Time for first snail to find a partner:", str(round(min\_dist, 2)) + "s") # min\_dist / speed = min\_dist / 1 = min\_dist

print("Snails are confused:", confused)

# print("Runtime:", (end\_time - start\_time).total\_seconds())

# x = [i for i in range(250, 50000, 500)]

# y\_snails = []

# for number\_of\_snails in x:

# y\_snails.append(round(test\_algorithm(SIDE\_SIZE, number\_of\_snails) \* 1000, 2))

# y\_n = [i // 10 for i in x]

# y\_log = [log2(i) for i in x]

# y\_nlog = [i \* log2(i) // 100 for i in x]

# plt.plot(x, y\_snails, color='b', label="snails")

# plt.plot(x, y\_n, color='g', label="y(n) = n / 10")

# plt.plot(x, y\_log, color='r', label="y(n) = log(n)")

# plt.plot(x, y\_nlog, color='y', label="y(n) = nlog(n) / 100")

# plt.xlabel("number of snails")

# plt.ylabel("time, ms")

# plt.title("Time complexity")

# plt.legend()

# plt.show()