**Министерство образования и науки Украины**

**Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского"**

**Факультет информатики и вычислительной техники**

**Кафедра автоматизированных систем обработки**

**информации и управления**

**ОТЧЕТ**

по лабораторной работе № 1 по дисциплине

«Проектирование и анализ вычислительных алгоритмов»

„ **Проектирование и анализ алгоритмов сортировки** ”

**Выполнил**

(шифр, фамилия, имя, отчество)

*ІП-61 Кушка М.О.*

**Проверил**

(фамилия, имя, отчество )

*Головченко М.Н.*

Киев 2018

Содержание

[1 Цель лабораторной работы 3](#_Toc509037345)

[2 Задание 4](#_Toc509037346)

[3 Выполнение 6](#_Toc509037347)

[3.1 Анализ алгоритма на соответствие свойствам 6](#_Toc509037348)

[3.2 Псевдокод алгоритма 7](#_Toc509037349)

[3.3 Анализ временной сложности 9](#_Toc509037350)

[3.4 Программная реализация алгоритма 9](#_Toc509037351)

[3.4.1 Исходный код 9](#_Toc509037352)

[3.4.2 Примеры работы 16](#_Toc509037353)

[3.5 Испытания алгоритма 18](#_Toc509037354)

[3.5.1 Временные оценочные характеристики 18](#_Toc509037355)

[3.5.2 Графики зависимости временных оценочных характеристик от размерности массива 20](#_Toc509037356)

[Выводы 21](#_Toc509037357)

[Критерии оценивания Error! Bookmark not defined.](#_Toc509037358)

# Цель лабораторной работы

Цель работы – изучить основные подходы к анализу вычислительной сложности алгоритмов внутренней сортировки и оценить порог их эффективности.

# Задание

Согласно варианту (таблица 2.1), выполнить анализ алгоритма внутренней сортировки на соответствие следующим свойствам:

* устойчивость;
* естественность поведения;
* основанность на сравнениях;
* потребности в дополнительной памяти (объем);
* потребности в знаниях о структуре данных.

Записать алгоритм внутренней сортировки при помощи псевдокода (или другого способа по выбору).

Провести анализ временной сложности в худшем, лучшем и среднем случае и записать временную сложность в асимптотических оценках.

Выполнить программную реализацию алгоритма на любом языке программирования с фиксацией временных оценочных характеристик (количество сравнений, количество перестановок, глубина рекурсивного углубления и пр. в зависимости от алгоритма).

Провести ряд испытаний алгоритма на массивах разной размерности (10, 100, 1000, 5000, 10000, 20000, 50000 элементов) и разных наборах входных данных (упорядоченный массив, обратно упорядоченный массив, массив случайных чисел) и построить графики зависимости временных оценочных характеристик от размерности массива, нанести на график асимптотические оценку худшего и лучшего случаев для сравнения.

Сделать обобщенный вывод по лабораторной работе.

Таблица 2.1 – Варианты алгоритмов

|  |  |
| --- | --- |
| **№** | **Алгоритм сортировки** |
| 1 | Сортировка перемешиванием |
| 2 | Сортировка расчёской |
| 3 | Сортировка выбором |
| 4 | Сортировка Шелла (классическая) |
| 5 | Сортировка Шелла (*d* = последовательность Хиббарда) |
| 6 | Сортировка Шелла (*d* = последовательность Седжвика) |
| 7 | Сортировка Шелла (*d* = последовательность Пратта) |
| 8 | Сортировка Шелла (*d* = последовательность Марцина Циура) |
| 9 | Сортировка Шелла (*d* = последовательность Фибоначчи) |
| 10 | Сортировка Шелла (*d* = (3j−1)≤n, j ∈ N) |
| 11 | Быстрая сортировка (разбиение Ломуто) |
| 12 | Быстрая сортировка (разбиение Хоара) |
| 13 | Быстрая сортировка (повторяющиеся элементы) |
| 14 | Пирамидальная сортировка |
| 15 | Плавная сортировка |
| 16 | Интроспективная сортировка |
| 17 | Сортировка с помощью двоичного дерева |
| 18 | Сортировка перемешиванием |
| 19 | Сортировка Шелла (классическая) |
| 20 | Сортировка Шелла (*d* = последовательность Хиббарда) |
| 21 | Сортировка Шелла (*d* = последовательность Седжвика) |
| 22 | Сортировка Шелла (*d* = последовательность Пратта) |
| 23 | Сортировка Шелла (*d* = последовательность Марцина Циура) |
| 24 | Сортировка Шелла (*d* = последовательность Фибоначчи) |
| 25 | Сортировка с помощью двоичного дерева |
| 26 | Быстрая сортировка (разбиение Ломуто) |
| 27 | Быстрая сортировка (разбиение Хоара) |
| 28 | Быстрая сортировка (повторяющиеся элементы) |
| 29 | Пирамидальная сортировка |
| 30 | Плавная сортировка |

# Выполнение

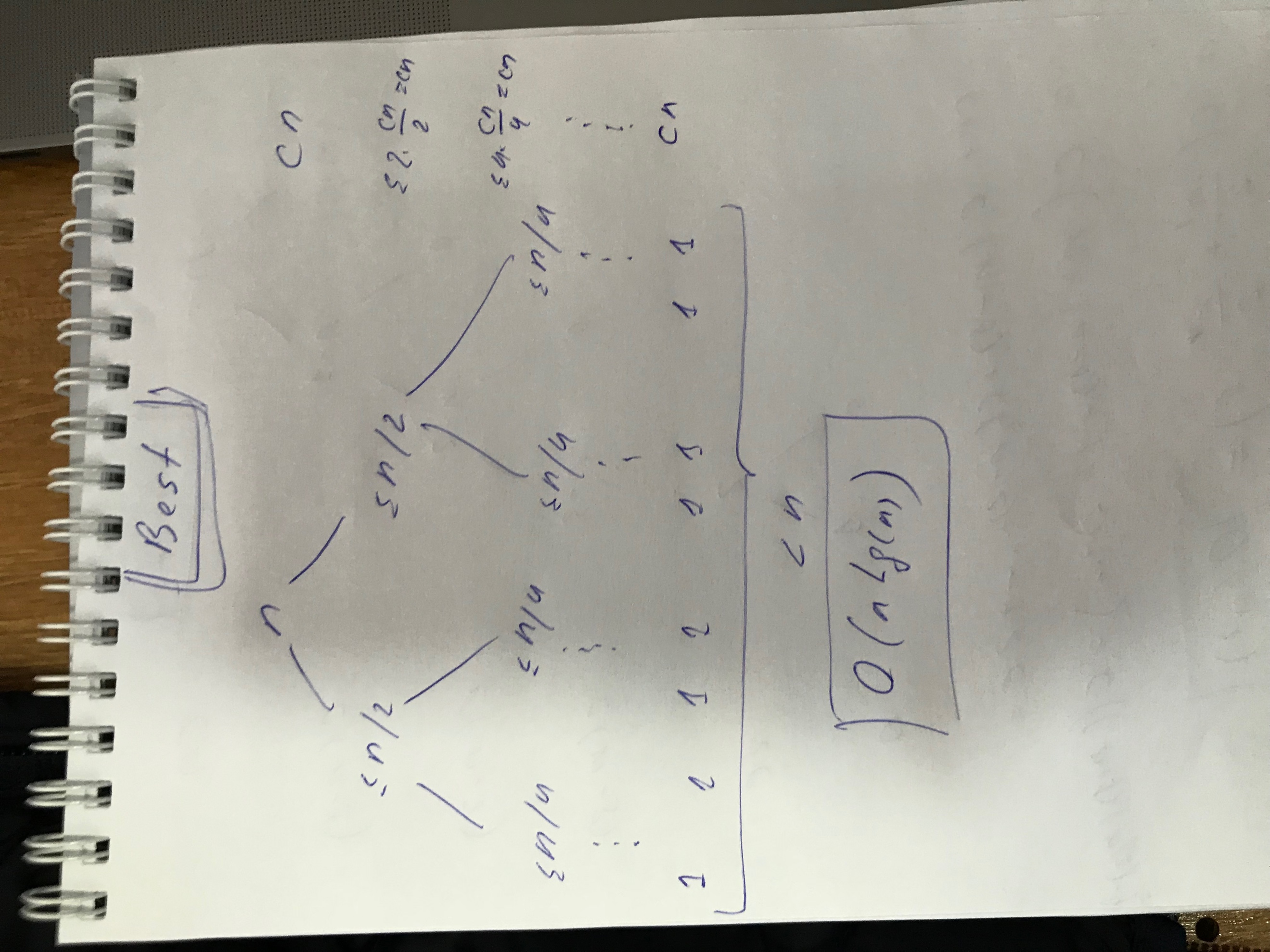
## Анализ алгоритма на соответствие свойствам

Анализ алгоритма быстрой сортировки на соответствие свойствам приведен в таблице 3.1.

Таблица 3.1 – Анализ алгоритма на соответствие свойствам

|  |  |
| --- | --- |
| **Свойство** | **Быстрая сортировка** |
| Устойчивость | Да |
| Естественность поведения | Да |
| Основанность на сравнениях | Да |
| Потребности в дополнительной памяти (объем) | O(log *n*) |
| Потребности в знаниях о структуре данных | Нет |

## 



## Псевдокод алгоритма

**algorithm** quickSort(alist, first, last) **is**

**if** first < last **then**

splitpoint = partition(alist,first,last)

quickSort(alist,first,splitpoint-1)

quickSort(alist,splitpoint+1,last)

**algorithm** partition(alist, first, last) **is**

pivotvalue = alist[first]

leftmark = first+1

rightmark = last

done = False

**while** leftmark < rightmark **do**

**while** leftmark <= rightmark and alist[leftmark] <= pivotvalue **do**

leftmark += 1

**while** alist[rightmark] >= pivotvalue and rightmark >= leftmark **do**

rightmark -= 1

**if** rightmark < leftmark **do**

done = True

**else** **do**

swap(alist[leftmark], alist[rightmark])

swap(alist[first], alist[rightmark])

return rightmark

## Анализ временной сложности

|  |  |
| --- | --- |
| Худшее время |  |
| Лучшее время | (обычное разделение)  или (разделение на 3 части) |
| Среднее время |  |

## Программная реализация алгоритма

### Исходный код

import random

import time

import pickle

import sys

import matplotlib.pyplot as plt

import numpy as np

import scipy as sp

import math

class QuickSort:

"""QuickSort algorithm with Hoare partition scheme."""

def \_\_init\_\_(self, lst):

"""Init list and auxiliary variables."""

self.lst = lst

self.comparisons = 0

self.swaps = 0

self.maxRecursionLevel = 0

def sort(self):

"""Sort list with the quicksort method."""

self.quickSort(self.lst, 0, len(self.lst)-1, self.maxRecursionLevel)

def showResult(self):

"""Show sorted list and additional sorting information."""

# print("List after sort:", self.lst)

print("Number of comparisons:", self.comparisons)

print("Number of swaps:", self.swaps)

print("Max recursion lever:", self.maxRecursionLevel)

def quickSort(self, lst, first, last, recursionLevel):

"""Main quicksort function."""

# Calculate recursion level.

if (recursionLevel > self.maxRecursionLevel):

self.maxRecursionLevel += recursionLevel

if first < last:

self.comparisons += 1

splitpoint = self.partition(self.lst, first, last)

self.quickSort(self.lst, first, splitpoint-1, recursionLevel+1)

self.quickSort(self.lst, splitpoint+1, last, recursionLevel+1)

def partition(self, lst, first, last):

"""Additional quicksort function to divide list on two parts."""

pivotvalue = lst[first]

leftmark = first + 1

rightmark = last

done = False

while not done:

while leftmark <= rightmark and lst[leftmark] <= pivotvalue:

leftmark += 1

while lst[rightmark] >= pivotvalue and rightmark >= leftmark:

rightmark -= 1

if rightmark < leftmark:

done = True

else:

lst[leftmark], lst[rightmark] = lst[rightmark], lst[leftmark]

self.swaps += 1

lst[first], lst[rightmark] = lst[rightmark], lst[first]

self.swaps += 1

return rightmark

def \_\_exit\_\_(self):

"""Exempts memory after using the class."""

print("Destructor")

del self.lst[:]

del self.comparisons

del self.swaps

del self.maxRecursionLevel

class EvaluateAlgorithm:

"""Evaluate the quicksort algorithm for different lists."""

def \_\_init\_\_(self):

self.lengths = [10, 100, 1000, 2000, 5000, 10000, 20000]

self.folders = ['10', '100', '1000', '2000', '5000', '10000', '20000']

self.files = ['ordered', 'backordered', 'random']

def saveToFile(self):

"""Save different lists to files."""

for lng, folder in zip(self.lengths, self.folders):

ordered = [x for x in range(1, lng+1)]

backordered = [x for x in range(lng, 0, -1)]

rand = [random.randint(1, lng) for x in range(lng)]

with open('data/' + folder + '/' + self.files[0], 'wb') as file:

pickle.dump(ordered, file)

with open('data/' + folder + '/' + self.files[1], 'wb') as file:

pickle.dump(backordered, file)

with open('data/' + folder + '/' + self.files[2], 'wb') as file:

pickle.dump(rand, file)

def evaluate(self):

"""

Evaluate the time of sorting with the quicksort algorithm for different

lists.

"""

sys.setrecursionlimit(40000)

evaluation = list()

for folder in self.folders:

line = []

print("Folder:", folder)

with open('data/' + folder + '/' + self.files[0], 'rb') as file:

ordered = pickle.load(file)

quicksort = QuickSort(ordered)

start = time.time()

quicksort.sort()

end = time.time()

quicksort.showResult()

del quicksort

line.append(end - start)

print("Ordered:", line[0])

print("\n\n")

with open('data/' + folder + '/' + self.files[1], 'rb') as file:

backordered = pickle.load(file)

quicksort = QuickSort(backordered)

start = time.time()

quicksort.sort()

end = time.time()

quicksort.showResult()

del quicksort

line.append(end - start)

print("Backordered:", line[1])

print("\n\n")

with open('data/' + folder + '/' + self.files[2], 'rb') as file:

rand = pickle.load(file)

quicksort = QuickSort(rand)

start = time.time()

quicksort.sort()

end = time.time()

quicksort.showResult()

del quicksort

line.append(end - start)

print("Random:", line[2])

print("\n\n")

evaluation.append(line)

return evaluation

def bestTime(self, n):

"""Best algorithm's working time."""

return n \* math.log(n)

def worstTime(self, n):

"""The worst algorithm's working time."""

return n\*n

def showPlots(self):

"""Shows 3 plots for 3 types of arrays. Plot is time/size."""

times = [

[6.198883056640625e-05, 5.412101745605469e-05,

3.886222839355469e-05],

[0.0013110637664794922, 0.0012891292572021484,

0.00041794776916503906],

[0.11106300354003906, 0.10880279541015625, 0.0057849884033203125],

[0.41941189765930176, 0.42520689964294434, 0.015279054641723633],

[2.740694046020508, 2.76031494140625, 0.03850197792053223],

[10.79902696609497, 11.202094078063965, 0.0696408748626709],

[37.34802293777466, 36.93501591682434, 0.1409311294555664]

]

orderedTimes = []

backorderedTimes = []

randTimes = []

for ordered, backordered, rand in times:

orderedTimes.append(ordered)

backorderedTimes.append(backordered)

randTimes.append(rand)

lng = len(orderedTimes)

best = [self.bestTime(x) for x in range(1, 100)]

worst = [self.worstTime(x) for x in range(1, 100)]

y = [x for x in range(1, 100)]

plot = DrawPlot(

[orderedTimes, backorderedTimes, randTimes],

[self.lengths, self.lengths, self.lengths],

legends=[

"Ordered lists",

"Backordered lists",

"Random data lists"

]

)

asymptoticEvaluation = DrawPlot(

[best, worst],

[y, y],

legends=[

"O(n\*log(n))",

"O(n^2)",

],

showLine=False

)

plot.show()

asymptoticEvaluation.show()

class DrawPlot:

"""Can draw any x/y plots."""

def \_\_init\_\_(

self,

xs, ys,

legends,

showLine=True,

title="Different lists comparison",

xlabel="Time (sec)",

ylabel="Length (elements)",

figureName="Quicksort evaluation"):

"""Set all necessary parameters."""

self.xs = xs

self.ys = ys

self.legends = legends

self.showLine = showLine

self.title = title

self.xlabel = xlabel

self.ylabel = ylabel

self.figureName = figureName

self.colors = ['b', 'm', 'y', 'c', 'g', 'r', 'k']

self.dots = ['x', 'o', 'v', '\*', '+', '.', ',']

def show(self):

"""Show the plot."""

for x, y, color, legend, marker in zip(self.xs, self.ys, self.colors, self.legends, self.dots):

f2p = sp.polyfit(x, y, 2)

f2 = sp.poly1d(f2p)

fx = sp.linspace(0, x[-1], 1000)

plt.figure(figsize=(12, 8), num=self.figureName)

plt.plot(fx, f2(fx), linewidth=int(self.showLine), color=color, label=legend)

plt.title(self.title)

plt.xlabel(self.xlabel)

plt.ylabel(self.ylabel)

plt.scatter(x, y, color=color, marker=marker)

plt.legend(loc='lower right')

plt.grid(True, linestyle='-', color='0.75')

plt.show()

def main():

# Ask user about list parameters.

n = int(input("Enter number of elements in the list\n> "))

minValue = int(input("Enter min possible value in the list\n> "))

maxValue = int(input("Enter max possible value in the list\n> "))

print("n =", n)

print("max =", maxValue)

print("min =", minValue)

lst = [random.randint(minValue, maxValue) for x in range(n)]

print("List before sort:", lst)

quickSortClass = QuickSort(lst)

start = time.time()

quickSortClass.sort()

end = time.time()

quickSortClass.showResult()

print("Working time: %f sec" % (end - start))

# Test the algorithm

# evaluate = EvaluateAlgorithm()

# evaluation = evaluate.evaluate()

# print(evaluation)

# evaluate.showPlots()

# evaluate.saveToFile()

if \_\_name\_\_ == '\_\_main\_\_':

main()

### Примеры работы

На рисунках 3.1 и 3.2 показаны примеры работы программы сортировки массивов на 10 и 100 элементов соответственно.

Рисунок 3.1 – Сортировка массива на 10 элементов

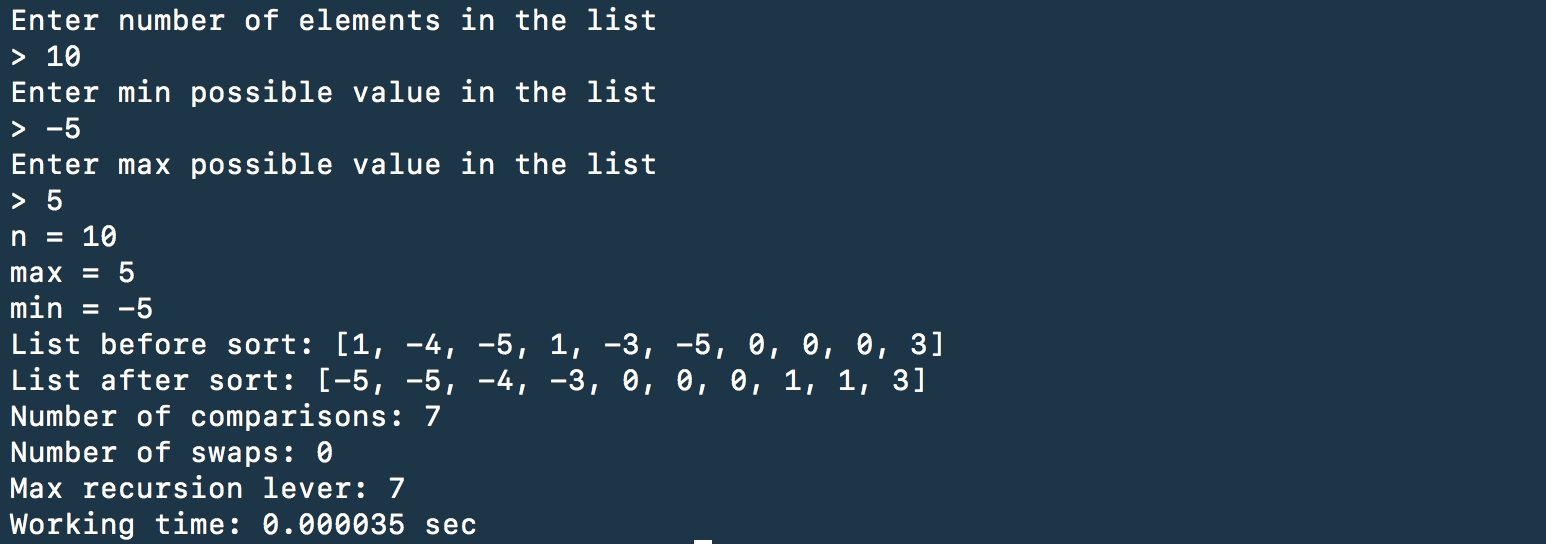


Рисунок 3.2 – Сортировка массива на 100 элементов

## 

## Испытания алгоритма

### Временные оценочные характеристики

В таблице 3.2 приведены оценочные характеристики числа сравнений, числа перестановок и глубины рекурсии алгоритма быстрой сортировки с разбиением Хоара для массивов разной размерности, когда массивы содержат упорядоченную последовательность элементов.

Таблица 3.2 – Оценочные характеристики алгоритма быстрой сортировки с разбиением Хоара для упорядоченной последовательности элементов в массиве

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность массива | Число сравнений | Число перестановок | Глубина рекурсии |
| 10 | 9 | 9 | 9 |
| 100 | 99 | 99 | 99 |
| 1000 | 999 | 999 | 999 |
| 2000 | 1999 | 1999 | 1999 |
| 5000 | 4999 | 4999 | 4999 |
| 10000 | 9999 | 9999 | 9999 |
| 20000 | 19999 | 19999 | 19999 |

В таблице 3.3 приведены оценочные характеристики числа сравнений, числа перестановок и глубины рекурсии алгоритма быстрой сортировки с разбиением Хоара для массивов разной размерности, когда массивы содержат обратно упорядоченную последовательность элементов.

Таблица 3.3 – Оценочные характеристики алгоритма быстрой сортировки с разбиением Хоара для обратно упорядоченной последовательности элементов в массиве

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность массива | Число сравнений | Число перестановок | Глубина рекурсии |
| 10 | 9 | 9 | 9 |
| 100 | 99 | 99 | 99 |
| 1000 | 999 | 999 | 999 |
| 2000 | 1999 | 1999 | 1999 |
| 5000 | 4999 | 4999 | 4999 |
| 10000 | 9999 | 9999 | 9999 |
| 20000 | 19999 | 19999 | 19999 |

В таблице 3.4 приведены оценочные характеристики числа сравнений, числа перестановок и глубины рекурсии алгоритма быстрой сортировки с разбиением Хоара для массивов разной размерности, массивы содержат случайную последовательность элементов.

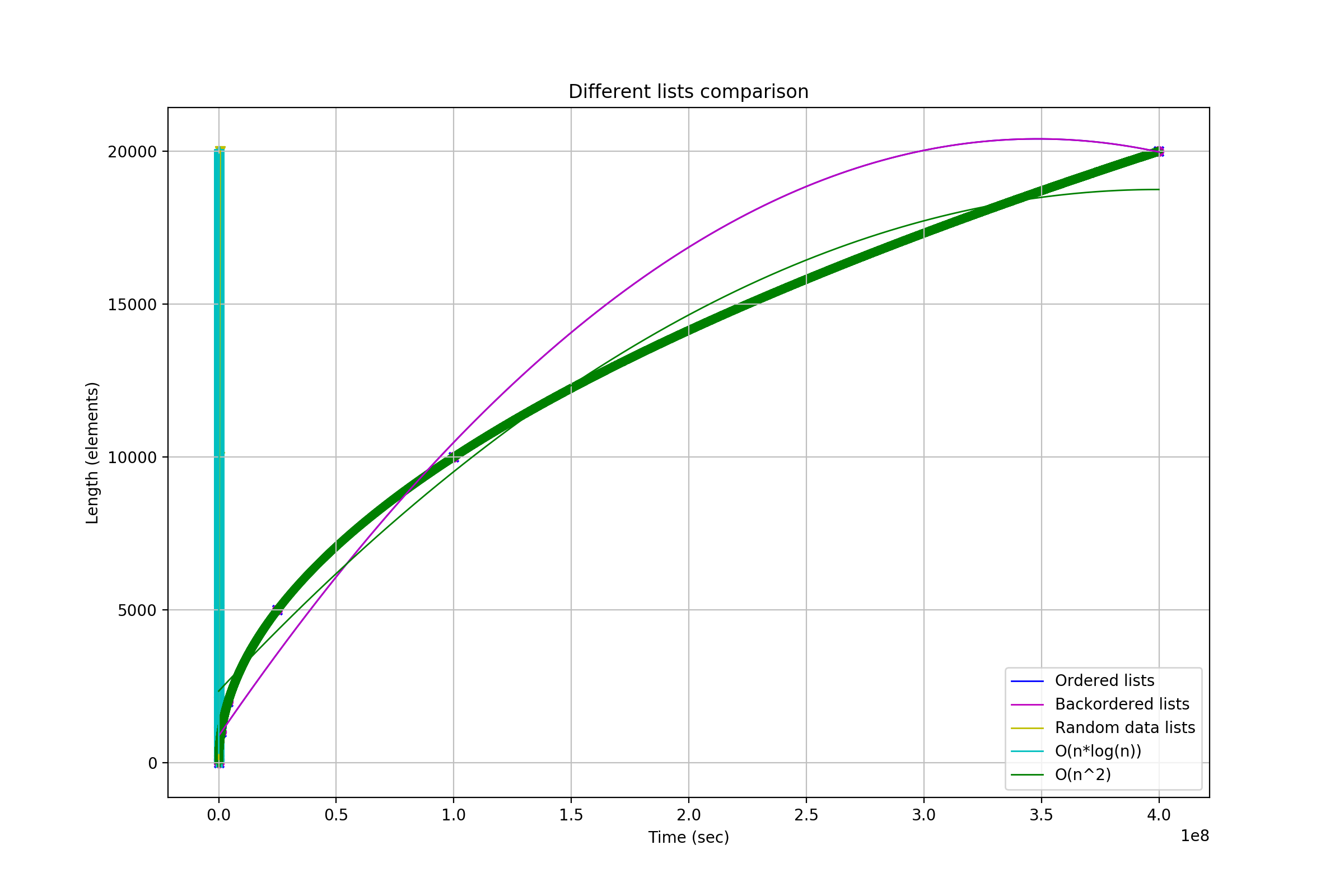
Таблица 3.4 – Оценочные характеристики алгоритма быстрой сортировки с разбиением Хоара для случайной последовательности элементов в массиве

|  |  |  |  |
| --- | --- | --- | --- |
| Размерность массива | Число сравнений | Число перестановок | Глубина рекурсии |
| 10 | 6 | 9 | 4 |
| 100 | 66 | 156 | 14 |
| 1000 | 691 | 2302 | 22 |
| 2000 | 1395 | 5159 | 24 |
| 5000 | 3460 | 14150 | 30 |
| 10000 | 6967 | 31266 | 30 |
| 20000 | 13950 | 66848 | 34 |

### Графики зависимости временных оценочных характеристик шот размерности массива

На рисунке 3.3 показаны графики зависимости временных оценочных характеристик от размерности массива для случаев, когда массивы содержат упорядоченную последовательность элементов, когда массивы содержат обратно упорядоченную последовательность элементов, когда массивы содержат случайную последовательность элементов. На рисунке 3.4 также показаны асимптотические оценки худшего и лучшего случаев и асимптотические оценки алгоритмов.

Рисунок 3.3 – Графики зависимости временных оценочных характеристик



Выводы

В рамках данной лабораторной работы я познакомился с таким видом сортировки как «быстрая сортировка» при разбиении входного массива методом Хоара. Данный алгоритм, хотя и был придуман более сорока лет назад, до сих пор остается довольно быстрым на фоне современных алгоритмов сортировки.

Я реализовал данный алгоритм при помощи языка программирования Python, что, безусловно, делает сортировку довольно емкую по времени (что обусловлено особенностями языка реализации), однако, в целях лабораторной работы этот язык я считаю уместным, поскольку Python обладает рядом преимуществ по сравнению с другими языками программирования такими как, например простое построение графиков функций и прочее.