Национальный исследовательский университет «Высшая школа экономики» Специальность «Компьютерная безопасность»

Отчет
По лабораторной работе №1
«Сортировки»

по дисциплине «Методы программирования»
направления «Компьютерная безопасность»

Вариант 16

Шаплавский Л.П.
СКБ-182

Москва, 2021

Общее задание:

1) Реализовать на языке C++ сортировки для массива объектов в соответствии с вариантом.

2) Перегрузить операторы сравнения (>, <, >=, <=) для сравнения объектов.

3) Входные данные для сортировки массива обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).

4) Выбрать 7-10 наборов данных для сортировки размерности от 100 и более (но не менее 100000). Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Сделать вывод о том, в каком случае, какой из методов лучше применять. Графики можно строить программно или в любой из прикладных программ (MS Excel, Matlab, MathCad и т.д.).

Описание варианта:

Массив данных об абитуриентах, поступающих в институт: ФИО абитуриента, факультет, специальность, сумма баллов по вступительным испытаниям (сравнение по полям – сумма баллов (по убыванию), ФИО, факультет, специальность)

Необходимо реализовать следующие сортировки:

- Сортировка выбором
- Пирамидальная сортировка
- Сортировка слиянием

Код программы:

Программа состоит из следующих файлов:

- StudentClass.py - в нем реализован класс, его инициализация и перегрузка операторов сравнения объектов класса (>, <, >=, <=)
- Data_modul.py - в нем реализована функция для генерации выборки определённого размера, запись в файл, чтение файла и различные функции для вывода данных в консоль.
- SortingModul.py - в нем реализованы функции сортировок
- main.py - главный файл, из которого происходит вызов необходимых функций и засекается время работы каждого алгоритма сортировки

main.py

```
import data_modul
import StudentClass
import time
import SortingModule
import datetime




print(datetime.datetime.now())
sampl = [100,500,1000,5000,10000,50000,100000]

for i in range(len(sampl)):
    print('\n' + '-----' + str(sampl[i]) + '-----')
    data_modul.creation_of_data(sampl[i])  # создаем файл с данными
    data_array = data_modul.creation_array(sampl[i])

    start2 = time.time()
    b = SortingModule.HeapSort(data_array)
    end2 = time.time()
    data_modul.write_data_info('HeapSort     ', start2 - end2, sampl[i])
    data_modul.write_data_info_live('HeapSort     ', start2 - end2, sampl[i])

    start3 = time.time()
    c = SortingModule.merge_sort(data_array)
    end3 = time.time()
    data_modul.write_data_info('merge_sort   ', start3 - end3, sampl[i])
    data_modul.write_data_info_live('merge_sort   ', start3 - end3, sampl[i])

    start1 = time.time()
    a = SortingModule.selection_sort(data_array)
    end1 = time.time()
    data_modul.write_data_info('election_sort', start1 - end1, sampl[i])
    data_modul.write_data_info_live('election_sort', start1 - end1, sampl[i])



Time = 0
data_modul.output_file(list(reversed(c)),Time)

print(datetime.datetime.now())
```

data_modul.py

```
import random
import StudentClass
import datetime

def creation_of_data(N):
    name = ['Римский', 'Чайковский', 'Мусоргский', 'Бородин', 'Глинка', 'Скрябин',
'Рахманинов', 'Стравинский',
        'Прокофьев',
'Шостакович','Шаплавский','Шварев','Шимановский','Смирнов','Иванов','Кузнецов','Соколов','По
пов',

'Лебедев','Козлов','Новиков','Морозов','Петров','Волков','Соловьёв','Васильев','Зайцев','Пав
лов','Семёнов',
            'Голубев']

    fak = ['Факультет экономических наук','Московский институт электроники и математики им.
А.Н. Тихонова',
           'Факультет компьютерных наук','Высшая школа бизнеса','Факультет права',
           'Высшая школа юриспруденции и администрирования','Факультет гуманитарных
наук','Факультет социальных наук']

    spec = [['Департамент теоретической экономики','Департамент прикладной
экономики','Департамент математики'],
            ['Департамент электронной инженерии','Департамент компьютерной
инженерии','Департамент прикладной математики'],
            ['Департамент программной инженерии','Департамент анализа данных и
искусственного интеллекта','Департамент больших данных и информационного поиска'],
            ['Департамент бизнес-информатики','Департамент маркетинга','Департамент
операционного менеджмента и логистики'],
            ['Департамент теории права и межотраслевых юридических дисциплин','Департамент
публичного права','Департамент частного права'],
            ['Институт юридического менеджмента','Институт кадрового
администрирования','Институт спортивного менеджмента и права'],
            ['Школа философии и культурологии','Школа филологических наук','Институт
классического Востока и античности'],
            ['Департамент социологии','Департамент политики и управления','Департамент
психологии']]

    f1=open('datd.txt','w')#создание файла с данными
    for i in range(N):
        fak_num = random.randint(0,len(fak)-1)
        f1.write('{}'.format(random.randint(100,300))+ '|' +
name[random.randint(0,len(name)-1)] + '|' + fak[fak_num] + '|' +
spec[fak_num][random.randint(0,2)] + '\n')
    f1.close()

    return 0

def creation_array(N): #передача данных файла в массив
    f1 = open('datd.txt','r')
    data_array = []
    for i in range(N):
        line = f1.readline()
        data_array.append(StudentClass.Student(line))
    f1.close()
    return data_array

def array_info(array): #вывод массива
    for i in range(len(array)):
        array[i].PrintInfo()
    return 0
```

```
def output_file(array,Time): #запись данных в файл
    f1 = open('data_output.txt','w')
    for i in range(len(array)):
        f1.write(array[i].PrintInfoSTR())
    f1.write("time:"+str(Time))
    f1.close()
    return 0

def write_data_info(word,time,sampl):
    f1=open('datd_info.txt','a')
    f1.write(str(word)+ '|' + str(time) + '|' + str(sampl) + '|' +
str(datetime.datetime.now()) + '\n'+'\n')
    f1.close()
    return 0

def write_data_info_live(word,time,sampl):
    print(str(word) + '|' + str(time) + '|' + str(sampl) + '|' +
str(datetime.datetime.now()) + '\n' +'_________')
    return 0
```

StudentClass.py

```
class Student:
    level: int
    name: str
    fac: str
    spec: str

    def __init__(self,line):
        level,name,fac,spec = line.split('|')
        level = int(level)
        self.level = level
        self.name = name
        self.fac = fac
        self.spec = spec

    def PrintInfo(self):
        print(self.level,'|',self.name,'|',self.fac,'|',self.spec)

    def PrintInfoSTR(self):
        return str(str(self.level) + '|' + self.name + '|' + self.fac + '|' + self.spec)

    def __gt__(self, other):  # >
        if self.level > other.level:
            return True
        elif self.level == other.level:
            if self.name > other.name:
                return True
            elif self.name == other.name:
                if self.fac > other.fac:
                    return True
                elif self.fac == other.fac:
                    if self.spec > other.spec:
                        return True
                    else:
                        return False
                else:
                    return False
            else:
                return False
        else:
            return False

    def __lt__(self, other):  # <
        if self.level < other.level:
            return True
        elif self.level == other.level:
            if self.name < other.name:
                return True
            elif self.name == other.name:
                if self.fac < other.fac:
                    return True
                elif self.fac == other.fac:
                    if self.spec < other.spec:
                        return True
                    else:
                        return False
                else:
                    return False
            else:
                return False
        else:
            return False
```

```
    def __ge__(self, other):  # >=
        if self.level > other.level:
            return True
        elif self.level == other.level:
            if self.name > other.name:
                return True
            elif self.name == other.name:
                if self.fac > other.fac:
                    return True
                elif self.fac == other.fac:
                    if self.spec > other.spec:
                        return True
                    if self.spec == other.spec:
                        return True
                    else:
                        return False
                else:
                    return False
            else:
                return False
        else:
            return False

    def __le__(self, other):  # <=
        if self.level < other.level:
            return True
        elif self.level == other.level:
            if self.name < other.name:
                return True
            elif self.name == other.name:
                if self.fac < other.fac:
                    return True
                elif self.fac == other.fac:
                    if self.spec < other.spec:
                        return True
                    elif self.spec == other.spec:
                        return True
                    else:
                        return False
                else:
                    return False
            else:
                return False
        else:
            return False
```

SortingModul.py

```
import StudentClass
import copy



def selection_sort(data):
    array = copy.deepcopy(data)
    i = 0
    while i < len(array) - 1:
        smal = i
        j = i + 1
        while j < len(array):
            if array[j].__lt__(array[smal]):
                smal = j
            j += 1
```

```
        array[i], array[smal] = array[smal], array[i]
        i += 1
    return array



def HeapSort(lst):
    data=copy.deepcopy(lst)
    for i in range((len(data) - 2) // 2, -1, -1):
        HeapSift(data, i, len(data) - 1)

    for end in range(len(data) - 1, 0, -1):
        data[end], data[0] = data[0], data[end]
        HeapSift(data, 0, end - 1)
    return data

def HeapSift(data, start, end):
    root = start

    while True:

        child = root * 2 + 1
        if child > end:
            break

        if ((child + 1) <= end) and (data[child].__lt__(data[child + 1]) ):
            child += 1

        if data[root].__lt__(data[child]):
            data[root], data[child] = data[child], data[root]
            root = child
        else:
            break


def merge(left, right): #сортировка слиянием
    res = []
    i = 0
    j = 0
    while i < len(left) and j < len(right):
        if left[i].__le__(right[j]):
            res.append(a[i])
            i += 1
        else:
            res.append(right[j])
            j += 1
    res += left[i:] + right[j:]
    return res

def merge_sort(lst):
    array = copy.deepcopy(lst)
    if len(array) <= 1:
        return array
    else:
        left = array[:len(array) // 2]
        right = array[len(array) // 2:]
    return merge(merge_sort(left), merge_sort(right))
```

Результат:

HeapSort |-0.0023155212402343 75|100|2021-02-15 23:06:17.628034

merge_sort |-0.011972665786743164|100|2021-02-15 23:06:17.640198

election_sort|-0.003009319305419922|100|2021-02-15 23:06:17.643339

HeapSort |-0.011184453964233398|500|2021-02-15 23:06:17.659942

merge_sort |-0.07253575325012207|500|2021-02-15 23:06:17.732680

election_sort|-0.06038331985473633|500|2021-02-15 23:06:17.793271

HeapSort |-0.019860267639160156|1000|2021-02-15 23:06:17.821520

merge_sort |-0.14161086082458496|1000|2021-02-15 23:06:17.963295

election_sort|-0.29642629623413086|1000|2021-02-15 23:06:18.259943

HeapSort |-0.12724661827087402|5000|2021-02-15 23:06:18.437569

merge_sort |-0.9484734535217285|5000|2021-02-15 23:06:19.386259

election_sort|-4.873639106750488|5000|2021-02-15 23:06:24.260192

HeapSort |-0.25153350830078125|10000|2021-02-15 23:06:24.609956

merge_sort |-1.889941930770874|10000|2021-02-15 23:06:26.500170

election_sort|-21.05887484550476|10000|2021-02-15 23:06:47.569815

HeapSort |-1.602583646774292|50000|2021-02-15 23:06:49.918722

merge_sort |-11.456157207489014|50000|2021-02-15 23:07:01.375134

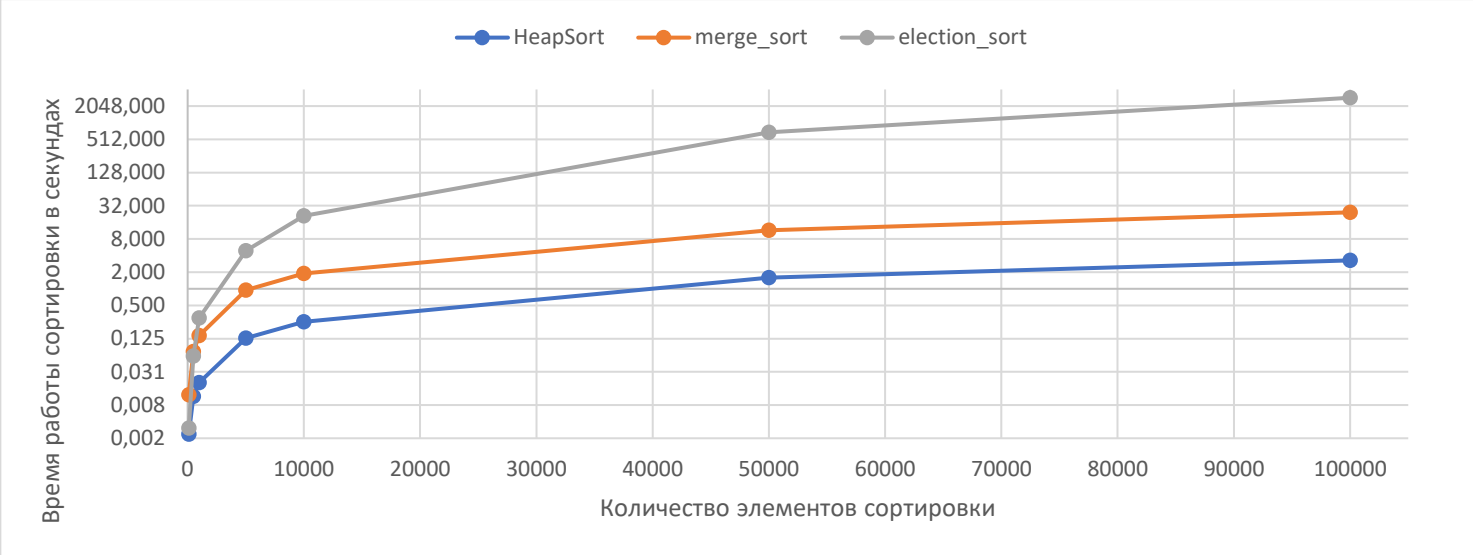
election_sort|-687.7948722839355|50000|2021-02-15 23:18:29.170279

HeapSort |-3.2735838890075684|100000|2021-02-15 23:18:33.613802

merge_sort |-24.29147481918335|100000|2021-02-15 23:18:57.905525

election_sort|-2897.26864027977|100000|2021-02-16 00:07:15.174423

Визуализация:



Вывод:

Результаты показывают, что что алгоритм пирамидальная сортировка работает немного быстрее, чем сортировка слиянием и гораздо быстрее, сортировка выбором.