Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного
обеспечения и администрирования
информационных систем
Направление подготовки
математическое обеспечение и
администрирование
информационных систем
Форма обучения очная

Отчет

по лабораторной работе №1
«Сравнительный анализ методов сортировки»

Выполнил:
студент группы 213 Тихонов Е.Е.

Проверил:
профессор кафедры ПОиАИС Кудинов В.А.

Курск, 2021

Цель работы: изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

Задание

1. Изучить временные характеристики алгоритмов.
2. Изучить методы сортировки.
3. Программно реализовать 3 метода сортировки массивов: быстрая сортировка (для разных способов выбора ключа), обменная поразрядная сортировка, двухпутевое слияние.
4. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов сортировки.
5. Провести эксперименты по определению временных характеристик алгоритмов сортировки. Результаты экспериментов представить в виде таблиц 1, 2 и 3, клетки которых содержат количество операций сравнения при выполнении алгоритма сортировки массива с заданным количеством элементов.
6. Построить график зависимости количества операций сравнения от количества элементов в сортируемом массиве.
7. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.
8. Определить порядок функций временной сложности алгоритмов сортировки при сортировке упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов.

Алгоритмы решения задач

Алгоритм быстрой сортировки Хоара состоит в том, чтобы разбить сортируемое множество на три подмножества относительно ключа. Ключ - это элемент множества. Обычно в качестве ключа берут первый, последний или средний элемент множества, или для наилучшей стабильности сортировки берут медианное значения из этих трех элементов. Множество разбивается на три подмножества: меньшее ключа, равное ключу и большее ключа. Для этого мы идем от левого края множества в центр и от правого края в центр и когда слева мы находим элемент больший ключа, а справа элемент меньший ключа, они меняются местами. Затем для получившегося подмножества которое меньше и для того которое больше производят тоже вычисление ключа и разбивают их на новые подмножества по тому же алгоритму, и так до момента пока подмножество не будет содержать всего 1 элемент.

Алгоритм обменной поразрядной сортировки похож на алгоритм быстрой сортировки. описанной выше, но в нем не используется ключ. Вместо ключа здесь сравниваются биты чисел начиная от старших разрядов и двигаясь к младшим. Самое левое число содержащее в бите 1 обменивается с самым правым числом содержащем в том же бите 0. Стоит отметить что данный алгоритм работает только на целых числах.

Алгоритм двухпутевого слияния заключается в разбиении множества на два подмножества относительно середины множества. Новые подмножества также разбиваются относительно середины и так до момента пока они не разобьются на одноэлементные множества. Затем начинаются операции слияния наших подмножеств. Берутся элементы подмножеств стоящие на одинаковых местах, сравниваются, меньший элемент записывается в новое объединенное множество, затем в том подмножестве из которого мы взяли элемент - берется следующий элемент, в том подмножестве из которого не взяли элемент - элемент остается прежним, они снова сравниваются и так далее пока одно из объединяемых подмножеств не кончится. Если во втором подмножестве еще остались элементы добавляем их в объединенное множество. Выполняем данный алгоритм для полученных объединенных множеств пока они не кончатся и не останется одно.

Сравнительный анализ алгоритмов

Временные затраты алгоритмов для разных множеств показаны в таблицах 1, 2, 3.

Таблица 1

Упорядоченный массив

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Быстрая (средний ключ)	31410 ns	13440 ns	10879 ns	16116 ns	34361 ns	24345 ns	43373 ns	34532 ns	41043 ns
Быстрая (левый ключ)	30570 ns	26161 ns	28992 ns	42316 ns	118897 ns	91607 ns	189711 ns	160753 ns	159821 ns
Быстрая (правый ключ)	35967 ns	22267 ns	24110 ns	34894 ns	84408 ns	87906 ns	135139 ns	106850 ns	152174 ns
Быстрая (медиана трех как ключ)	18827 ns	8619 ns	11616 ns	17853 ns	28468 ns	30329 ns	62309 ns	39323 ns	46446 ns
Поразрядная	71307	18934	25448	29938	37680	43445	84839	67442	60409

	ns	ns	ns	ns	ns	ns	ns	ns	ns
Слияние	48152 ns	13246 ns	20076 ns	27320 ns	53164 ns	43373 ns	84770 ns	57158 ns	67853 ns

Таблица 2

Массив, упорядоченный в обратном порядке

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Быстрая (средний ключ)	4463 ns	10459 ns	10615 ns	16921 ns	21546 ns	24006 ns	29938 ns	35659 ns	41275 ns
Быстрая (левый ключ)	4755 ns	17257 ns	24226 ns	39314 ns	57775 ns	80218 ns	100328 ns	133385 ns	145758 ns
Быстрая (правый ключ)	4924 ns	19758 ns	26320 ns	45321 ns	64200 ns	90715 ns	103786 ns	145310 ns	151811 ns
Быстрая (медиана трех как ключ)	3780 ns	11590 ns	11233 ns	26006 ns	24927 ns	26930 ns	32651 ns	43155 ns	46217 ns
Поразрядная	13292 ns	28091 ns	24923 ns	55154 ns	42081 ns	53082 ns	58516 ns	66505 ns	68971 ns
Слияние	6658 ns	18236 ns	20246 ns	47344 ns	37137 ns	42769 ns	51624 ns	60735 ns	67486 ns

Таблица 3

Неупорядоченный массив

Сортировка	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Быстрая (средний ключ)	6511 ns	10542 ns	19574 ns	22638 ns	32305 ns	35936 ns	40661 ns	57942 ns	53907 ns
Быстрая (левый ключ)	5517 ns	8409 ns	14745 ns	17791 ns	23382 ns	31073 ns	35468 ns	45913 ns	68154 ns
Быстрая	6279	8279	15421	20075	26291	30974	56289	51407	44465

(правый ключ)	ns	ns	ns	ns	ns	ns	ns	ns	ns
Быстрая (медиана трех как ключ)	4463 ns	9204 ns	14658 ns	19931 ns	24764 ns	32203 ns	35249 ns	42208 ns	52112 ns
Поразрядная	17217 ns	22161 ns	31378 ns	39635 ns	47197 ns	71813 ns	69491 ns	82974 ns	133008 ns
Слияние	7777 ns	22161 ns	21598 ns	29319 ns	47197 ns	73997 ns	61522 ns	65317 ns	95390 ns

Общая оценка сложности алгоритмов показана в таблице 4.

Таблица 4

Сортировка	Сложность		
	Данные упорядочены	Данные упорядочены в обратном порядке	Данные не упорядочены
Быстрая	n * log n	n * log n	n * log n
Поразрядная	n * k/d (*)	n * k/d (*)	n * k/d (*)
Слияние	n * log n	n * log n	n * log n

*k - количество разрядов в самом длинном ключе.

d - разрядность данных: количество возможных значений разряда ключа.

Текст программы

Текст файла quick_sort.py

```
def med(x, y, z):
    if y <= x and x <= z:
        return x
    elif z <= x and x <= y:
        return x

    elif z <= y and y <= x:
```

```
        return y
    elif x <= y and y <= z:
        return y

    elif y <= z and z <= x:
        return z
    elif x <= z and z <= y:
        return z



def quick_sort(array, key, timer):
    less = []
    equal = []
    greater = []

    if len(array) > 1:

        elm = array[0]
        if key == "left":
            elm = array[0]
        elif key == "mid":
            elm = array[len(array)//2]
        elif key == "right":
            elm = array[len(array)-1]
        elif key == "median":
            elm = med(array[0], array[len(array)//2],
array[len(array)-1])

        for x in array:

            timer.inc()

            if x < elm:
                less.append(x)
            elif x == elm:
                equal.append(x)
            elif x > elm:
                greater.append(x)
        return quick_sort(less, key,
timer)+equal+quick_sort(greater,key, timer)
    else:
        return array
```

Текст файла radix_sort.py

```
def get_bit(num, bit_ind):
    newnum = num>>bit_ind
    if(newnum % 2 == 1):
        return 1
    else:
        return 0

def radix_sort(arr, start, end, bit_ind, timer):

    if bit_ind == 0:
        return

    if start >=end:
        return

    i=start
    j=end

    while(i < j):

        timer.inc()

        while get_bit(arr[i], bit_ind) == 0 and i<j:
            i+=1

        while get_bit(arr[j], bit_ind) == 1 and i<j:
            j-=1

        if i < j:
            arr[i], arr[j] = arr[j], arr[i]

    if get_bit(arr[i], bit_ind) == 1:
        i-=1

    radix_sort(arr, start, i, bit_ind-1, timer)
    radix_sort(arr, i+1, end, bit_ind-1, timer)
```

Текст файла merge_sort.py

```
def merge_sort(arr, start, end, timer):
    if end - start > 1:
        mid = (start + end)//2
        merge_sort(arr, start, mid, timer)
        merge_sort(arr, mid, end, timer)
        merge(arr, start, mid, end, timer)
def merge(arr, start, mid, end, timer):
    left = arr[start:mid]
    right = arr[mid:end]
    k = start
    i = 0
    j = 0

    while (start + i < mid and mid + j < end):

        timer.inc()

        if (left[i] <= right[j]):
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            j += 1

        k += 1

    if start + i < mid:

        while k < end:

            timer.inc()

            arr[k] = left[i]
            i += 1
            k += 1
    else:

        while k < end:

            timer.inc()

            arr[k] = right[j]
```

```
            j = j + 1
            k = k + 1
```

Текст файла timer.py

```
class timer:
    def __init__(self):
        self.arr_len = 0
        self.iters = 0

    def set(self, arr):
        self.iters = 0
        self.arr_len = len(arr)

    def inc(self):
        self.iters+=1

    def get(self):
        print("len/iters = " + str(self.arr_len) + "/" +
str(self.iters))
```

Текст файла main.py

```
from quick_sort import quick_sort as qsort
from radix_sort import radix_sort as rsort
from merge_sort import merge_sort as msort
from timer import timer

import random
import time

def rnd_arr(len, min=-100, max=100, step = 1):
    arr = []
    for i in range(len):
        arr.append(random.randrange(min, max, step))
    return arr

def main():
    tmr = timer()

    size = 100

    for key in ["mid", "left", "right", "median"]:
        print("quick sort "+key+" key")
```

```
        arr1 = rnd_arr(size)
        tmr.set(arr1)
        print(arr1)
        t_start = time.monotonic_ns()
        s_arr1 = qsort(arr1, key, tmr)
        t_stop = time.monotonic_ns()
        print(s_arr1)
        tmr.get()
        print(str(t_stop - t_start) + " ns")

    print("radix sort")
    arr2 = rnd_arr(size, min=0, max = 128)
    tmr.set(arr2)
    print(arr2)
    t_start = time.monotonic_ns()
    rsort(arr2, 0, len(arr2)-1, 7, tmr)
    t_stop = time.monotonic_ns()
    print(arr2)
    tmr.get()
    print(str(t_stop - t_start) + " ns")

    print("merge sort")
    arr3 = rnd_arr(size)
    tmr.set(arr3)
    print(arr3)
    t_start = time.monotonic_ns()
    msort(arr3, 0, len(arr3), tmr)
    t_stop = time.monotonic_ns()
    print(arr3)
    tmr.get()
    print(str(t_stop - t_start) + " ns")


main()
```