**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

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

на тему: **«**Сортировка**»**

Выполнил: студент гр. ИП-31

Коваленко А.И

Принял: преподаватель Шибеко В.Н.

Гомель 2023

**Цель работы:** Изучить основные методы сортировки. Сравнить эффективность методов.

Разработать проект для обработки дерева поиска, каждый элемент которого содержит структуру данных из сформировать случайным образом массив записей из 100000 штук. Скопировать его трижды. Провести сортировку по одному ключу, двум ключам и трем ключам. Оценить эффективность каждой сортировки по времени и количеству перестановок и сравнений. Провести четвертую сортировку, включив проверку на каждом шаге, что массив уже отсортирован. Оценить эффективность включения такой оценки. Организовать поиск по трем указанным ключам. Оценить эффективность поиска.



**Алгоритм решения задачи:**

Сортировка вставками:

1. Предполагается, что первый элемент списка отсортирован. Переходим к следующему элементу, обозначим его value.

2. Если х больше первого, оставляем его на своём месте. Если он меньше, копируем его на вторую позицию, а value устанавливаем в качестве первого элемента.

3. выполняем алгоритм n раз, где n – длина массива

**Листинг программы:**

def binary\_search(sequence, start\_element, key):  
 end\_element = len(sequence) - 1  
 while start\_element <= end\_element:  
 middle\_element = start\_element + (end\_element - start\_element) // 2  
 if sequence[middle\_element].i == key.i and sequence[middle\_element].d==key.d and sequence[middle\_element].s==key.s :  
 return middle\_element  
 elif sequence[middle\_element].i < key.i or (sequence[middle\_element].i == key.i and sequence[middle\_element].d<key.d )\  
 or (sequence[middle\_element].i == key.i and sequence[middle\_element].d==key.d and sequence[middle\_element].s<key.s):  
 start\_element = middle\_element + 1  
 else:  
 end\_element = middle\_element - 1  
 return -1

keyDictionary = {1: lambda x,y:x.i<y.i,  
 2: lambda x,y:x.i < y.i or (x.i == y.i and x.d > y.d),  
 3: lambda x,y:x.i<y.i or(x.i==y.i and x.d>y.d) or (x.i==y.i and x.d==y.d and x.s>y.s)}

class Element:  
 def \_\_init\_\_(self,int , data,str):  
 self.i = int  
 self.d = data  
 self.s = str  
  
 def \_\_str\_\_(self):  
 return str.format("({},{},{})", self.i, self.d,self.s)

import Generate  
from Sort import \*

from ChooseSort import \*  
def insertion\_sort(list1, countKey):  
 countComp = 0  
 countIter = 0  
 for i in range(1, len(list1)):  
 Value = list1[i]  
 Position = i  
 while Position > 0 and keyDictionary[countKey](list1[Position - 1], Value):  
 list1[Position] = list1[Position - 1]  
 Position = Position - 1  
 countComp+=1  
 list1[Position] = Value  
 countIter += 1  
 return countIter+countComp  
def insertion\_sort\_with\_check(list1, countKey):  
 countComp = 0  
 countIter = 0  
 sort=sorted(list1,key=lambda x:(-x.i,x.d,x.s))  
 for i in range(1, len(list1)):  
 if(list1==sort):  
 return countIter + countComp  
 Value = list1[i]  
 Position = i  
 while Position > 0 and keyDictionary[countKey](list1[Position - 1], Value):  
 list1[Position] = list1[Position - 1]  
 Position = Position - 1  
 countComp += 1  
 list1[Position] = Value  
 countIter += 1  
 return countIter + countComp

import random  
import string  
from datetime import date, datetime, timedelta  
from random import randint  
from Element import Element  
def gen\_datetime(min\_year=1900, max\_year=datetime.now().year):  
 start = datetime(min\_year, 1, 1, 00, 00, 00)  
 years = max\_year - min\_year + 1  
 end = start + timedelta(days=365 \* years)  
 return datetime.date(start + (end - start) \* random.random())  
def generate\_random\_string():  
 letters = string.ascii\_lowercase  
 rand\_string = ''.join(random.choice(letters) for i in range(7))  
 return rand\_string  
def gen\_int():  
 return randint(0,1000)  
def generateArray(size):  
 list=[]  
 i=0  
 while i<size:  
 list.append(Element(gen\_int(),gen\_datetime(),generate\_random\_string()))  
 i+=1  
 return list

*Асимптотическая оценка сложности алгоритма:*

O(n^2) – вне зависимости от компаратора для сортировки вставками.

*Наилучшую, наихудшую и среднюю оценки:*

Все оценки равны O(n^2), если только не была описана оптимизация на проверку предварительно отсортированного массива за O(n), в таком случае лучшая оценка будет O(n)

*Псевдокод операции на упрощённом языке:*

*for i = 1 to n-1 do*

*j = i*

*while (j > 0 and A[j] < A[j - 1]) do*

*swap (A[j], A[j - 1])*

*j = j - 1*

*end while*

*end for*

*Выкладки расчета асимптотической оценки:*

Первый цикл проходит O(n) шагов, второй точно также O(n), с учетом того что мы элемент будем вставлять в необходимую позицию. Итого: O(n^2)

*Выкладки или результаты тестирования на различных по объему набору данных:*

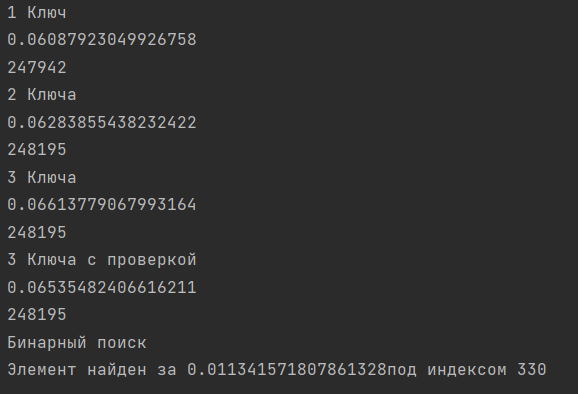


Рисунок 1 – Результат выполнения задания

**Вывод:** изучили основные методы сортировки. Сравнили эффективность методов.