# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования «Новосибирский государственный технический университет»



## Кафедра прикладной математики

## Лабораторная работа № 2 по дисциплине «Методы проектирования и анализа алгоритмов»

**Алгоритмы сортировки**

|  |  |  |
| --- | --- | --- |
|  | Факультет: | ПМИ |
| Группа: | ПМ-53 |
| Студент: | Тябин Е.А. |
| Преподаватель: | Щукин Г.А. |

Новосибирск

2016

**Условия задачи:**

Сортировка вставками, карманная сортировка. Ключ: имя + день рождения

1. **Описание алгоритма**

Формируем список людей из следующего списка составляющих:

* Имя
* Фамилия
* День рождения
* Месяц рождения
* Год рождения

Осуществляем сортировку:

*Сортировка вставками:*

* На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан.
* Сравнение идёт по полю: «имя» и если собираются переставляться элементы с одинаковым именем, сортировка продолжается по полю «день рождения»

*Карманная сортировка:*

* Делим множество входных данных на подмножества по буквам английского алфавита
* Сортируем каждый блок по отдельности с помощью метода **sort**.
* Помещаем отсортированные блоки в единый массив

1. **Текст программы**

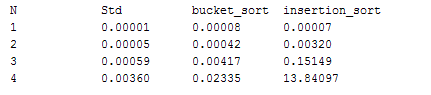
**def** insertion\_sort( array ):  
 array = [p.split(**' '**) **for** p **in** array]  
 **for** i **in** range(len( array )):  
 tmp = array[i]  
 k = i  
 **while** k > 0 **and** tmp[0] <= array[k - 1][0]:  
 **if** tmp[0]==array[k-1][0]:  
 **if** int(tmp[2])<int(array[k-1][2]):  
 array[k]=array[k-1]  
 **else**:  
 array[k] = array[k - 1]  
 k -= 1  
 array[k] = tmp  
 check = list()  
 **for** x **in** array:  
 x = **' '**.join(x)  
 check.append(x)  
 **return** check

**def** bucket\_sort(lst):  
 bucket, bucket1, bucket2 = [], [], [] *# The three empty buckets  
 # Populating the buckets with the list elements* lst = [p.split(**' '**) **for** p **in** lst]  
 **for** i **in** range(len(lst)):  
 **if** lst[i][0][0] **in** [**'A'**, **'B'**, **'C'**, **'D'**, **'E'**, **'F'**, **'G'**, **'H'**, **'I'**]:  
 bucket.append(lst[i])  
 **elif** lst[i][0][0] **in** [**'J'**, **'K'**, **'L'**, **'M'**, **'N'**, **'O'**, **'P'**, **'Q'**, **'R'**]:  
 bucket1.append(lst[i])  
 **elif** lst[i][0][0] **in** [**'S'**, **'T'**, **'U'**, **'V'**, **'W'**, **'X'**, **'Y'**, **'Z'**]:  
 bucket2.append(lst[i])  
 bucket.sort(key=**lambda** k: k[0])  
 bucket1.sort(key=**lambda** k: k[0])  
 bucket2.sort(key=**lambda** k: k[0])  
 final\_lst = bucket+bucket1+bucket2  
 check = list()  
 **for** x **in** final\_lst:  
 x=**' '**.join(x)  
 check.append(x)  
 **return** check

1. **Текст юнит-тестов**

**import** unittest  
**import** DZ  
  
pustos=[]  
  
**class** MyTestCase(unittest.TestCase):  
 **def** test\_DZ(self):  
 self.assertEqual(DZ.insertion\_sort([]), pustos)  
 **def** test\_DZ2(self):  
 self.assertEqual(DZ.bucket\_sort([]), pustos)  
 **def** test\_ins(self):  
 self.assertEqual(DZ.insertion\_sort([**'Keena Mith 19 Dec 1977'**, **'Keena Ya 10 May 1979'**]), [**'Keena Ya 10 May 1979'**,**'Keena Mith 19 Dec 1977'**])  
 **def** test\_ins2(self):  
 self.assertEqual(DZ.insertion\_sort([**'Deirdre Recore 31 Jan 1945'**, **'Andre Londono 20 Sept 1987'**]), [**'Andre Londono 20 Sept 1987'**,**'Deirdre Recore 31 Jan 1945'**])  
 **def** test\_buc2(self):  
 self.assertEqual(DZ.bucket\_sort([**'Deirdre Recore 31 Jan 1945'**, **'Andre Londono 20 Sept 1987'**]), [**'Andre Londono 20 Sept 1987'**, **'Deirdre Recore 31 Jan 1945'**])  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

**4. Результаты замера времени работы алгоритма**



**5. Выводы**

Проделав работу, мы ознакомились со средствами работы Python со списками, а именно реализацией сортировок: вставками и карманной. Замерив время работы алгоритмов, увидели, что карманная сортировка намного превосходит сортировку вставками по времени, оно и верно, ведь в сортировке вставками происходит огромное количество сравнений, что, несомненно, во много раз увеличивает время работы данного алгоритма. Были сделаны юнит-тесты для проверки правильности работы алгоритма. Программа выдала верное решение на всех тестах и, следовательно, правильно работает.