Лабораторная работа № 3

1. Сортировка вставками (Insertion Sorting)

Используя шаблон программы, реализуйте функцию сортировки массива чисел методом вставок.

```
# File: InsertionSort.py
# Реализация алгоритма сортировки вставками
#
# Пример исходных данных и результата:
# [3,6,8,2,9,1,7,0,5,9,4] -> [0,1,2,3,4,5,6,7,8,9,9]
#
# Задача: Дан массив чисел. Необходимо упорядочить его по
# возрастанию с помощью алгоритма сортировки вставками.
#
# Основная идея:
# Из неупорядоченной последовательности элементов поочередно
# выбирается элемент, сравнивается с предыдущим (уже упорядоченным)
# списком и помещается в соответствующее место
#
# Отсортированный
#    участок
#       |
#  <---------->
# [1,2,3,6,8,9,7,0,5,9,4]
#              |
#           Текущий
#           элемент

def insertionsort(L):
   pass # Требуется реализовать функцию

L = [3,6,8,2,9,1,7,0,5,9,4]
print(insertionsort(L)) #Ожидаемый результат [0,1,2,3,4,5,6,7,8,9,9]
```

2. Сортировка слиянием (Mergesort)

Используя шаблон программы, реализуйте функцию сортировки массива чисел методом слияния.

```
# File: Mergesort.py
# Реализация алгоритма сортировки слиянием
#
# Пример исходных данных и результата:
# [3,6,8,2,9,1,7,0,5,9,4] -> [0,1,2,3,4,5,6,7,8,9,9]
#
# Задача: Дан массив чисел. Необходимо упорядочить его по
# возрастанию с помощью алгоритма сортировки слиянием.
#
# Основная идея:
# Рекурсивно разбиваем исходный массив на две части.
# Рекурсия останавливается, когда получившиеся массивы состоят
```

```
# максимум из одного элемента. Такие массивы заведомо отсортированы.
# Получив отсортированные массивы, начинаем обратное объединение
# массивов чисел с помощью алгоритма слияния.
# Алгоритм слияния работает следующим образом.
# Даны два отсортированных массива чисел.
# Например: [1,4,6] и [2,3,5]
# Сравнивая по порядку элементы каждого массива, записываем в
# результат меньшее значение.
# Таким образом выполняем следующие действия:
# [1,4,5], [2,3,6] -> []
# [4,5],   [2,3,6] -> [1]
# [4,5],   [3,5]   -> [1,2]
# [4,5],   [6]     -> [1,2,3]
# [5],     [6]     -> [1,2,3,4]
# [],      [6]     -> [1,2,3,4,5]
# [],      []      -> [1,2,3,4,5,6]
# Сложность шага слияния O(n)

def mergesort(L):
   pass # Реализуйте функцию

L = [3,6,8,2,9,1,7,0,5,9,4]
print(mergesort(L)) # Ожидаемый результат [0,1,2,3,4,5,6,7,8,9,9]
```

3. Формирование входных данных с помощью генератора случайных чисел

Изучите приведенные ниже примеры и выберите способ для создания массива случайных чисел, заданной длины.

Используйте полученный массив для демонстрации работы алгоритмов сортировки.

```
# для работы со случайными числами используется модуль random
import random

print(random.randint(0,10000))

print([int(1000*random.random()) for i in range(10)])

list_size = 15
L = list(range(list_size))
print(L)
random.shuffle(L)
print(L)

for list_size in [5, 10, 20, 30]:
     L = list(range(list_size))
     random.shuffle(L)
     print(L)
```

4. Тестирование алгоритмов

Выполните тестирование двух алгоритмов. Для этого сравните результаты сортировки функций `insertionsort()` и `mergesort()`, используя оператор `assert`.

Если выражение после оператора `assert` ложное, Python выдаст предупреждение `AssertionError` и остановит выполнение программы.

```
assert 1 > 2

L1 = L[:]
L2 = L[:]
assert insertionsort(L1) == mergesort(L2)
```

5. Сравнение скорости работы алгоритмов

Замерьте время работы функций сортировки для массивов разного размера.

Выведите результаты замеров в виде таблицы, где в первом столбце отображается размер входных данных, а в следующих время работы функций в миллисекундах:

```
    10:        0.0        0.0
  1000:      120.1        0.3
  2000:      472.3        0.6
  3000:     1048.9        1.1
  4000:     1942.0        1.4
  5000:     2956.3        1.8
 10000:    12056.4        3.9
```

```
import time # используем модуль time для работы с функциями времени

t1 = time.perf_counter()
insertionsort()
t2 = time.perf_counter()
print("Time sorting: {:.2f} ms".format((t2 - t1) * 1000.))

def print_times(L):
    print("{0:7}".format(len(L)), end=":")
    for func in (insertionsort, mergesort):
        L_copy = L[:]
        t1 = time.perf_counter()
        func(L_copy)
        t2 = time.perf_counter()
        print("{0:10.1f}".format((t2 - t1) * 1000.), end='')
    print()
```

6. Дополнительный материал

Визуализация алгоритмов сортировки
http://www.sorting-algorithms.com