Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение информационных систем и технологий»

**Отчёт**

***по Лабораторной работе №21***

по дисциплине ***«Языки программирования»***

Исполнитель: Колосов Артём Александрович

Группа: 10701219

Преподаватель: Лапанович Ирина Олеговна

2020

Цель работы:

Научиться оценивать эффективность разработанных алгоритмов с использованием Big O нотации, а также изучить простые алгоритмы сортировки элементов и практически закрепить их использование.

Основное задание:

Задан вектор (последовательность) вещественных чисел.

Необходимо отсортировать элементы данного вектора по возрастанию и/или убыванию используя различные алгоритмы. Для реализации сортировки использовать два подхода:

a) использовать собственную реализацию сортировок на базе минимум трёх алгоритмов

b) использовать стандартную сортировку, которая предоставляется стандартной библиотекой языка Python.

Требования к выполнению:

1) Необходимо спроектировать блок-схемы алгоритмов решений соответствую-

щих заданий согласно своему варианту (варианты назначаются преподавате-

лем или выбираются самостоятельно студентом, но ни один вариант не дол-

жен повторяться у студентов из одной группы) и на базе данных алгоритмов

разработать интерактивные консольные приложения с использованием архи-

тектурного шаблона проектирования ***MVC***.

2) Для каждого алгоритма решения задания оценить (вычислить) алгоритмиче-

скую сложность по необходимому времени и затрачиваемой памяти исполь-

зуя определение ***Big O***.

3) Для масштабируемости разрабатываемого программного решения размер

последовательности (списка) и его элементы должны задаваться пользовате-

лем во время выполнения программы или с помощью генератора псевдослу-

чайных чисел.

4) Для автоматизации заполнения различными значениями искомого контей-

нера рекомендуется использовать соответствующие функции генерирования

псевдослучайных чисел модуля ***random***.

5) Рекомендуется избегать использования глобальных переменных при написа-

нии основной логики приложения.

6) Если логически не подразумевается или в заданиях иного не указано, то вход-

ными и выходными данными являются вещественные числа (числа с плаваю-

щей запятой).

7) Все программы должны быть разбиты на отдельные функции. Среди данных

функций рекомендуется добавлять стартовую функцию ***main***, с которой лучше

производить запуск программы.

8) При выполнении заданий необходимо по максимуму пытаться разрабатывать

универсальный, масштабируемый, легко поддерживаемый и читаемый код.

9) Также рекомендуется придерживаться ***Single Responsibility Principle****,* ***SRP***

(принципа единственной ответственности) – постарайтесь вынести основную

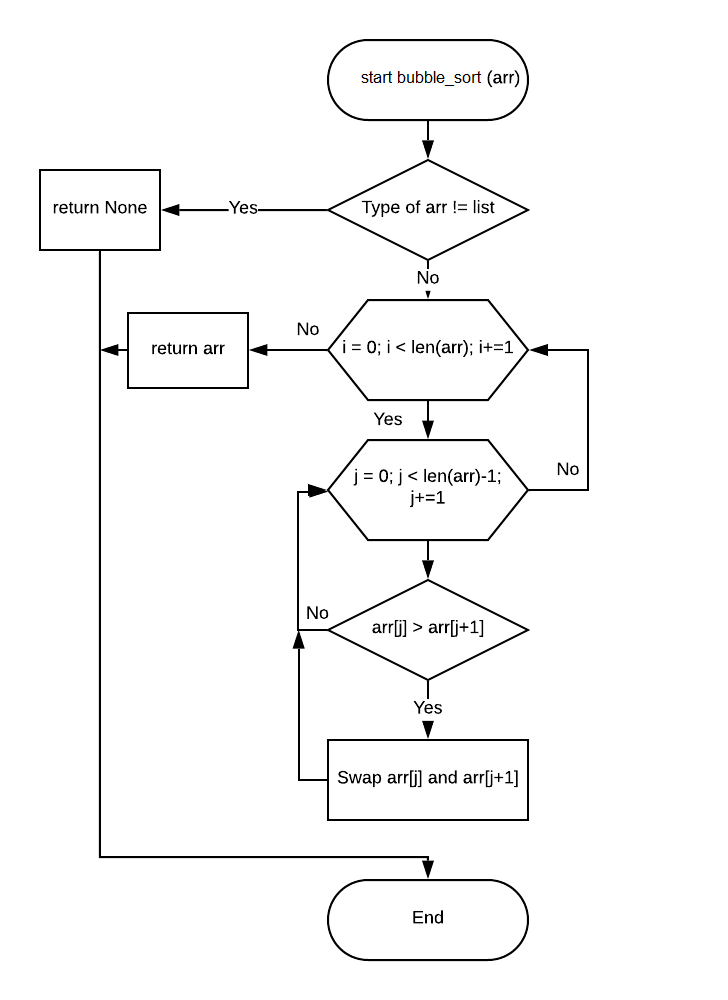
бизнес-логику задания в отдельную функцию (т.е. архитектура приложения

должна минимум состоять из нескольких функций).

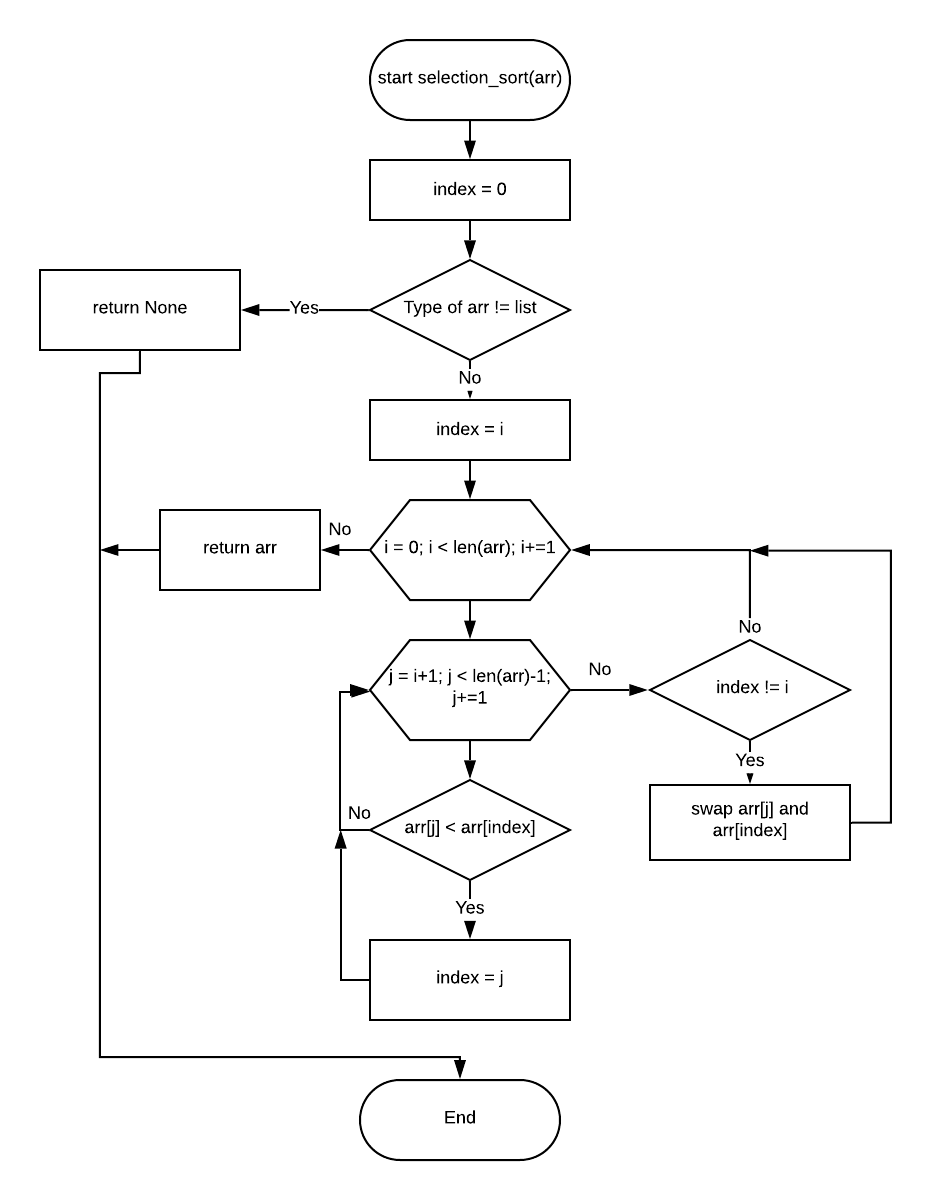
10) В соответствующих компонентах (функциях) бизнес-логики необходимо

предусмотреть «защиту от дурака».

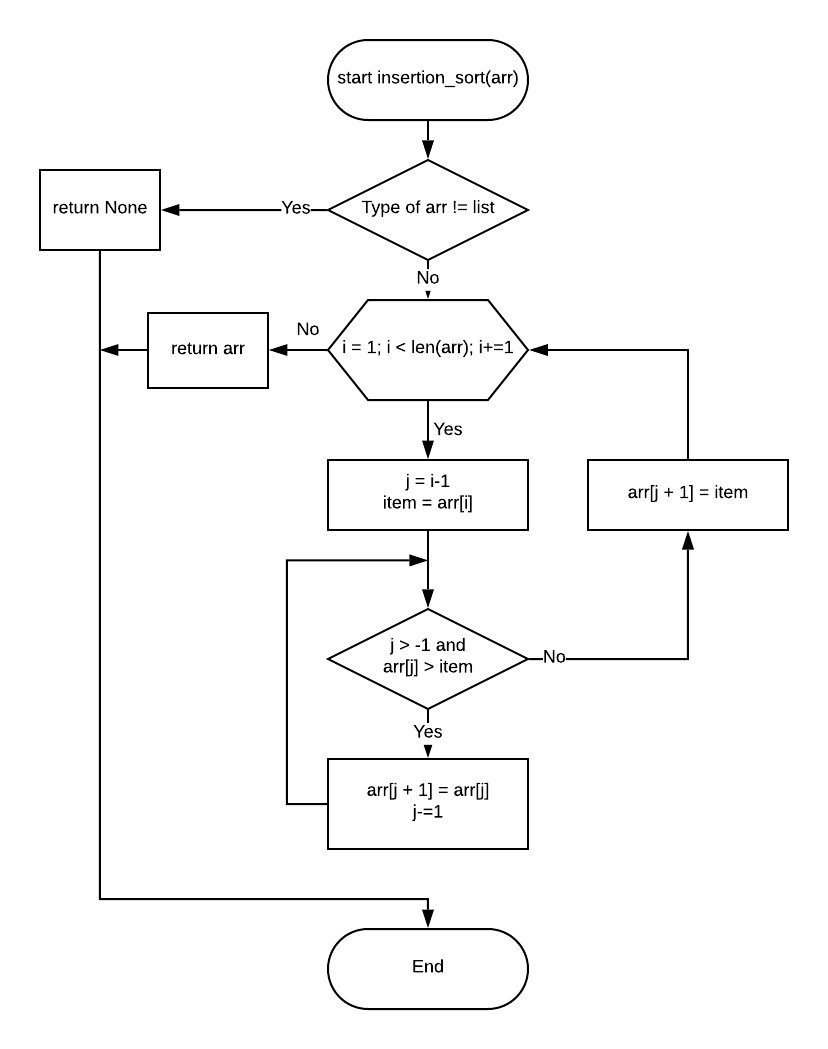
**Блок-схемы основных алгоритмов основного задания**



**Рис 21.1 Блок-схема алгоритма сортировки пузырьком**

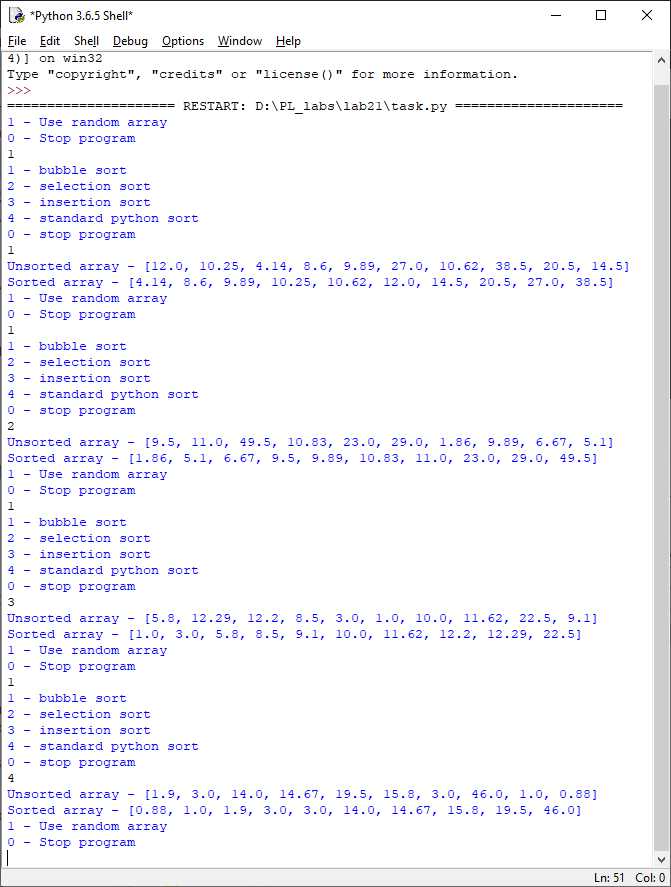


**Рис 21.2 Блок-схема алгоритма сортировки выбором**



**Рис 21.3 Блок-схема алгоритма сортировки вставками**

Результаты выполнения основного задания:



**Рис 21.4 Результат выполнения основного задания**

Оценка алгоритмов по BigO:

Сортировка пузырьком:

Худшее время O(n^2)

Лучшее время O(n)

Среднее время O(n^2)

Затраты памяти O(1)

Сортировка выбором:

Худшее время О(n2)

Лучшее время О(n2)

Среднее время О(n2)

Затраты памяти О(n) всего, O(1) дополнительно

Сортировка вставками:

Худшее время О(n2) сравнений, обменов

Лучшее время O(n) сравнений, O(1) обмен

Среднее время О(n2) сравнений, обменов

Затраты памяти О(n) всего, O(1) вспомогательный

Ответы на контрольные вопросы:

1.Назовите один из фундаментальных принципов одной из

крупных компании по разработке аппаратного и программного

обеспечения IBM.

*Машина должна работать, человек — думать.*

2. Зачем нужно уметь оценивать сложность алгоритма? (Назовите

минимум три причины.)

1. *Чтобы узнать скорость выполнения*
2. *Чтобы классифицировать и оценивать алгоритмы по степени сложности*
3. *Для предварительного оценивания затрачиваемых ресурсов*

3. Что в математике задаёт функция?

*В математике функция обозначает какую-то либо зависимость.*

4. Какая функция (нотация) используется для записи оценки сложности

алгоритма при программировании?

*Для оценки сложности алгоритмов в программировании создали специальное обозначение под названием Big O (“большая О”).*

5. Дайте определение Big O нотации. Что вообще показывает Big O нотация?

*Нотация Big O является одним из самых фундаментальных инструментов анализа сложности алгоритма.*

*Big O позволяет оценить, насколько время выполнения алгоритма зависит от переданных в него данных.*

6. Какие бывают разновидности подсчёта оценки сложности алгоритма?

*Факториальная N!*

*Экспоненциальная K^n*

*Полиноминальная N^k*

*Линейный логарифм N\*log(N)*

*Линейная (K\*N)*

*Логарифмическая (K\*log(N))*

*Константная K*

7. Почему основная функция (зависимость) оценки сложности алгоритма не

учитывает время выполнения программы?

*Потому что время выполнения программы зависит от компьютера, на котором она выполняется . И так как мощности компьютеров отличаются, то брать время выполнения было бы не корректно*

8. Что берётся за неизвестное в Big O нотации?

*Количество входных данных*

9. Что такое отбрасывание констант при оценки алгоритмической сложности?

Приведите пример расчёта.

*Отбрасывание констант нужно для того, чтобы отбросить ненужные переменные, которые не зависят от кол-ва переданных алгоритму переменных*

*O(n^2 + 15) -- > O(n^2)*

10. Что такое неважная сложность при оценки алгоритмической сложности и как

с ней поступают? Приведите пример расчёта.

*Неважная сложность – это тот компонент, который является несущественным при вычислении сложности и его можно откинуть.*

*O(n^2 + n\*log(n) + 11n) -> O(n^2)*

11. Приведите наиболее часто встречаемые зависимости при оценки сложности

алгоритмов. Какие из них показывают наиболее эффективные реализации

алгоритмов? Приведите примеры таких алгоритмов.

*O(n) — линейная сложность*

*O(log n) — логарифмическая сложность*

*O(n2) — квадратичная сложность*

12. Какая зависимость растёт быстрее остальных и является показателем самого

неэффективного решения из возможных?  
 3n

13. Как вычисляются оценки эффективности алгоритмов в лучшем случае,

среднем и худшем случаях? Что эти случаи обозначают?

*Лучший случай обозначает самые лучшие из возможных входных данных, а*

*Худший случай наоборот, самые плохие.*

14. В каких ситуациях при оценки алгоритмической сложности необходимо

складывать, а когда умножать сложности выполнения?

*Умножение происходит если у нас есть несколько циклов*

15. На что при оценки алгоритмической сложности показывает зависимость

log(N)? Приведите примеры алгоритмов, в которых входит данная

зависимость.

*Алгоритм «бинарный поиск»: делим массив на две половины, отбрасываем не нужную, оставшуюся опять делим на две части и так пока не найдем нужное значение. Такой тип алгоритмов называется «разделяй и влавствуй» Divide and Conquer.*

Что я освоил в процессе выполнения лабораторной работы (выводы):

1. Научился оценивать эффективность разработанных алгоритмов с использованием Big O нотации.
2. Изучил простые алгоритмы сортировки элементов и практически закрепил их использование.

*ПРИЛОЖЕНИЕ A*

Листинг исходных кодов программ

Файл task1.py

Файл task.py

from random import randint

# Программа реализует несколько алгоритмов сортировки

#

# Version: 1.0

# Group: 10701219

# Author: Колосов Артём Александрович

# Date: 2.28.2020

def get\_input(msg=""):

return input(str(msg))

def output\_to\_user(msg="", s="", e="\n"):

print(msg, sep=s, end=e)

def random\_arr(n, start=1, stop=100):

return [round(randint(start, stop) / randint(start, round(stop / 10)), 2) for i in range(n)]

def bubble\_sort(arr):

if not isinstance(arr, list):

return None

for i in range(len(arr)):

for j in range(len(arr) - 1):

if arr[j] > arr[j + 1]:

arr[j + 1], arr[j] = arr[j], arr[j + 1]

return arr

def selection\_sort(arr):

if not isinstance(arr, list):

return None

index = 0

for i in range(len(arr) - 1):

index = i

for j in range(i + 1, len(arr)):

if arr[j] < arr[index]:

index = j

if index != i:

arr[i], arr[index] = arr[index], arr[i]

return arr

def insertion\_sort(arr):

if not isinstance(arr, list):

return None

for i in range(1, len(arr)):

j = i - 1

item = arr[i]

while j > -1 and arr[j] > item:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = item

return arr

def main():

while True:

command = get\_input("1 - Use random array\n"

"0 - Stop program\n")

if command == "1":

arr = random\_arr(10, start=1, stop=100)

command = get\_input("1 - bubble sort\n"

"2 - selection sort\n"

"3 - insertion sort\n"

"4 - standard python sort\n"

"0 - stop program\n")

output\_to\_user("Unsorted array - {0}".format(arr))

if command == '1':

arr = bubble\_sort(arr)

elif command == '2':

arr = selection\_sort(arr)

elif command == '3':

arr = insertion\_sort(arr)

elif command == '4':

arr = sorted(arr)

elif arr == '0':

break

else:

output\_to\_user("Incorrect input!")

continue

output\_to\_user("Sorted array - {0}".format(arr))

elif command == "0":

break

if \_\_name\_\_ == "\_\_main\_\_":

main()