Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

Выпускная квалификационная работа бакалавра

«Метод автоматического управления памятью с гарантированным временем выполнения на основе подсчёта ссылок»

Студент: Сапожков Андрей Максимович ИУ7-83Б

Научный руководитель: Строганов Юрий Владимирович

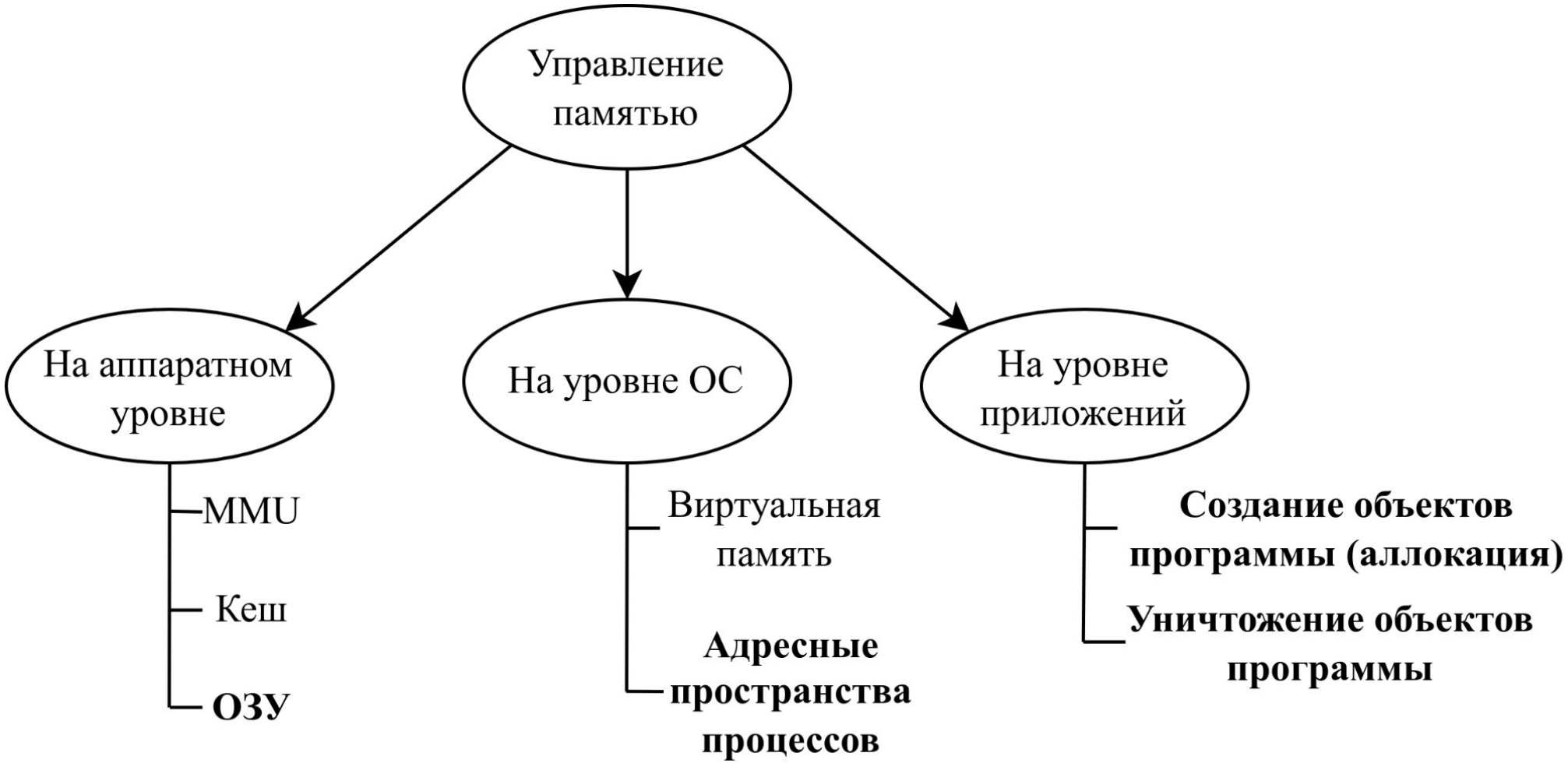
Цель и задачи

Цель – разработка метода автоматического управления памятью с гарантированным временем выполнения на основе подсчёта ссылок.

Задачи:

1. Проанализировать и классифицировать существующие методы распределения памяти в языках программирования с автоматической сборкой мусора.
2. Классифицировать алгоритмы по требованиям к дополнительной памяти.
3. Спроектировать и реализовать метод автоматического управления памятью.
4. Определить области применения реализованного метода.

Управление памятью



Управление памятью на уровне приложений

- Ручное, определяемое человеческим фактором.
- **Автоматическое**, определяемое **алгоритмами распределения памяти**.

```
int main(void) {
    int *a = new int;
    // ...
    *a = 5;
    // ...
    delete a;
}
```

Ручное управление памятью (C++)

```
package main
func main() {
    a := new(int)
    // ...
    *a = 5
    // ...
}
```

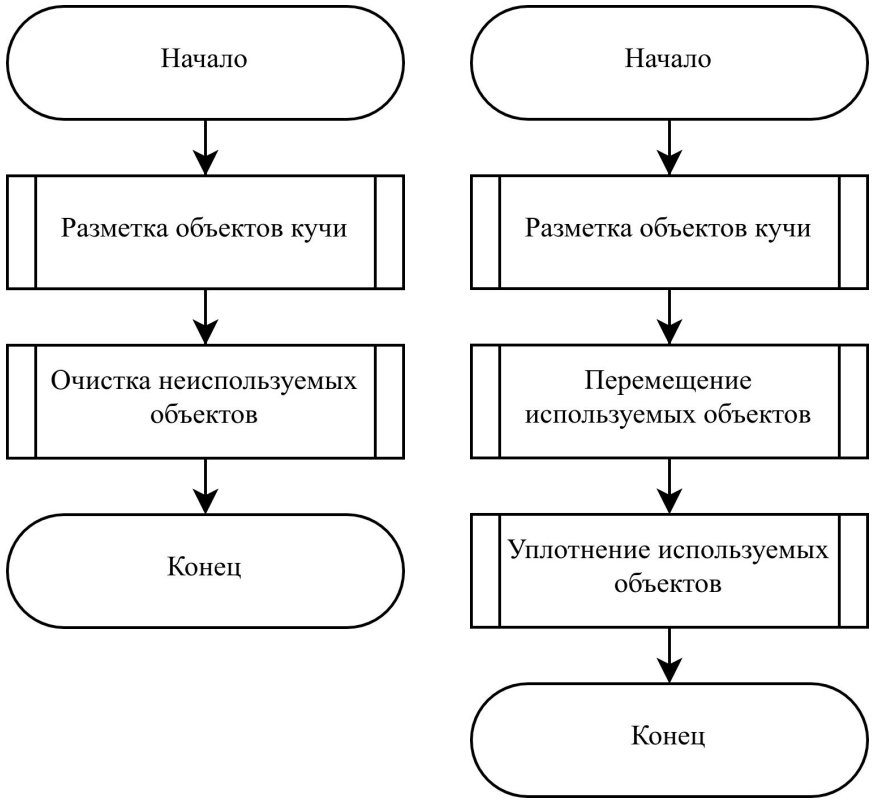
Автоматическое управление памятью (Golang)

Сборка мусора

Это автоматическая переработка динамически выделяемой памяти.

Трассирующая сборка мусора

mark-sweep mark-compact



Подсчёт ссылок

\+ более равномерное распределение накладных расходов;
\+ устойчивость к высоким нагрузкам;
\+ масштабируемость по размеру кучи;
\+ не требует среды выполнения языка;

\- накладные расходы на чтение/запись;
\- атомарность операций;
\- не выявляются циклы ссылок.

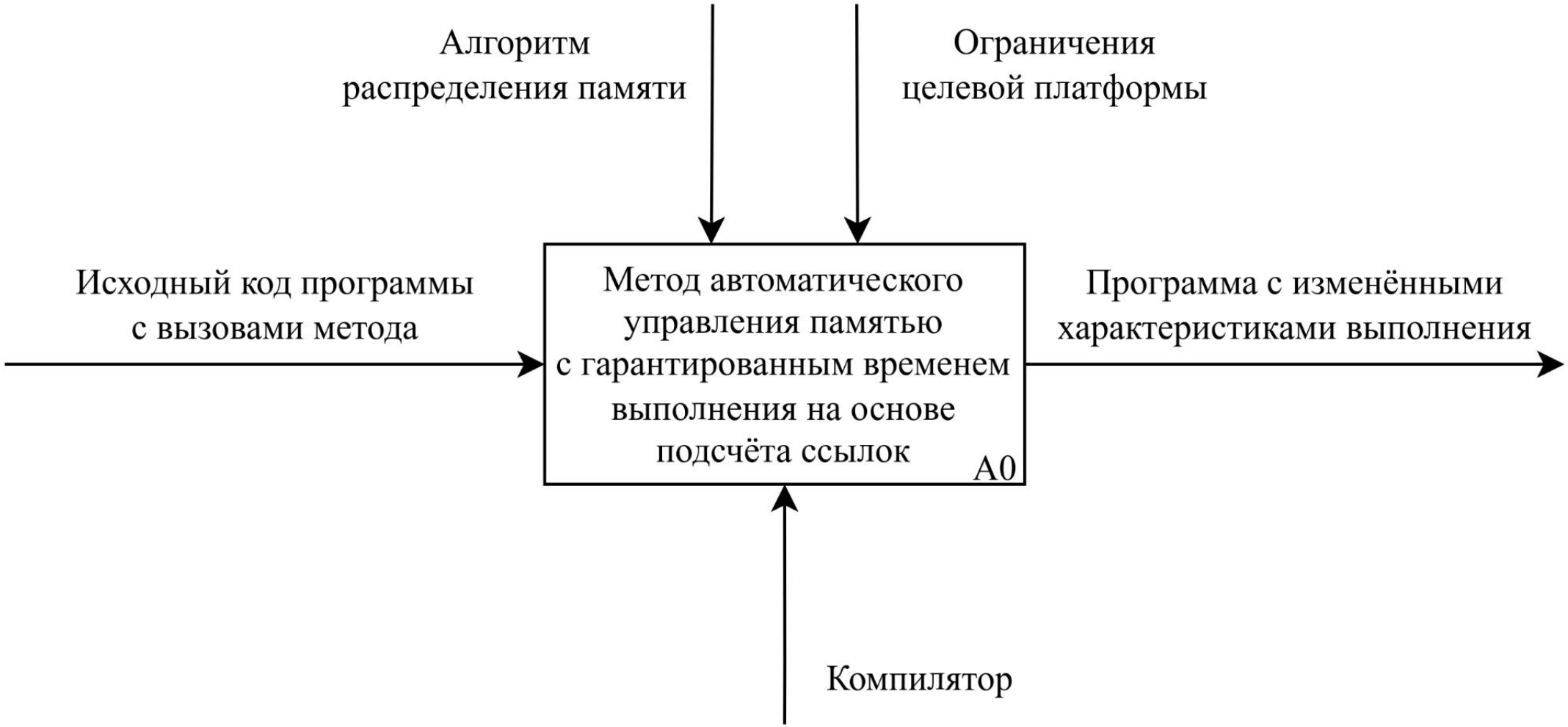
Классификация существующих решений (1/2)

Язык программирования	Сборщик мусора	Использование поколений	Отсутствие хранения доп. данных в объектах	Конкурентная сборка мусора
Python	**По умолчанию**	+	**-**	+
Java	**Serial**	+	+	+
	Parallel	+	+	+
	Garbage-First	+	+	+
	ZGC	+	+	+
JavaScript	По умолчанию	-	+	-
C#	По умолчанию	+	+	+
Golang	По умолчанию	-	+	+

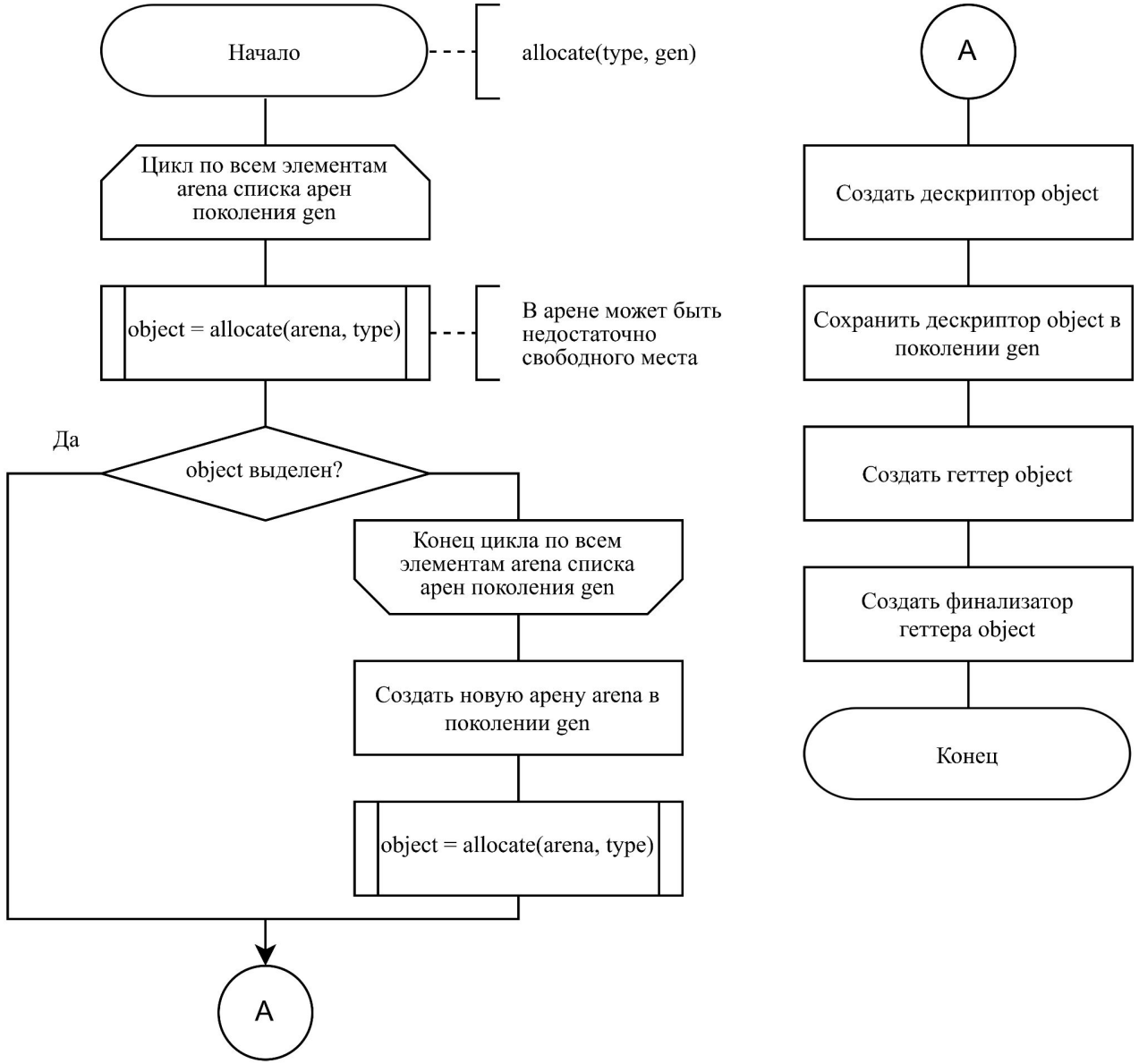
Классификация существующих решений (2/2)

Язык программирования	Сборщик мусора	Параллельная сборка мусора	Остановка программы не на весь цикл сборки	Количество остановок программы за один цикл сборки
Python	**По умолчанию**	**-**	+	**1**
Java	**Serial**	**-**	**-**	**1**
	Parallel	+	**-**	**1**
	Garbage-First	+	+	**2**
	ZGC	+	+	**1**
JavaScript	По умолчанию	-	-	1
C#	По умолчанию	+	-	1
Golang	По умолчанию	+	+	2

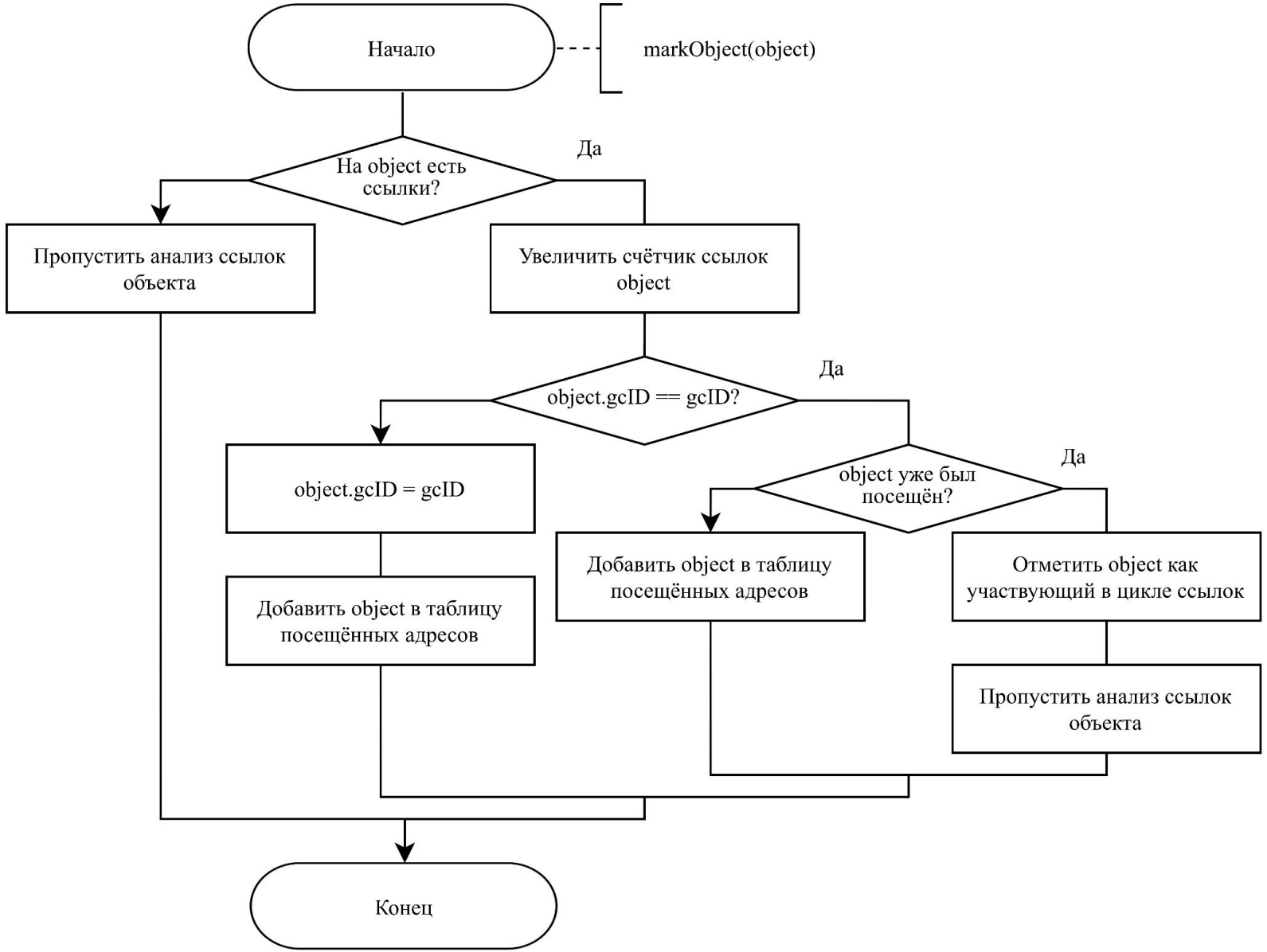
Предлагаемый метод



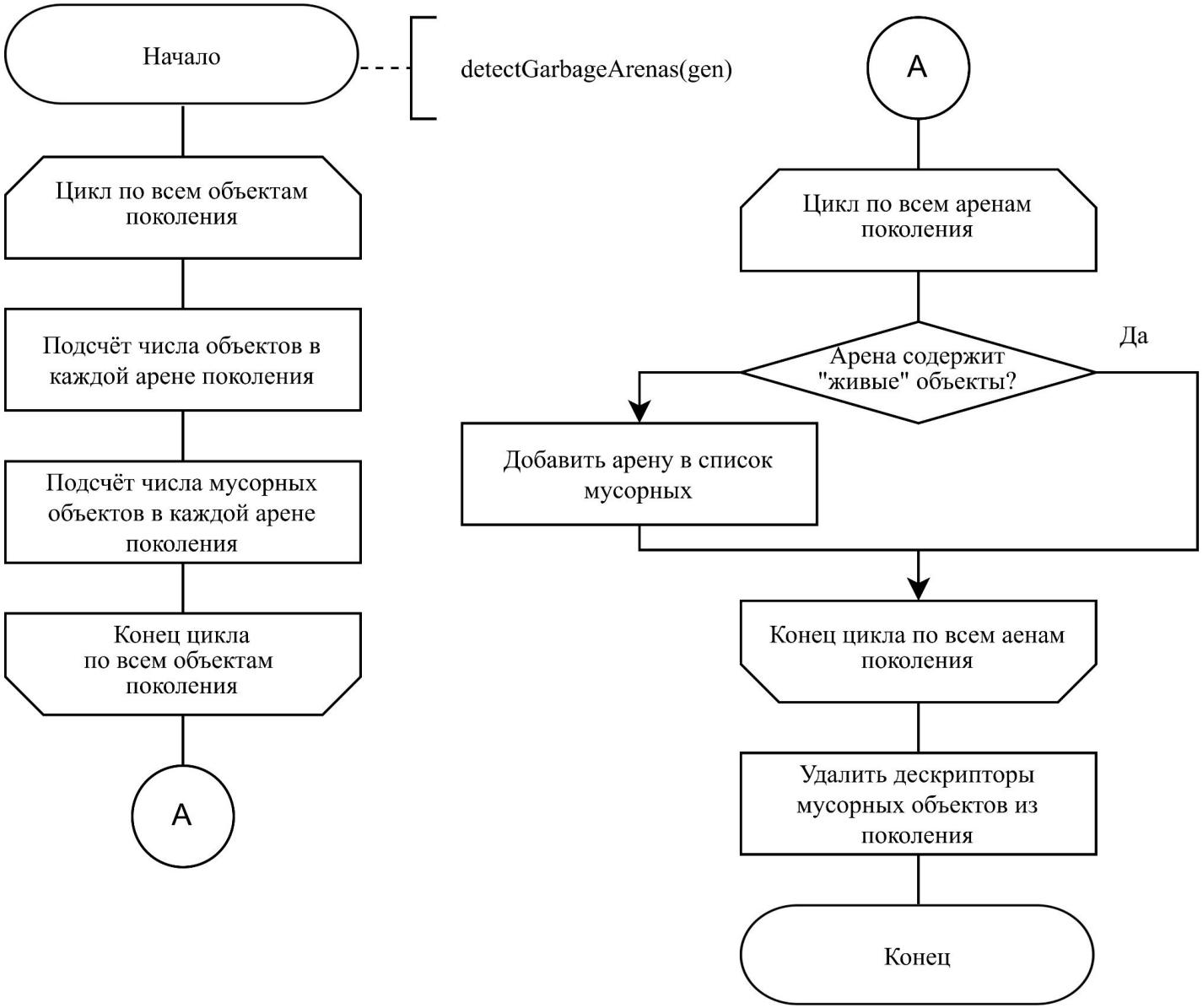
Алгоритм выделения памяти



Алгоритм разметки



Алгоритм очистки



Использование метода

```
> go get github.com/Inspirate789/alloc
```

```
package main

func main() {
    object := new(int)
    println(*object)
    *object = 7
    println(*object)
}
```

Встроенные функции Golang

```
package main
import "github.com/Inspirate789/alloc"
func main() {
    object := alloc.New[int]()
    println(*object.Get())
    *object.Get() = 7
    println(*object.Get())
}
```

Реализованный метод

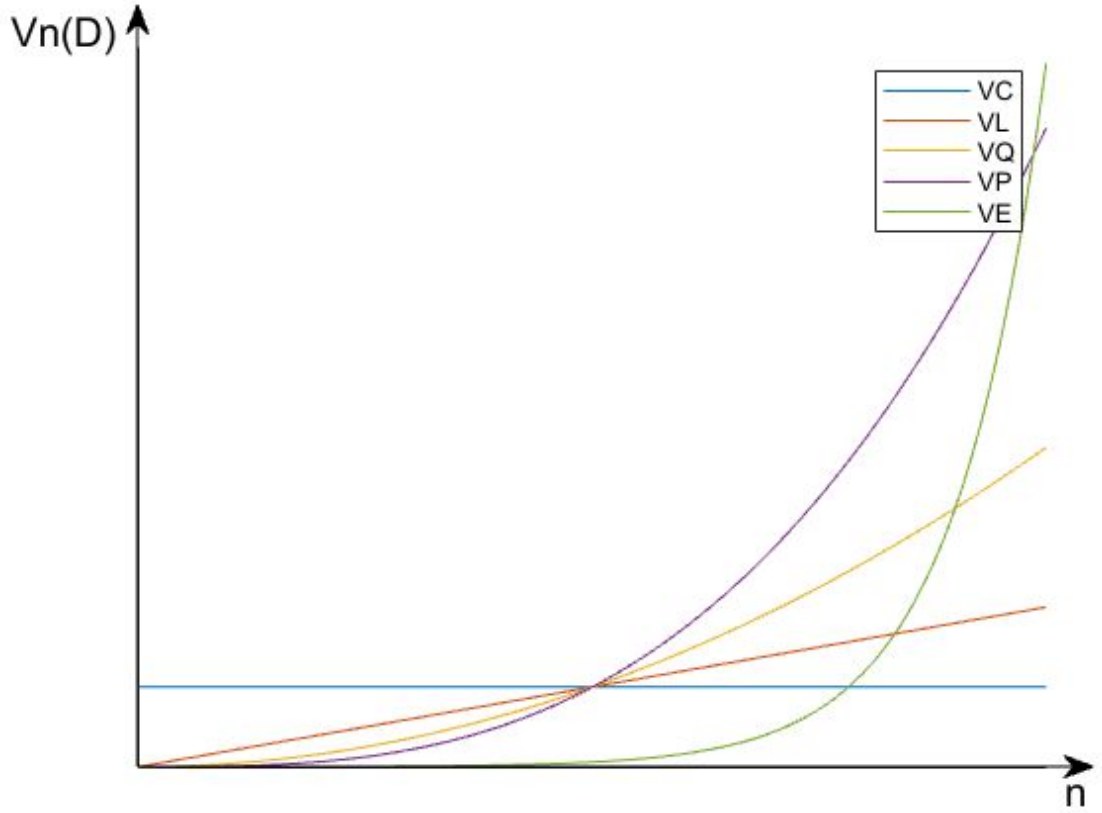
Классификация алгоритмов

Пусть D_n – вход алгоритма A, $V_t(D)$ – дополнительная память алгоритма A,

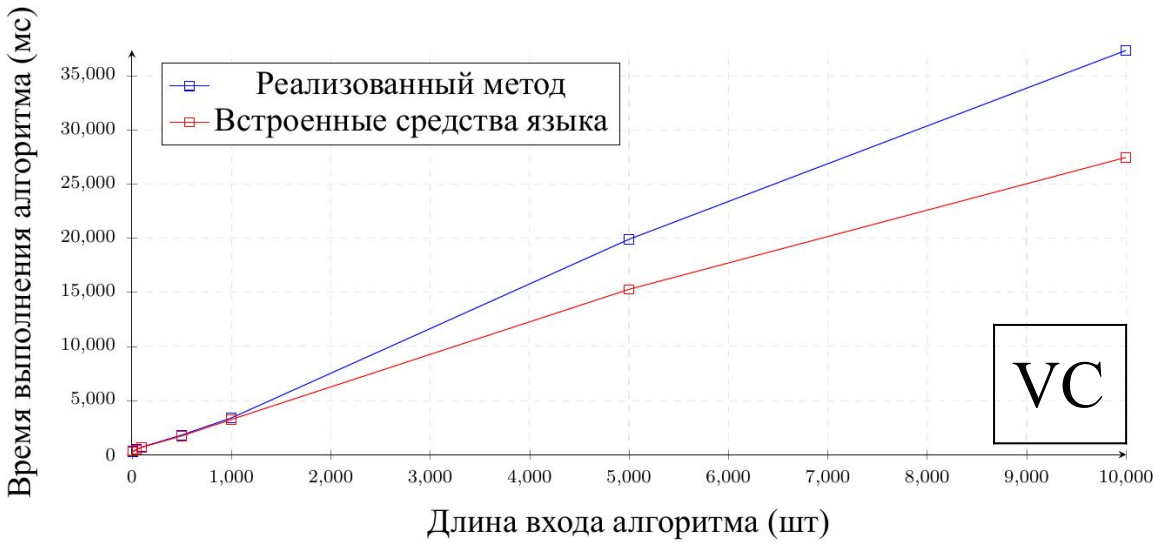
$V_t\hat{}(n) = \max V_t(D)$ – дополнительная память алгоритма A в худшем случае для всех входов длины n.

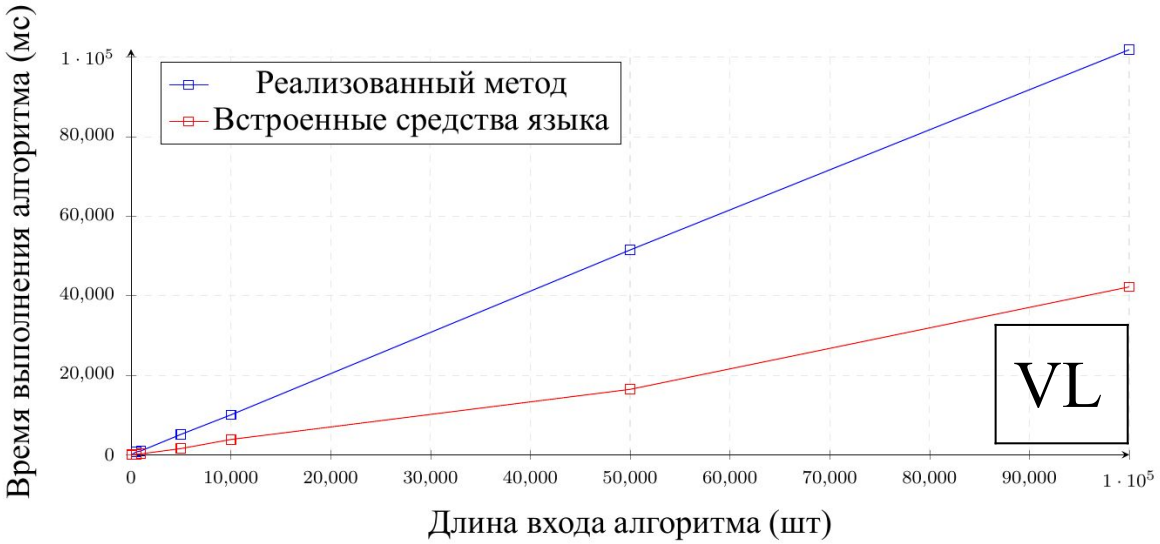
Классы алгоритмов по требованиям к дополнительной памяти:

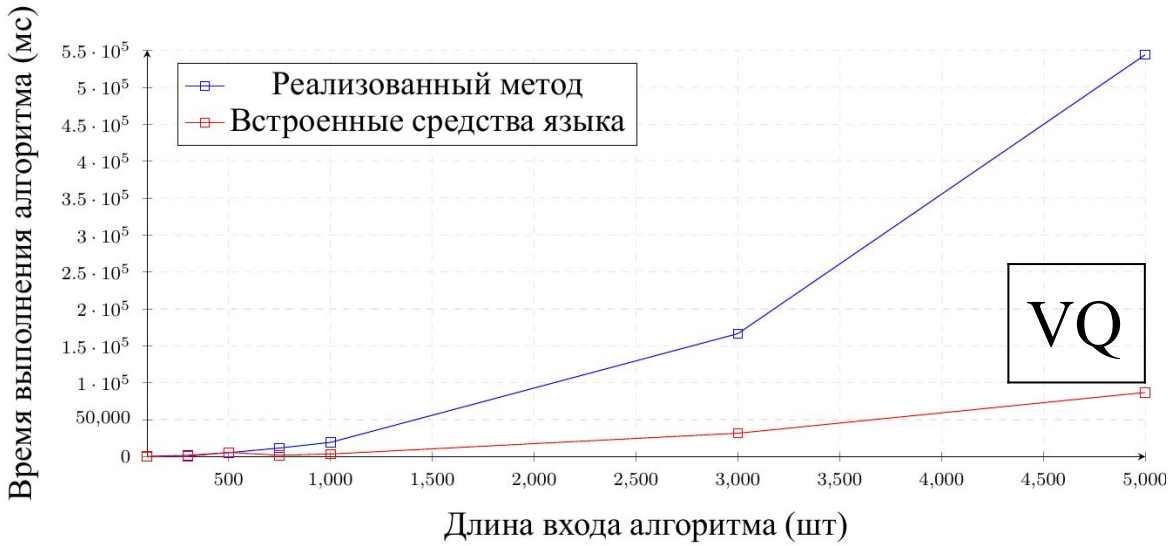
- **V0**: $V_t(D) = 0$.
- **VC**: $V_t\hat{}(n) = const \neq 0$.
- **VL**: $V_t\hat{}(n) = \Theta(n)$.
- **VQ**: $V_t\hat{}(n) = \Theta(n^2)$.
- **VP**: $V_t\hat{}(n) = \Theta(n^k)$, $k > 2$.
- **VE**: $V_t\hat{}(n) = \Theta(e^{\lambda n})$, $\lambda > 0$.



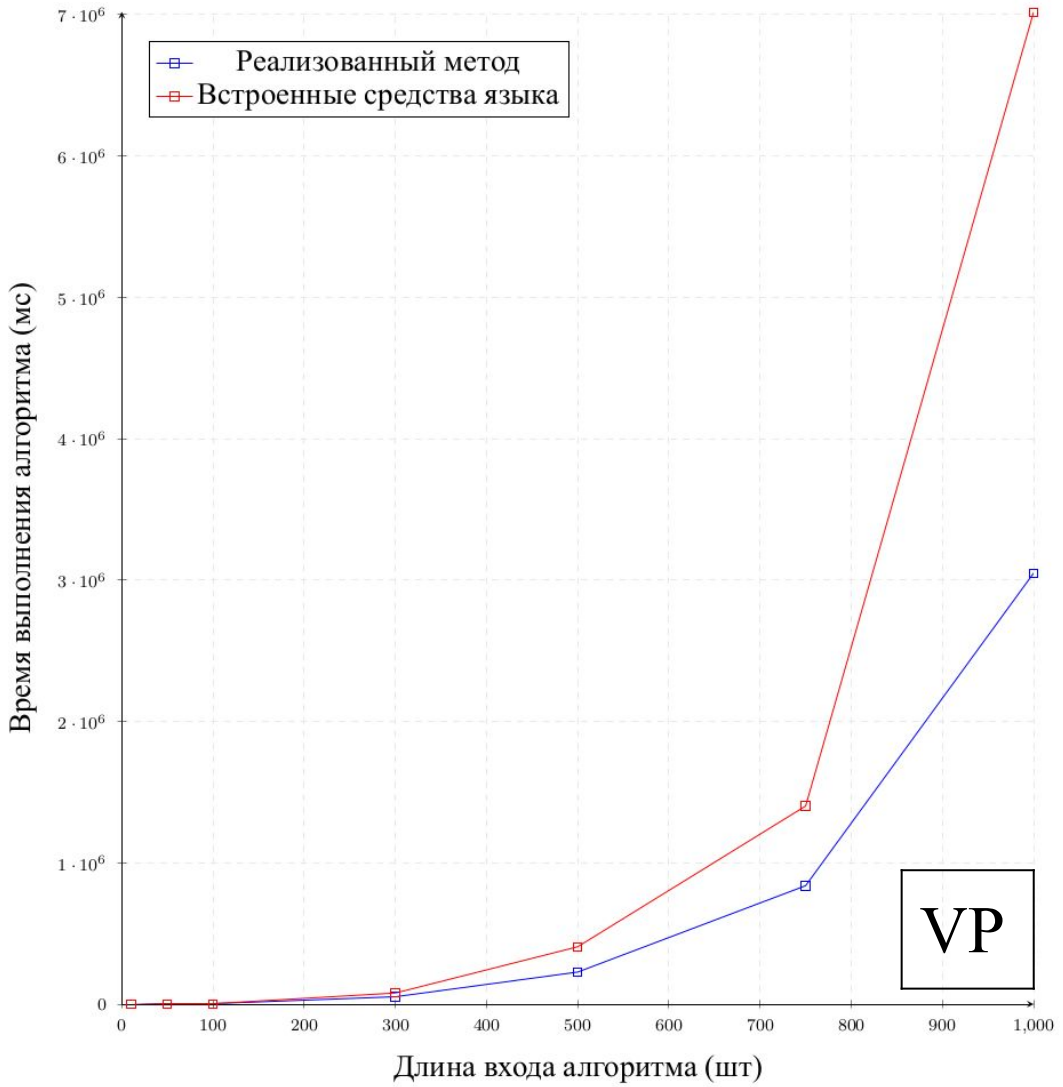
Результаты исследования (1/2)

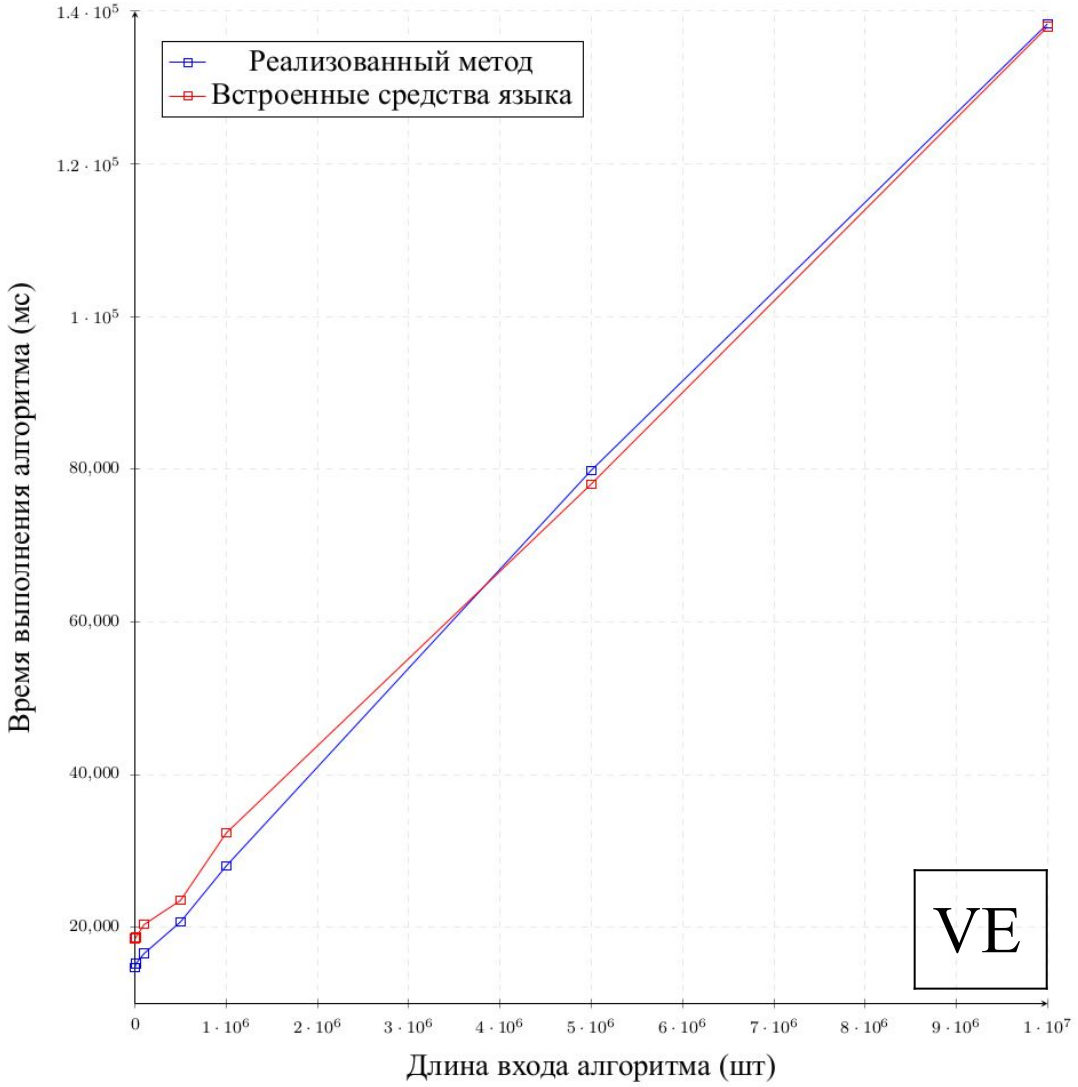






Результаты исследования (2/2)





Заключение

Цель работы достигнута: был разработан и реализован метод автоматического управления памятью. Все поставленные задачи были выполнены:

- Проанализированы существующие методы распределения памяти в языках программирования с автоматической сборкой мусора.
- Проведена классификация алгоритмов по требованиям к дополнительной памяти: выделены классы **V0, VC, VL, VQ, VP и VE**.
- Спроектирован и реализован метод автоматического управления памятью.
- Сравнительный анализ выявил области применения реализованного метода: алгоритмы класса **VP и VE**.

Дальнейшее развитие

- Исследование стабильности менеджера памяти.
- Исследование фрагментации кучи при использовании метода.
- Внедрение метода во встроенный сборщик мусора языка Golang.