общих технологических вызовов, характерных для подобных цифровых экосистем.

Во-первых, это обеспечение изоляции выполнения кода: платформы должны гарантировать, что решения участников не вмешиваются в работу основной системы и не нарушают её целостность, что требует применения контейнеризации, песочниц и строгих политик безопасности.

Во-вторых, необходима детерминированность условий проверки: одинаковый алгоритм должен получить аналогичный результат при любых условиях запуска, что требует стандартизованных сред выполнения и чётко определённых ограничений по времени и памяти.

В-третьих, важнейшим вызовом является масштабируемость архитектуры: системы обязаны обеспечивать надёжную работу при пиковых нагрузках, что предполагает автоматическое распределение нагрузки между вычислительными узлами, оптимизацию очередей заданий и грамотное управление ресурсами [1].

Наконец, интеграция аналитических и социальных функций, таких как рейтинги, форумы и образовательные материалы, требует от архитектуры гибкости и расширяемости, поскольку подобные компоненты должны не только работать автономно, но и поддерживать консистентное взаимодействие с основными модулями системы проверки.

Практика показала, что неправильная декомпозиция системы на сервисы или пренебрежение стандартами среды выполнения приводит к трудноуловимым ошибкам воспроизводимости и к рискам безопасности, тогда как продуманная модульность упрощает эволюцию функционала и снижает стоимость сопровождения.

Масштабируемость остаётся критическим фактором – как в аспекте обработки пиковых нагрузок во время массовых контестов, так и в обеспечении низкой задержки отклика для интерактивных задач – и требует внедрения адаптивных стратегий балансировки, продвинутых политик очередей и мониторинга загрузки. Кроме того, мультиформатность современных соревнований и потребность в поддержке множества языков программирования ставят дополнительные требования к унификации интерфейсов компиляции и исполнения, к стандартизации ограничений по времени и памяти, а также к обеспечению репродуцируемости тестовых сред.

Наконец, интеграция социальных, рейтинговых и образовательных компонентов в соревновательную инфраструктуру предъявляет высокие требования к согласованности данных и гибкости взаимодействия подсистем: платформа должна не только точно оценивать корректность решений, но и поддерживать прозрачную обратную связь, аналитические отчёты и инструменты для обучения участников.

Исходя из этих наблюдений, целесообразно направлять дальнейшие инженерные усилия на развитие модульных архитектур с явной границей ответственности сервисов, стандартизацию сред выполнения и тестовых спецификаций, внедрение механизмов автоматического масштабирования и продвинутых средств наблюдения за поведением системы; параллельно необходимы исследования по формализации критериев детерминированности и воспроизводимости в контексте соревновательной оценки, а также по разработке методик экспериментального сравнения архитектурных решений в реальных нагрузочных сценариях.

Такое сочетание практических улучшений и методологического осмысления будет способствовать созданию более надёжных, безопасных и педагогически ценных платформ, способных удовлетворять растущие требования сообществ программистов и образовательных учреждений.

Список литературы

1. Евстропов Г. О. Системы оценивания в задачах с автоматической проверкой решений // Информационный журнал по олимпиадам и программированию. Информатика и образование. 2016. № 3. С. 65-67.

2. Иртегов Д. В., Нестеренко Т. В., Чурина Т. Г. Системы автоматизированной оценки заданий по программированию: разработка, использование и перспективы // Вестник Новосибирского государственного университета. Серия: Информационные технологии. 2019. Т. 17, № 2. С. 61-73.

3. Корнеев Г. А., Елизаров Р. А. Автоматическое тестирование решений на соревнованиях по программированию // Телекоммуникации и информатизация образования. 2003. № 1. С. 61-73.

4. Материалы VIII Всероссийской с международным участием научно-практической конференции «День спортивной информатики», (Москва, 2 декабря 2024 года) / Ассоциация компьютерных наук в спорте. Москва, 2024. 132 с.

ПУЗЫРЬКОВАЯ СОРТИРОВКА: СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМА НА СОВРЕМЕННЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ В ЧИТАЕМОСТИ КОДА

Кузнецова В. А.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: skald.beregovoi@yandex.ru

Научный руководитель: Береговой А. В.

Введение

Пузырьковая сортировка (Bubble Sort) – один из старейших алгоритмов сортировки, известный еще с середины XX века. Его название, впервые введенное в работе Айверсона (Iverson) в 1962 году, точно отражает суть процесса: элементы с наибольшими (или наименьшими) значениями постепенно «всплывают» к своему законному месту в конце (или начале) массива, подобно пузырькам воздуха в воде. [1]

Цель исследования – на основе различных современных языков программирования (C++,

Java, JavaScript, Go и Python) произвести сравнительный анализ представлений визуальное представление пузырькового метода сортировки.

Материалы и методы исследования

В статье исследуется алгоритм, его способы оптимизации и реализация на различных языках программирования: C++, Java, JavaScript, Go и Python, сравнение разных параметров кода, в особенности его читаемость и простота понимания.

Результаты исследования и их обсуждение

Базовый принцип работы: Алгоритм многократно проходит по массиву, попарно сравнивая соседние элементы. Если два соседа расположены в неверном порядке относительно выбранного критерия (например, по возрастанию), алгоритм меняет их местами. Этот процесс повторяется до тех пор, пока весь массив не будет упорядочен, то есть пока не будет выполнен полный проход без единой перестановки. В обширном арсенале алгоритмов сортировки, где доминируют высокоэффективные методы вроде быстрой сортировки (QuickSort), сортировки слиянием (MergeSort) и пирамидальной сортировки (HeapSort), пузырьковая сортировка занимает особую, почти архаичную нишу.

Рассмотрим детальную пошаговую работу алгоритма на конкретном примере. Возьмем массив целых чисел, который требуется отсортировать по возрастанию: `[5, 3, 8, 4, 2]`.

Первый проход (i = 0):

– Сравнение индексов 0 и 1: `5 > 3` → истина → меняем местами. Массив: `[3, 5, 8, 4, 2]`
– Сравнение индексов 1 и 2: `5 > 8` → ложь → массив без изменений: `[3, 5, 8, 4, 2]`
– Сравнение индексов 2 и 3: `8 > 4` → истина → меняем местами. Массив: `[3, 5, 4, 8, 2]`
– Сравнение индексов 3 и 4: `8 > 2` → истина → меняем местами. Массив: `[3, 5, 4, 2, 8]`

Результат первого прохода: Наибольший элемент (8) «всплыл» на свою конечную позицию. Неотсортированная часть массива теперь `[3, 5, 4, 2]`.

Второй проход (i = 1):

– Сравнение индексов 0 и 1: `3 > 5` → ложь → массив без изменений: `[3, 5, 4, 2, 8]`
– Сравнение индексов 1 и 2: `5 > 4` → истина → меняем местами. Массив: `[3, 4, 5, 2, 8]`
– Сравнение индексов 2 и 3: `5 > 2` → истина → меняем местами. Массив: `[3, 4, 2, 5, 8]`

Результат второго прохода: Второй по величине элемент (5) занял свою позицию. Неотсортированная часть: `[3, 4, 2]`.

Третий проход (i = 2):

– Сравнение индексов 0 и 1: `3 > 4` → ложь → массив без изменений.
– Сравнение индексов 1 и 2: `4 > 2` → истина → меняем местами. Массив: `[3, 2, 4, 5, 8]`

Результат третьего прохода: Элемент 4 занял свою позицию. Неотсортированная часть: `[3, 2]`.

Четвертый проход (i = 3):

– Сравнение индексов 0 и 1: `3 > 2` → истина → меняем местами. Массив: `[2, 3, 4, 5, 8]`

Результат четвертого прохода: Массив полностью отсортирован. Алгоритм завершает работу.

Базовая версия алгоритма неэффективна, так как продолжает работу даже после того, как массив стал отсортированным. Существует две ключевые оптимизации:

1. Флаг обменов (Swapped Flag): На каждом проходе отслеживается, был ли совершен хотя бы один обмен. Если внутренний цикл завершился без единой перестановки, это означает, что массив уже отсортирован, и внешний цикл можно прервать досрочно. Именно эту версию мы будем реализовывать в данной статье.

2. Учет последней перестановки (Last Swap Index): Более продвинутая оптимизация. Вместо флага запоминается индекс последней выполненной перестановки во внутреннем цикле. На следующей итерации внутренний цикл будет выполняться только до этого индекса, а не до фиксированного `n – i – 1`. [3]

Теперь, имея теоретическое понимание алгоритма, перейдем к практической части. Мы реализуем оптимизированную версию с флагом обменов на пяти различных языках. Наша цель – не сравнение производительности (где компилируемые языки, такие как C++ и Go, заведомо выигрывают у интерпретируемого Python), а анализ читаемости, лаконичности кода.

Реализация на C++[6]

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
 vector<int> v={64,34,25,12,22,11,90};
 for(int i=0,n=v.size();i<n-1;i++){
```

```
 bool s=0;
 for(int j=0;j<n-i-1;j++)
  if(v[j]>v[j+1]) swap(v[j],v[j+1]),s=1;
 if(!s) break;}
cout<<»Отсортированный массив: «;
 for(int n:v) cout<<n<<» «;
 return 0;}
```

Анализ кода на C++:

– Управление памятью и типами: Явное указание типов обеспечивает безопасность и предсказуемость, но добавляет «синтаксического шума».

– Процесс обмена элементов: Классический трехшаговый обмен с временной переменной `temp` является универсальным, но выглядит громоздко и не является идиоматическим для современного C++.

– Читаемость: Код достаточно читаем для опытного разработчика, но может показаться перегруженным новичку из-за необходимости разбираться с заголовочными файлами, областью видимости (`std::`) и ручным управлением итерациями.

Реализация на Java

```
public class B{public static void main(String[]a){
int[]n={64,34,25,12,22,11,90},t=n.clone();
for(int i=0,l=t.length;i<l-1;i++){
boolean s=false;
for(int j=0;j<l-i-1;j++)
if(t[j]>t[j+1]){int x=t[j];t[j]=t[j+1];t[j+1]=x;s=true;}
if(!s)break;}
System.out.print(«Отсортированный массив: «);
for(int x:t)System.out.print(x+» «);}}
```

Анализ кода на Java:[4]

– Объектно-ориентированный каркас: Необходимость помещать код в класс `BubbleSort` и использовать модификатор `static` для простой процедурной функции добавляет церемониальности, которая не связана напрямую с логикой алгоритма.

– Синтаксическая схожесть с C++: Java унаследовала много синтаксических конструкций от C/C++, включая объявление переменных, циклы `for` и условные операторы `if`. Это делает код очень похожим на C++ версию со всеми ее достоинствами и недостатками.

– Читаемость: Для разработчиков, знакомых с C-подобными языками, код интуитивно понятен.

Реализация на JavaScript [7]

```
function bubbleSort(arr) {
   let n = arr.length;
   let swapped;
   for (let i = 0; i < n – 1; i++) {
      swapped = false;
      for (let j = 0; j < n – i – 1; j++) {
         if (arr[j] > arr[j + 1]) {
            [arr[j], arr[j + 1]] = [arr[j + 1], arr[j]];
            swapped = true;}}
      if (!swapped) {
         break;}}
   return arr;}
const numbers = [64, 34, 25, 12, 22, 11, 90];
console.log(«Исходный массив:», numbers);
console.log(«Отсортированный массив:»,
bubbleSort(numbers));
```

Анализ кода на JavaScript:

– Динамическая типизация: Отсутствие объявлений типов делает код менее многословным, но может скрывать потенциальные ошибки, которые в статически типизированных языках были бы выявлены на этапе компиляции.

– Деструктурирующее присваивание: Это мощная особенность современного JavaScript (ES6+), которая кардинально меняет читаемость критической части алгоритма – обмена элементов. Конструкция `[a, b] = [b, a]` интуитивно понятна и выразительна.

– Функциональный стиль: Возврат массива из функции (`return arr`) позволяет использовать вызов в выражениях, например, сразу в `console.log`, что удобно.

– Читаемость: Благодаря деструктуризации и лаконичному синтаксису, код на JavaScript становится одним из самых читаемых среди C-подобных языков.

Реализация на Go[5]

```
package main
import «fmt»
func bubbleSort(arr []int) {
   n := len(arr)
   var swapped bool
   for i := 0; i < n-1; i++ {
      swapped = false
      for j := 0; j < n-i-1; j++ {
         if arr[j] > arr[j+1] {
            arr[j], arr[j+1] = arr[j+1], arr[j]
            swapped = true}}
      if !swapped {
         break}}}
func main() {
   numbers := []int{64, 34, 25, 12, 22, 11, 90}
   fmt.Println(«Исходный массив:», numbers)
   bubbleSort(numbers)
            fmt.Println(«Отсортированный массив:», numbers)}
```

Анализ кода на Go:

– Простота и минимализм: Синтаксис Go намеренно прост и лишен многих особенностей других языков (нет классов, исключений в классическом понимании, перегрузки операторов). Это делает код последовательным и предсказуемым.

– Статическая типизация с лаконичностью: Как и в C++/Java, типы указываются явно (`[]int`),

что обеспечивает безопасность. Однако краткая форма объявления переменных (`n := len(arr)`) и автоматический вывод нулевых значений делают код менее многословным.

– Параллельное присваивание: Способ обмена элементов `a, b = b, a` является в Go идиоматическим и столь же элегантным, как и в Python.

– Читаемость: Код на Go исключительно читаем. Отсутствие лишних скобок и ключевых слов, а также встроенный обмен значениями ставят Go очень высоко в рейтинге читаемости.

```
 Python
def bubble_sort(arr):
   n = len(arr)
   for i in range(n – 1):
      swapped = False
       range(n – i – 1)
      for j in range(n – i – 1):
         if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
            swapped = True
      if not swapped:
         break
   return arr
```

Анализ и итоговое сравнение

1. Синтаксис как псевдокод (Максимальная читаемость):

– Код на Python практически неотличим от формального псевдокода, используемого для описания алгоритмов в учебниках. Ключевые слова `def`, `for...in range`, `if not` читаются как предложения на английском языке.

– Сравнение: C++ и Java требуют объявления типов и управления областью видимости с помощью фигурных скобок `{}`. Python использует обязательные отступы, которые одновременно являются и синтаксической конструкцией, и гарантией хорошо отформатированного, «красивого» кода.

2. Элегантность в критических операциях (Параллельное присваивание):

– Конструкция `arr[j], arr[j+1] = arr[j+1], arr[j]` является идиоматической для Python и решает одну из самых частых задач в алгоритмах – обмен значений – с беспрецедентной ясностью. Она интуитивно понятна даже тем, кто видит ее впервые.

– Сравнение: C++ и Java вынуждены использовать трехшаговый обмен с временной переменной, который является более низкоуровневым и заставляет мозг тратить дополнительные усилия на его «разбор». JavaScript и Go имеют схожие элегантные конструкции, что ставит их в один ряд с Python по этому параметру.

3. Выразительность и лаконичность:

– Объявление списка `numbers = [64, 34, ...]` предельно просто.

– Создание копии для чистоты эксперимента `.copy()` – это один ясный метод.

– Сравнение: Python делает максимум с минимумом синтаксических элементов. В нем почти нет «церемониального» кода, не связанного напрямую с решаемой задачей.

4. Динамическая типизация и полиморфизм:

– Хоть динамическая типизация и может быть источником ошибок, в контексте учебных алгоритмов она является благом. Одна и та же функция `bubble_sort` может работать с любыми типами данных, для которых определен оператор сравнения `>` (числа, строки и т.д.), без необходимости писать шаблоны или перегружать функции.

– Сравнение: В C++ и Java для этого потребовалось бы использовать шаблоны (C++) или дженерики (Java) или перегружать метод для разных типов, что значительно усложнило бы учебный пример.

5. Баланс между абстракцией и контролем:

– Python, будучи языком высокого уровня, эффективно абстрагируется от низкоуровневых деталей (управление памятью, явные типы переменных в теле функции), позволяя программисту сосредоточиться на сути алгоритма – его логике. Это идеально для образовательных целей.

Заключение

Проведенный анализ однозначно показывает, что приоритеты в программировании могут быть разными. Если конечной целью является максимальная производительность вычислений для сортировки миллионов записей, то компилируемые языки, такие как C++ и Go, будут вне конкуренции.

Однако если мы говорим о педагогической ценности, скорости прототипирования, сопровождаемости кода и, что самое главное, о чистой читаемости и выразительности, то Python демонстрирует безоговорочно лучший результат. Пузырьковая сортировка, будучи простым алгоритмом, служит идеальным полигоном для демонстрации этой философской разницы. Python-реализация превращает ее из набора инструкций для машины в ясное и лаконичное описание алгоритма для человека.

Преимущество Python заключается в его способности минимизировать когнитивную нагрузку на программиста, позволяя ему сосредоточиться на решении задачи, а не на преодолении синтаксических барьеров языка. И в этой дисциплине он был и остается эталоном, на фоне которого даже такие изящные и современные языки, как Go и JavaScript, хоть и приближаются к нему, но все же уступают в чистоте и ясности изложения мысли.

Список литературы

1. Колтыгин Д. С., Баева А. Ю. Сравнительный анализ быстродействия методов сортировки массивов URL: https://www.elibrary.ru/item.asp?id=43030420 (дата обращения: 15.12.2025).

2. Осипов А. А. Сравнение скорости сортировки (Bubble, Quick, Merge, Heap) // Перспективы развития науки и мирового сообщества: научно-методические и практические аспекты: материалы IX Международной научно-практической конференции. Физико-математические науки. 2024. URL: https://innova-science.ru/wp-content/uploads/2025/06/sbornik-nauchnyh-trudov-19.06.2025-prs-9.pdf#page=54 (дата обращения: 15.12.2025).

3. Валиева Э. Р., Лябах К. Ю., Мазяков А. В. Методы сортировок и их эффективность // Актуальные проблемы интеграции науки и образования в регионе: материалы Всероссийской научно-практической конференции (с международным участием) (г. Бузулук, 20–21 мая 2021 года). Бузулук: Оренбург. гос. ун-т, 2021. С. 227–231.

4. Реализация пузырьковой сортировки на Java // Javarush. URL: https://javarush.com/groups/posts/634-realizacija-puzihrjhkovoy-sortirovki-na-java (дата обращения: 15.12.2025).

5. Сортировка пузырьком (Go) // Expange.ru. URL: https://expange.ru/e/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC_(Go) (дата обращения: 15.12.2025).

6. Bubble Sort in C++ // Code of Code. URL: https://codeofcode.org/lessons/bubble-sort-in-cpp/ (дата обращения: 15.12.2025).

7. Bubble Sort: изучаем самый простой алгоритм сортировки // Proglib.io. URL: https://proglib.io/p/bubble-sort (дата обращения: 15.12.2025).

КОНФИДЕНЦИАЛЬНОСТЬ ПЕРСОНАЛЬНЫХ ДАННЫХ В СОЦИАЛЬНЫХ СЕТЯХ: РИСКИ И СПОСОБЫ ИХ МИНИМИЗАЦИИ

Кузьменко Р. Д.

ФГБОУ ВО «Мелитопольский государственный университет», Мелитополь, e-mail: anna.dyachenko597@mail.ru

Научный руководитель: Дяченко А. С.

Введение

Социальные сети в современном обществе занимают центральное место в системе коммуникации, распространения информации и организации повседневной деятельности пользователей [1]. Их активное развитие сопровождается ростом объёма обрабатываемых персональных данных, что существенно повышает риски нарушения конфиденциальности [2]. В условиях цифровизации утечки информации, взломы пользовательских аккаунтов и несанкционированный доступ к персональным данным становятся одной из наиболее значимых угроз информационной безопасности [3]. Актуальность проблемы обусловлена ростом числа пользователей и усилением последствий киберинцидентов, что требует комплексного анализа рисков и разработки эффективных мер их минимизации.

Материалы и методы исследования

Анализ научных публикаций показывает, что проблема защиты персональных данных в социальных сетях рассматривается в условиях цифровой трансформации и роста объёмов обрабатываемой информации. Исследователи отмечают, что социальные платформы являются приоритетной целью киберпреступности из-за концентрации персональных данных и высокой активности пользователей [2]. Современные работы подчёркивают многофакторный характер рисков конфиденциальности, при котором технические уязвимости усиливаются поведенческими и организационными факторами, что обосновывает необходимость комплексного подхода к защите персональных данных.

Целью данного исследования является анализ рисков нарушения конфиденциальности персональных данных пользователей социальных сетей, их классификация, а также определение наиболее эффективных способов минимизации выявленных угроз.

Результаты исследования и их обсуждение

В ходе исследования были проанализированы статистические данные об утечках персональных данных, а также обобщены типовые риски, характерные для социальных сетей [3, 4]. Полученные результаты позволили не только выявить масштабы проблемы, но и систематизировать основные виды угроз конфиденциальности.

Для количественной оценки динамики утечек персональных данных была проведена сравнительная характеристика показателей за 2024–2025 годы (таблица 1) [4].

Анализ статистики показывает, что при незначительном снижении числа инцидентов наблюдается резкое увеличение объёма утёкших данных, что свидетельствует об укрупнении масштабов отдельных утечек и росте потенциального ущерба для пользователей [4].

Таблица 1

Динамика утечек персональных данных в России в 2024-2025 гг.

Показатель	2024 год	2025 год
Количество зарегистрированных инцидентов утечек	310	281
Объём утёкших персональных данных (млрд строк)	3,5	≈ 13
Темп изменения объёма утечек	—	рост в 3,7 раза
Основные цифровые источники утечек	Социальные сети, онлайн-сервисы	Социальные сети, мессенджеры