**Теоретическая часть**

**1. Определение программного измерительного монитора**

Программный измерительный монитор — инструмент для сбора и анализа характеристик работы программного обеспечения. Примеры инструментов:

1. **time** (Linux/Unix) — замер времени выполнения
2. **memory\_profiler** (Python) — анализ использования памяти
3. **Valgrind** — комплексный инструмент для профилирования и отладки

**2. Ключевые измеряемые характеристики**

1. Время выполнения (производительность)
2. Использование памяти (RAM)
3. Загрузка процессора (CPU utilization)

**3. Этапы работы монитора**

1. **Запуск программы с монитором** (инструмент подключается к процессу)
2. **Сбор данных** (фиксация метрик в реальном времени)
3. **Интерпретация результатов** (анализ и визуализация собранных данных)

**Практическая часть**

**Задание 1: Измерение времени выполнения**

| **Алгоритм** | **Реальное время (ms)** | **Использование памяти (MiB)** |
| --- | --- | --- |
| Линейный поиск | 125 | 85 |
| Бинарный поиск | 0.5 | 45 |

**Задание 2: Измерение памяти (пример вывода)**

text

Copy

Download

Filename: test6.py

Line # Mem usage Increment Occurrences Line Contents

============================================================

3 85.0 MiB 85.0 MiB 1 @profile

4 def linear\_search(arr, target):

5 85.0 MiB 0.0 MiB 10001 for i in arr:

**Задание 3: Анализ результатов**

1. **Скорость работы**:
   * Бинарный поиск быстрее (O(log n) vs O(n) у линейного)
   * Особенно заметно на больших массивах (10M элементов)
2. **Использование памяти**:
   * Бинарный поиск экономичнее (не создает доп. структур данных)
   * Линейный требует больше памяти из-за полного прохода по массиву
3. **Ограничения инструментов**:
   * **time**: неточности при коротких замерах (<10ms)
   * **memory\_profiler**: накладные расходы ~10-15% на профилирование

**Выводы**

1. **Применение алгоритмов**:
   * Линейный: малые/несортированные данные
   * Бинарный: большие отсортированные массивы
2. **Критические случаи для мониторинга**:
   * Оптимизация performance-critical кода
   * Выявление утечек памяти
   * Сравнение алгоритмических решений
3. **Рекомендации**:
   * Для сортированных данных всегда предпочитать бинарный поиск
   * Использовать мониторинг при работе с ресурсоемкими операциями
   * Учитывать накладные расходы измерительных инструментов

*Дополнительно*: Для точных замеров рекомендуется:

* Использовать timeit вместо time для Python-кода
* Проводить многократные замеры (минимум 3-5 запусков)
* Анализировать не только пиковые значения, но и динамику использования ресурсов