**Теоретическая часть (краткие ответы)**

1. **Определение:**  
   Что такое *программный измерительный монитор*? Приведите три примера инструментов.

Программный измерительный монитор — это инструмент или система, предназначенная для отслеживания и измерения различных параметров программного обеспечения в реальном времени. Три примера инструментов — Nagios, Zabbix, Prometheus.

1. **Задачи мониторинга:**  
   Перечислите 3 ключевые характеристики ПО, которые можно измерить с помощью мониторов.

Три ключевые характеристики ПО, которые можно измерить с помощью мониторов:

* Производительность (например, время отклика, пропускная способность)
* Нагрузка и использование ресурсов (CPU, память, диск)
* Надежность и доступность (статус работы, время простоя)

1. **Этапы работы:**  
   Опишите последовательность действий: запуск программы с монитором → сбор данных → интерпретация результатов.

* Запуск программы с монитором: интеграция или запуск мониторингового инструмента вместе с программой для отслеживания.
* Сбор данных: монитор собирает показатели и метрики в реальном времени или по расписанию.
* Интерпретация результатов: анализ собранных данных для выявления проблем, оценки состояния системы и принятия решений по оптимизации или устранению неисправностей.

**Практическая часть (работа с инструментами)**

**Задание 1: Измерение времени выполнения**

1. Сгенерируйте **отсортированный массив** из 10 000 000 случайных чисел.
2. Замерьте время выполнения для поиска **крайнего элемента**  последнего в массиве) с помощью:
3. *# Для Windows:*
4. Measure-Command {python test6.py} *# Линейный*
5. Measure-Command {python test7.py}*# Бинарный*
6. Результаты занесите в таблицу:

| **Алгоритм** | **Реальное время(ms)** | **Использование памяти** |
| --- | --- | --- |
| Линейный поиск | ~450 ms | 38 MiB |
| Бинарный поиск | ~0.1 ms | 45 MiB |

**Задание 2: Измерение использования памяти**

1. Установите модуль memory\_profiler:

pip install memory\_profiler

1. Добавьте в код декораторы для замера памяти:

from memory\_profiler import profile

@profile

def linear\_search(arr, target): ...

@profile

def binary\_search(arr, target): ...

1. Запустите каждый скрипт и запишите **пиковое использование памяти** (MiB).

Линейный поиск - 38 MiB

Бинарный поиск - 45 MiB

**Задание 3: Анализ результатов**

1. Какой алгоритм работает **быстрее** на больших данных? Объясните, почему.

Бинарный. Делит массив пополам

1. Какой алгоритм использует **меньше памяти**? Почему?

Линейный. Не требует дополнительных структур данных.

1. Какие **ограничения** имеют использованные инструменты (time, memory\_profiler)?

Time — нет детализации по функциям

Memory\_profiler - добавляет накладные расходы, что могут искажать результаты.