1. Встановіть Valgrind у Docker-образі з компілятором GCC, створіть просту програму на C і перевірте правильність встановлення (valgrind --version, valgrind ./a.out).

2. Створіть Dockerfile, що автоматично встановлює Valgrind, компілює програму та запускає її під Valgrind.

3. Створіть короткий скрипт для компіляції gcc -g та запуску valgrind з найважливішими опціями (--leak-check=full, --track-origins=yes).

4. Реалізуйте Test Case #0 — "Hello world!" без помилок. Переконайтесь, що Valgrind не видає жодного попередження.

5. Test Case #1: реалізуйте і відловіть доступ до неініціалізованої змінної (наприклад, локальної int).

6. Test Case #5: реалізуйте переповнення при читанні масиву фіксованого розміру.

7. Test Case #6: реалізуйте переповнення при читанні в динамічному масиві (malloc).

8. Test Case #8: Use-after-free. Виділіть пам’ять, звільніть, спробуйте записати в неї.

9. Test Case #9: Use-after-return. Поверніть адресу локальної змінної з функції.

10. Test Case #13: створіть витік пам’яті, викликавши функцію з бібліотеки, яка виділяє пам’ять (наприклад, strdup) і не звільняється.

11. Порівняйте звіти Valgrind з параметрами --leak-check=no, summary-only, --track-origins=yes. Опишіть, які саме помилки приховуються або виявляються.

12. Побудуйте Valgrind Summary Table (таблицю), в яку включено тип помилки, приклад коду, вихід Valgrind та спосіб виправлення.

13. Напишіть сценарій, що виконує всі test cases під Valgrind та створює звіт по кожному з них.

14. Скомпілюйте програму з підтримкою ASan (-fsanitize=address -g) і порівняйте з Valgrind по швидкості та виходу.

15. Реалізуйте Test Case #1 з неініціалізованим читанням і перевірте, як ASan показує джерело помилки.

16. Test Case #2: Напишіть програму з write overflow в масиві фіксованого розміру.

17. Test Case #3: Напишіть програму з write overflow в динамічному масиві.

18. Test Case #8: Use-after-free. Порівняйте звіт ASan із Valgrind.

19. Test Case #9: Use-after-return. Спробуйте запустити ASan з опцією detect_stack_use_after_return=1.

20. Зберіть ASan summary table: тип помилки, приклад, реакція інструмента, час виконання.

21. Напишіть короткий звіт (до 1 сторінки): “Valgrind vs ASan — що краще для великих програм? Для CI/CD? Для роботи з Docker?”

22. Зробіть експеримент з великою програмою (наприклад, сортування 1 млн елементів) з помилкою пам’яті та порівняйте навантаження Valgrind і ASan.

23. Реалізуйте інтеграцію обох інструментів у CI-пайплайн через GitHub Actions або shell-скрипт.

24. Використайте mallopt() для зміни політики виділення пам’яті та перевірте вплив на фрагментацію.

25. Створіть конфігурацію програми з різними параметрами M_MXFAST, M_TRIM_THRESHOLD, M_TOP_PAD, заміряйте вплив.

26. Дослідіть вплив M_CHECK_ACTION=2 (glibc feature) на виявлення помилок з malloc/free.

27. Напишіть програму, яка викликає malloc_stats() після кожного основного кроку, і проаналізуйте ефективність розподілу.