Процес перескладання

Під час перескладання для отримання заліку необхідно (у дужках жирним шрифтом наведено мінімальні вимоги для отримання заліку, їх потрібно виконати у всіх трьох пунктах):

1. Продемонструвати теоретичні знання, відповівши на всі питання з блоку 1.1, а також по одному випадково вибраному питанню з блоків 1.2-1.4 **(необхідно правильно відповісти на питання 1.1, і на два із трьох питань із блоків 1.2-1.4)**.
2. Показати практичні вміння реалізації алгоритмів і структур даних мовою Python. Це буде відбуватись таким чином: із розділу 2 будуть випадковим чином вибрані два завдання, треба реалізувати одне з них на вибір **(необхідно правильно реалізувати відповідну задачу)**.
3. Довести складність одного з алгоритмів із розділу 3, питання буде вибрано випадковим чином **(необхідно правильно виконати доведення і пояснити)**.

1. Теоретичні питання

1.1 Загальні питання

1. Що таке алгоритм і структура даних? Що таке складність алгоритму, в чому вона вимірюється, що таке О-нотація, що означає “n” в О-нотації, які бувають складності, що таке складність по часу і по пам’яті? Що таке найкращий / середній / найгірший випадок роботи алгоритма відносно складності?

1.2 Базові алгоритми / структури даних / підходи

2. Що таке пошук, які ви знаєте алгоритми пошуку, як вони працюють?
3. Що таке рекурсія? Наведіть приклади алгоритмів, де вона використовується.
4. Що таке випадкове перемішування елементів (shuffle), які ви знаєте способи це зробити?
5. У чому різниця між масивом і зв’язним списком, які в них є основні операції і яка їх складність?
6. Як працює додавання/видалення елементу в динамічному масиві і зв’язному списку? Яка складність цих операцій?
7. В чому різниця між однозв’язним і двозв’язним списком і як вони влаштовані?

1.3 Сортування

8. Що таке сортування, які ви знаєте сортування, які в них складності?

9. Як працюють елементарні сортування,чим вони відрізняються, які в них складності (сортування бульбашкою, включенням, вибором)?
10. Що таке сортування злиттям (merge sort), як воно працює, які в нього складності в найкращому/середньому/найгіршому випадках (по пам'яті і по часу)
11. Що таке швидке сортування (quick sort), як воно працює, які в нього складності в найкращому/середньому/найгіршому випадках (по пам'яті і по часу)
12. Якою є нижня межа складності для алгоритмів сортування, що засновуються на порівняннях (lower bound of comparison based sorting)?
13. Які ви знаєте сортування, які не засновані на порівняннях? Яка складність (по пам'яті і по часу) цих алгоритмів?

1.4 Купа / хешування / структури даних, що засновані на хешуванні

14. Що таке купа (heap)? Які є варіації купи? Які є способи реалізувати купу? Яка складність (по пам'яті і по часу) операцій на купою?
15. Що таке пірамідальне сортування (heap sort), як воно працює, які в нього складності в найкращому/середньому/найгіршому випадках (по пам'яті і по часу)?
16. Що таке хешування (hashing)? Як захешувати рядок довільної довжини? Що таке символьна таблиця (symbol table) та для чого вона призначена? Які є способи реалізувати символьну таблицю?
17. Що таке хеш-таблиця (hash table)? Які є підходи до реалізації хеш-таблиці? Чи потрібно (і яким чином) вирішувати хеш-колізії? Яка складність операцій з хеш-таблицею? Як реалізувати хеш-множину (hash set)?

2. Задачі на реалізацію (на Python)

1. **Binary search** (бінарний пошук)
2. **Stack [push, pop, clear] (list-based)** (стек на основі однозв'язного списку) - *в цьому завданні не можна використовувати стандарний список з Python)*
3. **Queue [push, pop, clear] (list-based)** (черга на основі однозв'язного списку) - *в цьому завданні не можна використовувати стандарний список з Python)*
4. **Doubly Linked List [insert(index, element), get(index), find(element), clear()]** (Двозв'язний список) - *в цьому завданні не можна використовувати стандарний список з Python*
5. **Elementary sort (bubble/insertion/selection)** (Елементарні сортування: бульбашкою, включенням, вибором)
6. **Merge sort** (сортування злиттям)
7. **Quick sort** (швидке сортування)
8. **Heap / Priority Queue (based on heap) [push, pop, clear]** (купа / черга з пріоритетом)
9. **Hash table (put, get, clear)** (хеш-таблиця) - *в цьому завданні не можна використовувати стандарний словник з Python.* Вирішення колізій можна реалізувати одним з двох варіантів:

a. **Open addressing** (відкрита адресація)
b. **Separate chaining** (метод ланцюжків)

3. Доведення складності

1. Бінарний пошук
2. Сортування злиттям
3. Швидке сортування (найкращий випадок і найгірший випадок)