1. З яких етапів складається процес створення комп’ютерної програми для вирішення довільної практичної задачі?

1) **Постановка задач**і – чітко вказати, що дано і що треба знайти.

2) **Побудова моделі** – які структури даних та які математичні залежності використані.

3) **Розробка алгоритму** – опис алгоритму у вигляді блок-схеми.

4) **Правильність алгоритму** – довести покроково правильність розробленого алгоритму.

5) **Аналіз алгоритму та його складності** – оцінити використовуючи О-символіку час виконання алгоритму в найгіршому або/і в середньому.

6) **Реалізація алгоритму** – навести текст програми.

7) **Перевірка програми** – описати тестові дані для перевірки програми на правильність,ефективність реалізації та обчислювальну складність (для цього використайте блок-схему алгоритму та профілі виконання програми).

2. Що саме має з’ясувати розробник програми на етапі постановки задачі?

На етапі постановки задачі розробник програми повинен з'ясувати кілька ключових аспектів:

1. **Мета програми**: Розробник повинен ретельно з'ясувати, що саме має вирішувати програма. Який конкретний проблема чи завдання вона має вирішувати?
2. **Вимоги користувачів**: Важливо зрозуміти потреби та очікування користувачів програми. Які функції чи можливості вони очікують від програми? Якими мають бути її характеристики?
3. **Обмеження та умови**: Розробник повинен врахувати обмеження, які можуть впливати на розробку програми, такі як обмеження з ресурсами (наприклад, часом, пам'яттю, обчислювальною потужністю), або правові обмеження.
4. **Можливість реалізації**: Розробник повинен оцінити, чи можливо взагалі реалізувати задачу технічно та в межах доступних ресурсів.
5. **Стратегія вирішення**: На цьому етапі також важливо розглянути можливі стратегії вирішення задачі. Які підходи або технології можуть бути використані для досягнення мети програми?

3. Що робить розробник програми на етапі побудови моделі? Які фактори впливають на вибір структури моделі?

На етапі побудови моделі розробник програми зазвичай виконує наступні кроки:

1. **Вибір типу моделі**: Розробник визначає, який тип моделі найкраще підходить для вирішення конкретної задачі. Це може бути статична або динамічна модель, графічна модель, математична модель тощо.
2. **Збір та обробка даних**: Розробник збирає необхідні дані для побудови моделі. Це може включати як числові дані, так і текстову інформацію. Після збору даних вони можуть піддаватися обробці, включаючи очищення від шуму, видалення відсутніх значень та інші операції підготовки.
3. **Вибір ознак для моделювання**: Розробник обирає, які саме ознаки чи параметри будуть використовуватися для побудови моделі. Це можуть бути числові значення, категоріальні дані або інші характеристики, які вважаються важливими для передбачення або аналізу.
4. **Вибір алгоритму моделювання**: Розробник обирає підходящий алгоритм для побудови моделі, враховуючи характеристики даних та мету аналізу. Це може бути алгоритм машинного навчання, статистичний метод або інший підхід, залежно від конкретної задачі.
5. **Побудова та оцінка моделі**: Розробник побудовує модель на основі вибраного алгоритму та навчає її на доступних даних. Після цього модель оцінюється за допомогою тестового набору даних для перевірки її ефективності та точності передбачень.

Щодо факторів, що впливають на вибір структури моделі, основні з них включають:

1. **Характеристики даних**: Структура моделі повинна відповідати характеристикам та особливостям даних, які використовуються для моделювання.
2. **Мета задачі**: Структура моделі повинна відповідати меті конкретної задачі. Наприклад, для класифікації можуть використовуватися моделі з різними архітектурами, такими як логістична регресія, дерева рішень чи нейронні мережі.
3. **Обсяг даних та обчислювальні ресурси**: Вибір структури моделі також може залежати від обсягу доступних даних та обчислювальних ресурсів, які можна витратити на її навчання та використання.
4. **Рівень складності задачі**: Складність задачі також може впливати на вибір структури моделі. Для складних задач можуть використовуватися більш складні моделі з більшою кількістю параметрів.
5. **Вимоги до інтерпретованості**: Якщо важливо мати можливість інтерпретувати результати моделі, то вибір структури моделі може спрямовуватися на більш прості моделі, які легше інтерпретувати.

4. Якими міркуваннями має керуватися розробник програми на етапі розробки алгоритму? Чи потрібно перевіряти або доводити правильність алгоритму, якщо так, тоз якою метою?

На етапі розробки алгоритму розробник програми повинен керуватися різними міркуваннями:

1. **Ефективність**: Розробник повинен забезпечити, що алгоритм ефективно вирішує поставлену задачу. Це означає, що він працює достатньо швидко та з високою точністю навіть при обробці великих обсягів даних.
2. **Читабельність**: Алгоритм повинен бути легким для розуміння та редагування, щоб інші розробники змогли зрозуміти його логіку та внести необхідні зміни у майбутньому.
3. **Масштабованість**: Важливо, щоб алгоритм можна було легко модифікувати або розширювати для роботи з більшими обсягами даних чи для вирішення більш складних завдань.
4. **Ресурсоємність**: Розробник повинен враховувати обмеження щодо доступних ресурсів, таких як пам'ять та обчислювальна потужність, і забезпечити, щоб алгоритм не був надто ресурсоємним.
5. **Коректність**: Алгоритм повинен правильно вирішувати задачу для всіх можливих вхідних даних та ситуацій.

Щодо перевірки або доведення правильності алгоритму, це може бути важливим з певних причин:

1. **Впевненість у результаті**: Перевірка алгоритму дозволяє розробнику переконатися, що він працює правильно для всіх вхідних даних та сценаріїв.
2. **Виявлення помилок**: Перевірка дозволяє виявити та виправити помилки чи недоліки в алгоритмі перед його використанням у реальних ситуаціях.
3. **Довіра користувачів**: Якщо користувачі впевнені у правильності та надійності алгоритму, вони будуть більш схильні використовувати програму та довіряти їй.

5. Навіщо виконується аналіз алгоритму та його складності?

Аналіз алгоритму та його складності виконується з кількох причин:

1. **Оцінка ефективності**: Аналіз допомагає зрозуміти, наскільки швидко або повільно працює алгоритм у різних сценаріях та з різними обсягами вхідних даних. Це дозволяє обрати оптимальний алгоритм для вирішення конкретної задачі.
2. **Оцінка витрат ресурсів**: Аналіз складності допомагає визначити, скільки пам'яті та обчислювальної потужності потрібно для виконання алгоритму. Це дозволяє ефективно використовувати обмежені ресурси, особливо в умовах обмежених обчислювальних ресурсів або на мобільних пристроях.
3. **Планування оптимізацій**: Знання складності допомагає ідентифікувати частини алгоритму, які можна оптимізувати для покращення його продуктивності. Це може включати в себе виправлення найбільш витратних або неефективних частин алгоритму.
4. **Порівняння алгоритмів**: Аналіз складності дозволяє порівняти різні алгоритми та визначити, який з них найкраще підходить для конкретної задачі. Наприклад, можна порівняти швидкість роботи алгоритмів з однаковою функціональністю та вибрати найефективніший.
5. **Прогнозування витрат на масштабування**: Аналіз складності допомагає прогнозувати, як зміниться продуктивність алгоритму при збільшенні обсягу даних або розміру задачі. Це важливо для планування масштабування системи та ресурсів, необхідних для її підтримки.

6. Існує три аспекти перевірки програми: на правильність, на ефективність реалізації, на обчислювальну складність. Розкрийте суть кожної з перевірок.

1. **Перевірка на правильність**:
   * **Суть**: Ця перевірка спрямована на визначення того, чи працює програма коректно відповідно до вимог і очікувань.
   * **Процес**: Включає тестування програми з використанням різних тестових випадків, щоб переконатися, що програма поводиться очікуваним чином у всіх можливих сценаріях.
   * **Методи**: Тестування може включати ручне тестування, автоматизовані тестові сценарії, тестування взаємодії користувача, тестування єдиниць коду, інтеграційне тестування та інші методи.
2. **Перевірка на ефективність реалізації**:
   * **Суть**: Ця перевірка спрямована на оцінку продуктивності програми та швидкості її виконання.
   * **Процес**: Включає вимірювання часу виконання програми та аналіз її продуктивності у різних умовах та на різних обсягах даних.
   * **Методи**: Вимірювання часу виконання, аналіз алгоритмів, профілювання коду для виявлення бутланеків та оптимізації.
3. **Перевірка на обчислювальну складність**:
   * **Суть**: Ця перевірка спрямована на визначення того, як швидко зростає час виконання програми при збільшенні обсягу вхідних даних.
   * **Процес**: Включає аналіз алгоритмів та оцінку їх складності за допомогою математичних методів.
   * **Методи**: Використання теорії алгоритмів для аналізу часової та просторової складності алгоритмів, вимірювання швидкості зростання вимог до ресурсів при збільшенні обсягу даних.

7. Навіщо виконується вимірювання часу виконання програми? Які чинники на нього впливають?

1. **Оцінка продуктивності**: Вимірювання часу виконання дозволяє оцінити, наскільки ефективно програма виконується у реальному середовищі. Це важливо для визначення продуктивності програми та виявлення можливих проблем з її ефективністю.
2. **Порівняння алгоритмів**: Вимірювання часу дозволяє порівняти різні алгоритми та реалізації, щоб вибрати найбільш ефективний підхід для вирішення конкретної задачі.
3. **Виявлення бутланеків**: Вимірювання часу виконання допомагає ідентифікувати частини програми, які витрачають найбільше часу, що дозволяє зосередитися на їх оптимізації.
4. **Оцінка змін**: Вимірювання часу дозволяє оцінити вплив змін у програмі на її продуктивність, що корисно при внесенні покращень чи оптимізацій.

Чинники, що впливають на час виконання програми, включають:

1. **Складність алгоритму**: Ефективність програми значно залежить від складності алгоритмів, які вона використовує. Деякі алгоритми можуть вимагати більше часу на обробку великих обсягів даних або складних операцій.
2. **Оптимізації коду**: Якість та ефективність написаного коду також можуть впливати на час виконання програми. Швидка та оптимізована реалізація алгоритмів може значно скоротити час виконання.
3. **Обсяг вхідних даних**: Час виконання програми зазвичай зростає зі збільшенням обсягу вхідних даних. Великі обсяги даних можуть призвести до збільшення часу виконання через більшу кількість операцій, що необхідно виконати.
4. **Обчислювальні ресурси**: Наявність обчислювальних ресурсів, таких як процесор, пам'ять та інші, також може впливати на час виконання програми. Недостатність ресурсів може призвести до збільшення часу виконання через накладні витрати на обробку даних.