1. Свойства `Count` и `Capacity`.
В чем заключается принципиальное различие между свойствами `Count` и `Capacity`? Проиллюстрируйте на примере: список с начальной `Capacity=4`, в который последовательно добавили 5 элементов, а затем удалили 3. Опишите состояние `Count` и `Capacity` на каждом этапе. Почему `Count` никогда не может быть больше `Capacity` (что это за *инвариант* такой)?

2. Амортизированная сложность операции `Add`.
Объясните понятие "амортизированная временная сложность" на примере операции `Add`. Почему сложность этой операции определяется как амортизированная O(1)? Опишите сценарий, при котором фактическая сложность единичного вызова `Add` составляет O(n).

3. Вычислительная стоимость операций вставки и удаления.
Рассмотрим список, содержащий N элементов. Опишите последовательность действий, выполняемых при удалении элемента из середины списка (например, по индексу N/2).

4. Стратегии увеличения ёмкости.
В методичке мы удваиваем `Capacity` при росте (`* 2`). А что если бы мы увеличивали ее на константу, например, `+ 10` элементов? **Какой подход предпочтительнее и почему?** Сравните оба подхода (умножение на `K` против сложения с `N`) с точки зрения количества операций копирования и расхода памяти, если нам надо добавить 1000 элементов.

5. Управление памятью и роль Сборщика Мусора (GC).
В методе `Resize` (или `Grow`) есть строчка `_data = newData;`. Что в этот момент происходит со **старым** массивом, на который `_data` указывал раньше?

6. Использование `default(T)` при удалении элементов.
При реализации метода `RemoveAt` рекомендуется обнулять значение в ячейке, ставшей вакантной ("хвостик") после сдвига элементов (`_data[_count] = default(T);`). Разве мы не можем просто уменьшить `_count` и оставить хвост как есть? Как вы

считаете: зачем нам это делать и почему это может быть важно при работе со ссылочными типами данных.

7. Роль обобщений (Generics).
Предположим, в языке отсутствуют обобщения (generic). С какими проблемами вы столкнетесь при необходимости реализовать *типобезопасные* списки для различных типов данных? Каким образом механизм обобщений (`<T>`) решает проблемы дублирования кода и обеспечения безопасности типов?

8. Предварительное выделение ёмкости.
Каково предназначение конструктора, принимающего начальную ёмкость в качестве параметра (`MyArrayList(int capacity)`)? Приведите **конкретный, жизненный** пример, в котором использование этого конструктора будет предпочтительнее по сравнению с конструктором по умолчанию.

9. Поведение коллекции в граничных состояниях.
Ваш список пуст (`Count = 0`). Какие из этих операций должны выкинуть исключение, а какие — отработать "штатно"?

- Доступ к элементу по индексу `list[0]`
- `list.RemoveAt(0)`
- `list.Insert(0, item)`
- `list.Remove(item)`
- `list.Clear()`
  Для каждого случая укажите ожидаемое поведение (штатное выполнение или генерация исключения) и обоснуйте его.