1. а) Да, верно.

Докажем это по определению. $n = O(n \log n)$, следовательно, $\exists C > 0, N \in \mathbb{N}, \forall n \geqslant N$
$n \leqslant Cn \log n$
$1 \leqslant C \log n$
$\log n \geqslant 1/C$
$n \geqslant 2^{1/C}$

Таким образом, $N = 2^{1/C}$. Данные выкладки проведены для двоичного логарифма, но ничего не поменяется с логарифмом другого основания, кроме константы.

б) Нет, неверно.

По определению:
$n^{1+\epsilon} = O(n \log n)$
$\exists C > 0, N \in \mathbb{N}, \forall n \geqslant N$
$n^{1+\epsilon} \leqslant Cn \log n$
$\frac{n^{1+\epsilon}}{n \log n} \leqslant C$

Следовательно, $\lim\limits_{n\to\infty} \frac{n^{1+\epsilon}}{n \log n} < \infty$

Однако $\lim\limits_{n\to\infty} \frac{n^{1+\epsilon}}{n \log} = \lim\limits_{n\to\infty} \frac{n^{\epsilon}}{\log n} = \lim\limits_{n\to\infty} \frac{\epsilon n^{\epsilon-1}}{1/n} = \infty$

2. а) Да, возможно. Пусть $f(n) = n \log n, g(n) = 1$. Тогда $h(n) = n \log n$.

Для начала покажем, что возможно $f(n) = n \log n$.

По определению: $n \log n = O(n^2)$
$\exists C > 0, N \in \mathbb{N}, \forall n \geqslant N$
$n \log n \leqslant Cn^2$
$\frac{\log n}{n} \leqslant C$

Рассмотрим предел: $\lim\limits_{n\to\infty} \frac{\log n}{n} = 0$, следовательно, условия выполняются.

По определению Θ-нотации: $C_1 n \log n \leqslant n \log n \leqslant C_2 n \log n$, что возможно при $C_1 = C_2 = 1$

б) Нет, невозможно.

По определению Θ-нотации: $C_1 n^3 \leqslant h(n) \leqslant C_2 n^3$

$h(n) \leqslant C_2 n^3$ возможно при любом выборе $f(n)$ и $g(n)$, так как степень роста $h(n)$ не выше второй, что будет показано ниже.

Однако второе неравенство: $C_1 n^3 \leqslant h(n)$ означает, что степень роста$h(n)$ больше третьей, что, как было сказано выше, невозможно.

2. Верхняя оценка: $h(n) = O(n^2)$. Так как числитель $f(n) \leqslant Cn^2$, а знаменатель $g(n) \geqslant C$

Нижней лучшей оценки нет (0 не входит в класс рассматриваемых функций), ведь числитель может быть сколь угодно мал.

3. Первый внутренний цикл по j выполняется $n/2$ раз, второй – $\log_2 n$ раз, с точностью до единицы, если принимать во внимание четность n. *bound* пробегает значения: $1, 2, 4, ..., n$ опять с точностью до степени 2. Соответсвенно цикл по i будет выполняться $1, 2, 4, 8, ..., n$ раз. Таким образом, всего получаем: $(1 + 2 + 4 + 8 + ... + n)(n/2 + log_2 n)$.

$(1 + 2 + 4 + 8 + ... + n - 1) = \frac{(2^{\log_2 n + 1} - 1)}{2-1} = (2n - 1)$

То есть, без учета констант получим $g(n) = n(n+\log n)$. Следовательно, $g(n) = \Theta(n^2)$

5.

Псевдокод:

Обозначим массивы: a, b, c.

$min\_element = min\{a[0], b[0], c[0]\}$

$count = 1$ – *переменная - счетчик для числа элементов в объединении массивов*

удаляем $min\_element$ *из всех массивов, в которых есть* $min\_element$

`while` (не конец первого массива) `or` (не конец второго массива) `or` (не конец третьего массива) {

  $min\_element\_current = min\{a[0], b[0], c[0]\}$

  удаляем $min\_element\_current$ *из всех массивов, в которых есть* $min\_element\_current$

  `if` $(min\_element \neq min\_element\_current)$ {

    $count = count + 1$

  }

  $min\_element = min\_element\_current$

}

`переменная` $count$ `и будет содержать итоговое число элементов в объединении массивов`

Сложность алгоритма:

Так как на каждом шагу просматриваются три элемента, выбирается из них минимальный, и он удаляется, то при каждой итерации цикла в худшем случае удаляется элемент хотя бы из одного массива, то есть сложность будет $O(n_1 + n_2 + n_3)$ – линейная.

Корректность алгоритма:

1. Данный алгоритм детерминированный

2. Данный алгоритм проходит по всем элементам трех массивов, так что пропущенных элементов быть не может.

3. На каждом шагу алгоритма удаляется текущий минимальный элемент из всех массивов и сравнивается с предыдущим минимальным, следовательно, двух или более одинаковых элементов нет.

6.

Распишем сумму: $\sum\limits_{i \neq j} a_i b_j = (a_1b_1 + a_1b_2 + ...a_1b_n) + ... + (a_nb_1 + a_nb_2 + ...a_nb_n) - (a_1b_1 + a_2b_2 + ...a_nb_n) = (a_1 + a_2 + ... + a_n)(b_1 + b_2 + ... + b_n) - (a_1b_1 + a_2b_2 + ...a_nb_n)$

Обозначим: $g_1(a_1, a_2, ..., a_n) = a_1 + a_2 + ... + a_n$, $g_2(b_1, b_2, ..., b_n) = b_1 + b_2 + ... + b_n$, $g_3(a_1, a_2, ..., a_n, b_1, b_2, ..., b_n) = (a_1b_1 + a_2b_2 + ...a_nb_n)$

$g_1(a_1, a_2, ..., a_n), g_2(b_1, b_2, ..., b_n), g_3(a_1, a_2, ..., a_n, b_1, b_2, ..., b_n)$ – индуктивные функции.

$f(a_1, a_2, ..., a_n, b_1, b_2, ..., b_n) = \sum\limits_{i \neq j} a_i b_j$ – не индуктивная, построим ее индуктивное расширение.

Будем хранить в памяти результаты вычисления функций g_1, g_2, g_3. Обозначим их значения через s_1, s_2, s_3 соответственно. Таким образом, $f(a_1, a_2, ..., a_n, b_1, b_2, ..., b_n) = t(g_1, g_2, g_3)$, где $t(g_1, g_2, g_3) = g_1 * g_2 - g_3$. Следовательно, после подсчета индуктивных функций в цикле, мы будем иметь конечный результат: $res = s_1 * s_2 - s_3$

Сложность алгоритма:

Данный алгоритм является онлайн-алгоритмом, выполняется за один проход, его сложность $O(n)$ – линейная.

Корректность алгоритма:

1. Данный алгоритм детерминированный.
2. Значения высчитываются по выведенным математическим формулам.
3. Данный алгоритм получает на вход все элементы, так что пропущенных быть не может.

7.

Заведем массив len, который будет хранить длины возрастающих подпоследовательностей. Значения элементов этого массива считаются с помощью индуктивной функции: $len[i] = \max\limits_{j=0,\ldots,i-1, a[j]<a[i]}(len[i]+1)$. Но это для ситуации, когда длина подпоследовательности больше 1, иначе $len[i] = 1$. Таким образом, итоговая формула для расчета: $len[i] = \max(\max\limits_{j=0,\ldots,i-1, a[j]<a[i]}(len[i]+1), 1)$. Следовательно, нам нужно хранить в памяти длины всех возрастающих подпоследовательностей, чтобы потом найти максимум массива len, а индекс масимального элемента len и будет концом наибольшей возрастающей подпоследовательности, так что мы сможем ее восстановить.

Псевдокод:

for $(i = 0; i < n; i++)\{$
  $len[i] = 1$
  for $(j = 0; j < n; j++)\{$
    if $(a[j] < a[i])$
      $len[i] = \max(len[i]+1, len[i])$
    $\}$
  $\}$

Заведем переменные max_ind – индекс последнего элемента наибольшей возрастающей подпоследовательности и max_len – длина наибольшей возрастающей подпоследовательности.

for $(i = 0; i < n; i++)\{$
  if $(len[j] < max)$ $\{$
    $max\_len = len[i]$
    $max\_ind = i$
  $\}$
$\}$

Таким образом наидлинейшая возрастающая последовательность: от $a[max\_ind - max\_len + 1]$ до $a[max\_ind]$.

Сложность:

Нахождение длины наибольшей возрастающей подпоследовательности осущетсвляется за $O(n^2)$ за счет двух вложенных циклов. Восстановление подпоследовательности происходит за один проход, так что $O(n)$. Таким образом, итоговая сложность алгоритма – квадратичная $O(n^2)$.

Корректность:

1. Данный алгоритм детерминированный

2. Проходит все элементы массива и осуществляет подсчет результата по выведенным индуктивным формулам.

8.

В стек будем помещать вновь введенный элемент. При этом в отдельную переменную запоминая максимальный элемент.

$max\_el = x_1$
`for` $(i = 0; i < n; i++)\{$
  вводим очередной элемент x_i
  `if` $(x_i > max\_el)$ $\{$
    $max\_el = x_i$
  $\}$
$\}$

Затем создаем массив фиксированного размера $freq[max\_el]$ и инициализируем его нулями. Считываем последовательно элементы из стека и соотвествующий элемент $freq[x_i]$ инкрементируем на единицу. Проверяем, если значение этого элемента больше $n/2$, то это и есть искомый элемент (такой элемент по условию единственный), иначе считываем дальше.

Сложность:

Запись элементов в стек занимает $O(n)$, чтение также $O(n)$, создание массива $freq[max\_el]$ также $O(n)$, так что итоговая сложность – линейная $O(n)$.

Корректность:

1. Данный алгоритм детерминированный.

2. Получает на вход все элементы, пропущенных значений быть не может.

3. Считывание из стека элементов происходит по выполнения условия, что элемент встречается более $n/2$ раз, так как по условию он единственный такой, то завершение алгоритма на этом шаге является корректным.