Problem 1.

As noted **in A3 description** "giving" an algorithm means: describe the algorithm briefly in words, give high-level pseudocode, justify correctness, and analyze run-time.

(a) Description: The problem reduces to finding the maximal i index s.t. $c_i \geqslant i$.
Algorithm: Divide the array into two halves, check if the middle point $c_{mid} \geqslant mid$ with $mid = \frac{beg+end}{2}$. "beg" denotes the index of the first element in the current array within $C[1, 2, \ldots, n]$ and "end" denotes the index of the last element. If $c_{mid} \geqslant mid$, apply the same divide-and-conquer on the second half. Otherwise, apply the same divide-and-conquer on the first half. The base cases in the end are trivial to solve.

(b) Pseudocode:
Input: the sorted array C. C starts with index 1.
Output: the h-index.

$n = C.\text{size}();$
if $(n == 0)$ return 0;
$beg = 1, end = n;$
if $(n == 1)$ return $(c_1 == 0)\,?\,0 : 1;$
define $H(C[beg \ \ldots end])$:
  if $(beg == end)$ return beg;
  if $((end - beg) == 1)$ return $(c_{end} \geqslant end)\,?\,end : beg;$
  $mid = \frac{beg+end}{2};$
  return $(C[mid] \geqslant mid)\ ?\ H(C[mid \ \ldots end]) : H(C[beg \ \ldots mid]);$
return $H(C[1\,2 \ \ldots \ n]);$

(c) Correctness:
First we prove the problem is equal to finding the maximal index s.t. $c_i \geqslant i$. In this case, $c_1 > c_2 > \cdots > c_i \geqslant i$, according to the definition h-index $\geqslant i$. On the other hand, since i is maximal, $c_{i+1} < i+1$, so $c_n < c_{n-1} < \cdots < c_{i+1} < i+1$ so h-index $< i+1$.

Thus h-index $= i$. If $n = 0$, it is obvious that h-index $= 0$.
If $n = 1$, it depends on whether $c_1 = 0$. If so, the h-index is 0. Otherwise, $c_1 \geqslant 1$ and by definition h-index is 1.
$H(C[beg \ \ldots end])$ uses a subarray from $C[1, 2, \ldots, n]$ and outputs the maximal i among beg, beg + 1, …, end, s.t., $c_i \geqslant i$.

Fill in running time analysis …

Problem 3.

Sort items in decreasing order of c_i. Pick them up in this order. We claim that the total cost will be minimized.

Suppose (by re-indexing) that $c_1 > c_1 > \cdots c_n$. Then we pick up item i on day i which costs c_i^{i-1}. Item c_1 is picked up on the first trip, so it costs $1 = c_1^0$ The total cost is $c_1^0 + c_2^1 + c_{i_2}^2 + \cdots + c_n^{n-1}$.

Consider any different solution S. We will show that its cost can be decreased. Since solution S is different there must be two items i and j where $c_i < c_j$ but we pick up item i before item j. Suppose we pick up item i after d days and item j after $d+k$ days.

The cost of these two items in solution S is $c_i^d + c_j^{d+k}$.

Consider a new solution S' where we swap these two items. The cost of these two items in solution S' is $c_i^{d+k} + c_j^d$. Costs for all other items remain the same.

We want to prove that S' costs less, i.e.

$$c_i^{d+k} + c_j^d < c_i^d + c_j^{d+k}$$

Rearranging, we want to prove:

$$c_i^d(c_i^k - 1) < c_j^d(c_j^k - 1)$$

This is clear because $c_i < c_j$.

Since we can decrease the cost of any solution different from the greedy one, therefore the greedy solution minimizes the cost.

The running time of this algorithm is $O(n \log n)$ to sort.