1. In each of the following problems, a set of operations permissible on the underlying data structure are given. For each of the operations, compute their best and worst case complexities. Also, compute the actual cost (in terms of credits) per operation and manage credits such that each operation has $\mathcal{O}(1)$ amortized cost.
   (a) Convex Hull problem.
      1. $updateHull(H, p)$ - Add point p to the set of points under consideration and update the current hull H, accordingly.

   (b) n-Bit Counter problem.
      1. $increment(\chi)$ - Increment the counter χ by 1.
      2. $reset(\chi)$ - Set χ to 0.

   Hint: Maintain a pointer to the most significant bit of the counter.

   (c) Stack operations.
      1. $Push(S, x)$ - Push element x into the stack S.
      2. $Pop(S)$ - Pop the topmost element from S.
      3. $MultiPop(S, k)$ - Pop the topmost k elements from S; pop all elements if $k \geq |S|$.

   Hint: Try to store the credits in such a way that Pop and $MultiPop$ are "free of cost".

   (d) Dynamic array problem.
      1. $addElement(A, x)$ - Add the element x into the first free space in the (possibly unsorted) array A. If A has no free space, create a new array of size $|A|$, copy all the elements from A into the new array, free the space allocated to A and set A to point to the new array.
      2. $deleteElementAtIndex(A, i)$ - Delete the element at $A[i]$.

   Can you further adapt $deleteElementAtIndex(A, i)$ so that it reduces the size of the array if "too many" indices are empty, while still have only an $\mathcal{O}(1)$ amortized cost?

2. A *credit invariant* is a statement about the credit availability in the data structure. The amortized costs (that is, number of credits required by the operation) of an operation must be assigned in such a manner that even after its execution, the credit invariant is maintained.

   For each part in the preceding question, specify the credit invariant for the data structure and argue that no operation violates it.

3. Suppose that an algorithm has access to l different operations, such that the i^{th} operation has an amortized cost of c_i. If the algorithm has executed $k = \sum_{1 \leq i \leq l} a_i$ operations, where a_i is the number of times the i^{th} operation was executed, then prove the following.

   To the point at which k operations have been executed, the algorithm will have run for $\mathcal{O}(\sum_i a_i c_i)$ steps in the worst case.

4. Consider a specialized variant of the binary heap in which *deleteMin* and *decreaseKey* operations, respectively, have amortized costs of $\Theta(\log n)$ and $\Theta(1)$. Suppose that, over the course of the algorithm, these operations are respectively executed n, m times.

   We can then conclude that the algorithm's _____ *complexity is* $\Theta(m + n \log n)$.

   Complete the statement using one (or more) of the following options.

   1. amortized 2. worst case 3. average case