- **Algorithm**

  We need a two dimensional array where each cell stores the number of cards needed, $f(i, s)$, when we choose s videos out of the first i videos down the list. We have $s \leq i$ because it is impossible to choose more than what we have.

  We prefer to store $f(i, s)$ as a rational number rather than an integer because we also want to know how many bytes the last card currently have. For example, if the first five cards are not able to store more subsequent videos while the sixth card contains a video of $\frac{m}{2}$ bytes, then $f(i, s) = 5.5$.

i | s	1	2	3	4	...	n
1	$\frac{l_1}{m}$	NA	NA	NA	...	NA
2			NA	NA	...	NA
3				NA	...	NA
4					...	NA
...	...	...	...	...	...	NA
n						

  Our base case is when we select one video out of a list that contains exactly one video, which requires one card filling with $\frac{l_1}{m}$ of its memory.

  For each video, we either to choose it or not. We compute the number of cards for both cases and then choose the one with a smaller rational number.

  - **Case 1: Not to choose the ith video.** The video collection is not changed at all.
    $f(i, s)$ will remain the same as selecting $(s - 1)$ from i videos.

    $$f(i, s) = f(i - 1, s).$$

  - **Case 2: To choose the ith video.**
    The ith video is added to the selection. The current card, which is either empty or half-filled, may or may not have enough room for the ith video.
    If the ith video can fit into the current card, we put the video in it and only update the decimal part of the number of cards.

    $$f(i, s) = f(i - 1, s - 1) + \frac{l_i}{m}.$$

    If the ith video is too big to fit into the current card, we add an additional card to store it. We round up $f(i - 1, s - 1)$ and then update the decimal part of the number of cards.

    $$f(i, s) = round(f(i - 1, s - 1)) + \frac{l_i}{m}$$

    where $round(x)$ is the smallest integer larger than the non-negative rational number x.

  Starting at $s = 1$ from top to bottom and from left to right, we calculate for $f(i, s)$.

  If we pick $f(i, s) \leq k$ with the largest s to be the solution, then s must be the maximum number of videos than can be stored on k cards, which can be proved by contradiction.

  Suppose for a given k, there exists a n' that is greater than n found by our algorithm. That being said, k cards can store $(n' - n)$ more videos. These cards are not possible to be hold on the kth card or else $f(i, s)$ will not be the one with largest s. Then there exists a $k' < k$ that can store n videos, which contradicts our assumption that k cards is the minimum to store n videos.

- **Correctness**

  The above DP algorithm is optimal as it has the following three traits.

  - **Complete Choice**: We have included all combinations of videos that might be an optimal solution. For every video, we either choose or not to choose it and then compare their outcomes.
  - **Inductive Structure**: For any valid pair of i and s there are two cases, each containing a subproblem. We use different equations, as listed above, to calculate the number of cards in each subproblem. Having smaller i and smaller (or same) s, each subproblem is solved by exactly the same way. There are no external constraints on how to select s from i videos.
  - **Optimal Substructure**:

    There are multiple possible cases of how subproblem and DP choice are combined.

    * **Case 1: Not to choose the ith video.** The number of videos in P remains the same as that in P'. It is obvious that if S' is an optimal solution, so does S.
    * **Case 2: To choose the ith video.** Suppose we have an optimal solution S' to subproblem P' while the solution to P is not optimal. Then there exists an optimal solution $S*$ such that $f(S*) < f(S)$. Both $S*$ and S may choose or not choose a card but it is not possible that $S*$ opens a new card while S does not, otherwise $f(S*)$ will not be smaller than $f(S)$. Therefore there are three subcases.

      If both $S*$ and S do not open a new card, the sum in P is simply the minimum value in subproblem plus a constant fraction, the combined solution to P is still optimal according to standard contradiction argument.

      If S opens a new card while $S*$ does not, then

      $$round(f(S*')) + \frac{l_i}{m} < f(S') + \frac{l_i}{m}$$

      $$f(S*') \leq round(f(S*')) < f(S')$$

      $$f(S*') < f(S').$$

      If they each opens a new card, then

      $$round(f(S*')) + \frac{l_i}{m} < round(f(S')) + \frac{l_i}{m}$$

      $$f(S*') \leq round(f(S*')) < f(S') \leq round(f(S'))$$

      $$f(S*') < f(S').$$

      For each of the three subcases above, we obtain $f(S') < f(S')$, a solution optimal to P' contradicting our assumption that S' has been solved optimally.

    To sum up, in all cases we get an optimal solution to P if we have an optimal solution to P'.

- **Complexity**

  Computing each cell of the table requires making comparisons between two cases, which takes O(1) time. The table is two dimensional with O(n^2) cells.

  Therefore, the overall time complexity of the algorithm is $\mathbf{O}(n^2)$.