Assignment 4 Problem 2

a) Assume that we have a hash table of size 6 and that our keys are selected uniformly at random from the set $A = f\{1, 2, 3, \ldots, 600\}$. Consider the following two hash functions:

$$h_0(k) = k \mod 6$$
$$h_1(k) = 2k \mod 6$$

Is one hash function preferred over the other? Justify your answer.

b) Given an array A of size n storing integers and a number m, give an algorithm to determine whether array A contains a subarray whose elements sum to m. For example, the array $A = [5, 7, 9, 11, 13, 15]$ has a subarray, at consecutive indices 1 and 2, where elements sum to $m = 16$. Your algorithm must have an expected runtime of $O(n)$.

solution:

a) It seems that $h_0(k)$ is better than $h_1(k)$, because for "$h_1(k) = 2k \mod 6$", the index of even number (i.e. 1 3 5) will never be used. But for $h_0(k)$, each hash value (i.e. 0,1,2,3,4,5) is equally likely.

b) The algorithm states as follows: We construct two pointer to point the begin of the subarray (i), and the end of the subarray (j), and an variable to be the sum of all element in the current subarray (arr_sum). All these are initialed to be 0.
Then we start change j along the array, each time we increase j by 1, and see whether $arr\_sum + A[j]$ is greater than m, if yes, which means that the current subarray exceed what we want, then we delete the first term in the subarray by increase the i by 1 and changes the arr_sum (more details will state later), if it is exactly m, then we find the required subarray, otherwise, we continue increasing j.
When deleting the first few elements in the subarray, we may need to delete more than one element, so when we decide to remove the elements away from subarray, we firstly delete one, then we check the current sum with m, if it is exactly m, then we find the required subarray, if it is less than m, then we stop remove elements, and do the previous step (increase the end of the subarray j and check), if it is still greater than m, then we continue remove elements from subarray.
When j reach the end of the array A, and sum of current subarray is still less than m. Then we return that we did not find a subarray with sum m.
The pseudo-code:
Subarray_exist(A, m)
A: array of size n
m: the number that we want the subarray sum to

$i \leftarrow 0$ $j \leftarrow 0$ $arr\_sum \leftarrow 0$
while true // keep iterating
    if($j == n$) return not found!
    if($arr\_sum == m$) return found!
    if($arr\_sum < m$)
        $j \leftarrow j + 1$
        $arr\_sum \leftarrow arr\_sum + A[j]$
    else
        $arr\_sum \leftarrow arr\_sum - A[i]$
        $i \leftarrow i + 1$

This ensures the runtime in O(n), because each iteration we return or add 1 to i or j. And i will never get greater than j, the program will terminate when j is equal to n. Therefore, the maximum number of iterations is $\Theta(n)$. Therefore, the algorithm is in O(n).