Maoting Ren
mren

5-1

firstly we use an array A[n], to count the number of elements cast into each position. For an example, we have n = 6 elements, they are 1, 3, 4, 7, 7, 9. We choose value%6 as the hash function. And then array A will become [0, 3, 0, 2, 1, 0], as 1, 7, 7 will cast in position 1, so A[1] = 3 and so on.
Then we use another array B[n] to accumulate the number, that is array B will become [0, 3, 3, 5, 6, 6]. We still need the third array hash[m] (m>=n) as the hash table and cast element into it. Each time we cast one element into the hash table, it will do hash[B[val%6]-1] = value, and B[val%6]--.

And the operation is:
1. when we insert the first element 1 into the hash table, we know it should in position 1 as 1%6 = 1, and we check array B[1] = 3, it indicate that position 3-1 = 2 is available for this element, so we set hash[2] = 1, and update B[1] = 3-1 = 2, indicate the next element who will cast into position 1, will put into position 2-1 = 1.
2. when we insert the second element 3 into the hash table, we know it should in position 3 as 3%6 = 3, and we check B[3] = 5, it indicate 5-1 = 4 is available for this element, so we set hash[4] = 3, and update B[3] = 5 - 1 = 4.
3. when we insert 4 into the hash table, in to same way, the key will be 4, as 4%6 = 4, and B[4] = 6, so we set hash[5] = 4, and update B[4] = 6-1 = 5
4. in the same way, when we insert element 7, whose key is 7%6 = 1, B[1] = 2, so we set hash[1] = 7, and update B[1] = 1.
5. when we insert the element 7, whose key is 7%6 = 1, B[1] = 1, we set hash[0]=7, and update B[1]= 0
6. when we insert the last element 9, whose key is 9%6 = 3, B[3] = 4, so we set hash[3] = 9, and update B[3] = 4-1 = 3.
so at last, we insert all element into the hash table, and the hash table array will become [7, 7, 1, 9, 3, 4], and it will only cost Θ(n) time and space.

5-2

the length of the array is n, so firstly we choose k = n/2, low = 0, high = n-1. Sweep the array and construct a substring that length is k at each position, then cast it into a hash table,
1) if you can find any two substrings have same key(we can choose the substring as key), it indicate that the longest length will >= k, then make low = k+1 ,k = (k + high)/2, and, and find if exist such two substrings have the same key.
2) If no such two substrings that have the same key, it indicate that the longest length will < k, so we make high = k-1, k = (low + k)/2, and search again.
By this way we can find the longest length in O(n log n) time and O(n) space.

5-3

a. To find if there are two elements within some specified distance k, we can divide each element by 2k, and cast the result into a hash table. By this way, if two elements cast into the same position, the distances between them are at most 2k, so we can check the collision set and find if the distance is less than k.
But by this way, there are still some element pairs can not discover, that is they will cast into different position, so to solve this problem, if we can not find such element pair by above approach, we can add k to each element, and then each element divide by 2k and cast the result into a hash table, then check the collision set, it will cover all cases. So if we still can not find such element pair, there is no such

element pair exist.
This algorithm will cost only $\Theta(n)$ time and space.

b) same with the above algorithm, we apply this algorithm on X coordinate first, if we can find two elements whose distance is less than k, then we verify if the distance between the two points are less than k.
this approach will cost only $\Theta(n)$ time and space.

5-4
select two points randomly, and compute the distance between the two points k, and utilize the above algorithm to verify if there exist two points whose distance is less than k, if exist, then we set k equal that distance, and apply that algorithm again, until there are no such points pair that whose distance less than k, then we get our answer.