1. Given an array of unordered integers A and a target difference D, design an algorithm with $O(n)$ runtime complexity to find the first pair of integers within A such that their difference equals D. The order in which the integers appear in the pair should match their order in the array (i.e., if the pair is (a_i, a_j), then $i < j$). Your algorithm should utilize a hash table of size n the size of the array, for efficient computation. Note: The algorithm only needs to return the first pair that meets the criteria. (10P)

To solve this with O(n) runtime complexity and a hash table of n size.

```
function int[] findTheDifference (A, D) {        // A: array of integers; D: integers
    hashMap = new hashMap   // making a hashmap to store integers
    for (int i = 0; i < A.length-1; i++) {
        upperInteger = A[i] + D;
        lowerInteger = A[i] - D;   // calculating the potential integers
        if (hashMap.containsKey(upperInteger)){
            return new int[] {A[i], upperInteger};   // a pair where current integer is smaller than a previous integer, return with [current, previous]
        }
        else if (hashMap.containsKey(lowerInteger)){
            return new int[] {lowerInteger, A[i]}   // a pair where current integer is larger than a previous integer, return with [previous, current]
        }
        else {
            hashMap.put(A[i], i);   // if a pair is not found, add the integer to the hash map
        }
    }
    return "No such pair exists in this array!" (or null)   // no possible pair in the entire array, thus null.
}
```

2. Given input: 4371, 1323, 6173, 4199, 4344, 9679, 1989 and hash function h(x)=x (mod 10), show the result of inserting these keys into a hash table with:
a. Separate chaining. (3.5 P.)
b. Open addressing with linear probing. (3.5 P.)
c. Open addressing with quadratic probing. (4 P.)
d. Open addressing with double hashing and the secondary hash function h2(x) = 7 - (x mod 7). (4 P)

a) Separate chaining w/ hash function $h(x) = x \pmod{10}$

0	null
1	4371
2	null
3	1323 → 6173
4	4344
5	null
6	null
7	null
8	null
9	4199 → 9679 → 1989

b) Open addressing w/ linear probing

0	9679
1	4371
2	1989
3	1323
4	6173
5	4344
6	
7	
8	
9	4199

c) Open addressing w/ quadratic probing

0	9679
1	4371
2	
3	1323
4	6173
5	4344
6	
7	
8	1989
9	4199

$9679 \% 10 = 9+1 = 10$
$10 \% 10 = 0$ ✓

$1989 \% 10 = 9+1 = 10$
$10 \% 10 = 0$ ×
$9+2^2 = 9+4 = 13 \% 10 = 3$ ×
$9+3^2 = 9+9 = 18 \% 10 = 8$ ✓

d) Open addressing w/ double hashing + secondary function $h_2(x) = 7 - x \pmod 7$

Fails due to 1989.

0	
1	4371
2	
3	1323
4	6173
5	9679
6	
7	4344
8	
9	4199

$4371 \% 10 = 1$
$1323 \% 10 = 3$

$6173 \% 10 = 3$
try $3+1 \cdot (7 - 6173 \% 7)$
$= 3+(7-6)$
$= 3+1 = 4$

$4199 \% 10 = 9$

$4344 \% 10 = 4$
try $4+1 \cdot (7 - 4344 \% 7)$
$= 4+(7-4)$
$= 4+3 = 7$

$9679 \% 10 = 9$
try $9+1(7 - 9679 \% 7)$
$= 9+(7-5)$
$= 9+2 = 11 \% 10 = 1$

try $9+2(7 - 9679 \% 7)$
$9+2(7-5)$
$9+2(2)$
$9+4 = 13 \% 10 = 3$

try $9+3(2)$
$= 9+6 = 15 \% 10 = 5$

$1989 \% 10 = 9$
$9+1(7 - 1989 \% 7) = 9+(7-1)$*

3.Let an array arr = [9, 8, 8, 5, 7, 7, 4, 4, 4, 2]. Sort arr from the smallest to largest value using:
a. Selection sort. (3.5P)
b. Insertion sort. (3.5P)
c. Quicksort, by partitioning around the last element. (4P)
d. Mergesort. (4P)
To earn credit, you must show all steps for all algorithms by:
- Writing the arrays after each step for parts a and b
- Drawing the recursive call tree, similar to figure 16-3 on page 11 of slide Ch16, for parts c and d

*continued from 2d

9+(6)= 9+6=15 %10=5
9+2(6)=9+12=21 %10=1
9+3(6)=9+18=27 %10=7
9+4(6)=9+24=33 %10=3
9+5(6)=9+30=39 %10=9
9+6(6)=9+36=45 %10=5
9+7(6)=9+42=51 %10=1
9+8(6)=9+48=57 %10=7
9+9(6)=9+54=63 %10=3
9+10(6)=9+60=69 %10=9

a) Selection sort

Start by finding the smallest element + swapping it with the first element. Just keep doing the same for each element until the array is sorted.

Original: [9,8,8,5,7,7,4,4,4,2]

1. [2,9,8,8,5,7,7,4,4,4]
2. [2,4,9,8,8,5,7,7,4,4]
3. [2,4,4,9,8,8,5,7,7,4]
4. [2,4,4,4,9,8,8,5,7,7]
5. [2,4,4,4,5,9,8,8,7,7]
6. [2,4,4,4,5,7,9,8,8,7]
7. [2,4,4,4,5,7,7,9,8,8]
8. [2,4,4,4,5,7,7,8,9,8]
9. [2,4,4,5,7,7,8,8,9]
10. [2,4,4,5,7,7,8,8,9] ← array is sorted

b) Insertion sort

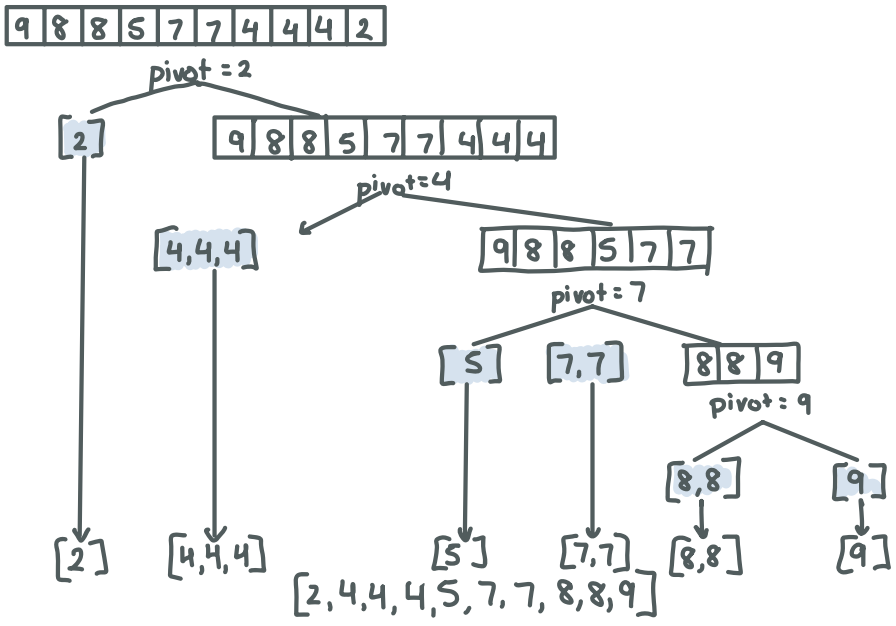
Start with the 2nd element in the array and put the element in the correct position according to the values to the left. Repeat until end.

Original: [9,8,8,5,7,7,4,4,4,2]

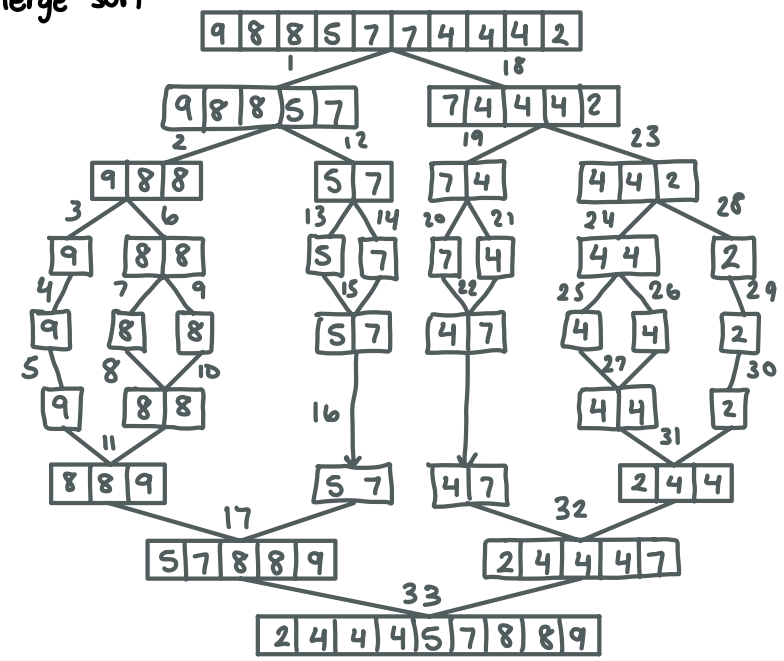
1. [8,9,8,5,7,7,4,4,4,2]
2. [8,8,9,5,7,7,4,4,4,2]
3. [5,8,8,9,7,7,4,4,4,2]
4. [5,7,8,8,9,7,4,4,4,2]
5. [5,7,7,8,8,9,4,4,4,2]
6. [4,5,7,7,8,8,9,4,4,2]
7. [4,4,5,7,7,8,8,9,4,2]
8. [4,4,4,5,7,7,8,8,9,2]
9. [2,4,4,4,5,7,7,8,8,9]

Array sorted!

c) Quick-sort, partion around last element



d) Merge-sort



copy to original