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.

✓ **Answer**

```
function findPair(A: number[], D: number) {
        let table = new Array(D)
        for (let i of A) {
                let o = Math.floor(i / D)
                let r = ((i % D) + D) % D
                if (!table[r]) table[r] = []
                if (table[r].contains(o-1)) return [D*(o-1) + r, i]
                if (table[r].contains(o+1)) return [D*(o+1) + r, i]
                table[r].push(o)
        }
        return null
}

console.log(findPair([1, 3, 5, 7, 9], 3) || "None")
console.log(findPair([1, 3, 5, 7, 9], 2) || "None")
console.log(findPair([10, 20, 30, 35], 5) || "None")
```

```
None
1,3
30,35
```

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 (of size 10) with:

a

Separate chaining

✓ **Answer**

```
let input = [4371, 1323, 6173, 4199, 4344, 9679, 1989]
let output = new Array(10)
for (let i of input) {
        output[i % 10] = [ ...(output[i % 10] || []), i]
        console.log(JSON.stringify(output))
}
```

```
[null,[4371],null,null,null,null,null,null,null,null]
[null,[4371],null,[1323],null,null,null,null,null,null]
[null,[4371],null,[1323,6173],null,null,null,null,null,null]
[null,[4371],null,[1323,6173],null,null,null,null,null,[4199]]
[null,[4371],null,[1323,6173],[4344],null,null,null,null,[4199]]
[null,[4371],null,[1323,6173],[4344],null,null,null,null,
[4199,9679]]
[null,[4371],null,[1323,6173],[4344],null,null,null,null,
[4199,9679,1989]]
```

b

Open addressing with linear probing

✓ **Answer**

```
let input = [4371, 1323, 6173, 4199, 4344, 9679, 1989]
const SIZE = 10
let output = new Array(SIZE)
for (let i of input) {
        let p = i % SIZE
        for (let j = 0; j < SIZE && output[p]; j++) {
                p = (p + 1) % SIZE
        }
        if (output[p]) {
                throw "HashTable is full"
        }
        output[p] = i
```

```
        console.log(JSON.stringify(output))
}
```

```
[null,4371,null,null,null,null,null,null,null,null]
[null,4371,null,1323,null,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,4199]
[null,4371,null,1323,6173,4344,null,null,null,4199]
[9679,4371,null,1323,6173,4344,null,null,null,4199]
[9679,4371,1989,1323,6173,4344,null,null,null,4199]
```

c

Open addressing with quadratic probing

✓ **Answer**

```
let input = [4371, 1323, 6173, 4199, 4344, 9679, 1989]
const SIZE = 10
let output = new Array(SIZE)
for (let i of input) {
        let p = i % SIZE
        for (let j = 0; j < SIZE; j++) {
                let ind = (p + (j ** 2)) % SIZE
                if (!output[ind]) {
                        p = ind
                        break
                }
        }
        if (output[p]) {
                throw "HashTable is full"
        }
        output[p] = i
        console.log(JSON.stringify(output))
}
```

```
[null,4371,null,null,null,null,null,null,null,null]
[null,4371,null,1323,null,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,4199]
```

```
[null,4371,null,1323,6173,4344,null,null,null,4199]
[9679,4371,null,1323,6173,4344,null,null,null,4199]
[9679,4371,null,1323,6173,4344,null,null,1989,4199]
```

d

Open addressing with double hashing and the secondary hash function h2(x) = 7 - (x mod 7)

✓ **Answer**

```
let input = [4371, 1323, 6173, 4199, 4344, 9679, 1989]
const SIZE = 10
let output = new Array(SIZE)
for (let i of input) {
        let p = i % SIZE
        let h2 = 7 - (i % 7)
        for (let j = 0; j < SIZE && output[p]; j++) {
                p = (p + h2) % SIZE
        }
        if (output[p]) {
                throw "HashTable is full"
        }
        output[p] = i
        console.log(JSON.stringify(output))
}
```

```
[null,4371,null,null,null,null,null,null,null,null]
[null,4371,null,1323,null,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,null]
[null,4371,null,1323,6173,null,null,null,null,4199]
[null,4371,null,1323,6173,null,null,4344,null,4199]
[null,4371,null,1323,6173,9679,null,4344,null,4199]
HashTable is full
```

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

✓ **Answer**

```
[9, 8, 8, 5, 7, 7, 4, 4, 4, 2]
[2, 8, 8, 5, 7, 7, 4, 4, 4, 9]
[2, 4, 8, 5, 7, 7, 8, 4, 4, 9]
[2, 4, 4, 5, 7, 7, 8, 8, 4, 9]
[2, 4, 4, 4, 7, 7, 8, 8, 5, 9]
[2, 4, 4, 4, 5, 7, 8, 8, 7, 9]
[2, 4, 4, 4, 5, 7, 8, 8, 7, 9]
[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
```

b

Insertion sort

✓ **Answer**

```
[9, 8, 8, 5, 7, 7, 4, 4, 4, 2]
[8, 9, 8, 5, 7, 7, 4, 4, 4, 2]
[8, 8, 9, 5, 7, 7, 4, 4, 4, 2]
[5, 8, 8, 9, 7, 7, 4, 4, 4, 2]
[5, 7, 8, 8, 9, 7, 4, 4, 4, 2]
[5, 7, 7, 8, 8, 9, 4, 4, 4, 2]
[4, 5, 7, 7, 8, 8, 9, 4, 4, 2]
[4, 4, 5, 7, 7, 8, 8, 9, 4, 2]
[4, 4, 4, 5, 7, 7, 8, 8, 9, 2]
[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
```

c

Quicksort, by partitioning around the last element

✓ **Answer**

Where `-` are subinvocations and `→` are returns

```
[9, 8, 8, 5, 7, 7, 4, 4, 4, 2]
├─2
├─[]
├─[9, 8, 8, 5, 7, 7, 4, 4, 4]
| ├─4
| ├─[]
| ├─[9, 8, 8, 5, 7, 7, 4, 4]
| | ├─4
| | ├─[]
| | ├─[9, 8, 8, 5, 7, 7, 4]
| | | ├─4
| | | ├─[]
| | | ├─[9, 8, 8, 5, 7, 7]
| | | | ├─7
| | | | ├─[5]
| | | | | ├─⟼5
| | | | ├─[9, 8, 8, 7]
| | | | | ├─7
| | | | | ├─[]
| | | | | ├─[9, 8, 8]
| | | | | | ├─8
| | | | | | ├─[]
| | | | | | ├─[9, 8]
| | | | | | | ├─8
| | | | | | | ├─[]
| | | | | | | ├─[9]
| | | | | | | | ├─⟼9
| | | | | | | ├─⟼[8, 9]
| | | | | | ├─⟼[8, 8, 9]
| | | | | ├─⟼[7, 8, 8, 9]
| | | | ├─⟼[5, 7, 7, 8, 8, 9]
| | | ├─⟼[4, 5, 7, 7, 8, 8, 9]
| | ├─⟼[4, 4, 5, 7, 7, 8, 8, 9]
| ├─⟼[4, 4, 4, 5, 7, 7, 8, 8, 9]
├─⟼[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
```

d

Mergesort

✓ **Answer**

Where ⊢ are subinvocations and ⟼ are returns

```
[9, 8, 8, 5, 7, 7, 4, 4, 4, 2]
├─[9, 8, 8, 5, 7]
│ ├─[9, 8, 8]
│ │ ├─[9, 8]
│ │ │ ├─→[8, 9]
│ │ ├─[8]
│ │ │ ├─→[8]
│ │ ├─→[8, 8, 9]
│ ├─[5, 7]
│ │ ├─→[5, 7]
│ ├─→[5, 7, 8, 8, 9]
├─[7, 4, 4, 4, 2]
│ ├─[7, 4, 4]
│ │ ├─[7, 4]
│ │ │ ├─→[4, 7]
│ │ ├─[4]
│ │ │ ├─→[4]
│ │ ├─→[4, 4, 7]
│ ├─[4, 2]
│ │ ├─→[2, 4]
│ ├─→[2, 4, 4, 4, 7]
├─→[2, 4, 4, 4, 5, 7, 7, 8, 8, 9]
```