PSO 6

Linear-{Sorts,Hashing}

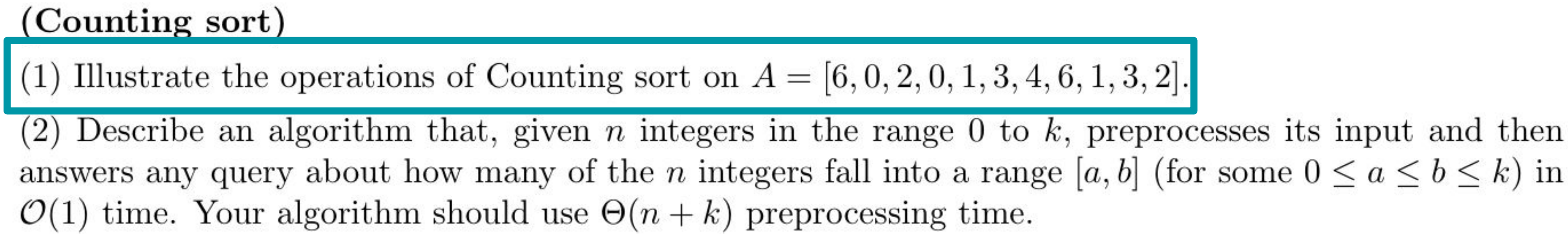
PrePSOGeoGussr: Guess the city

Announcements

Hw due this week

Midterm next week

There is a practice exam on Ed



(Counting sort)

(1) Illustrate the operations of Counting sort on $A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]$.

(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

Step 1: Array C keeps the number of occurrences for each element in A.

Step 2: Count the occurrences of each item in A. Use A[i] as the indices of C.

Step 3: Accumulate the count values in C from left to right.

Step 4: Use values in C to determine the final index for each element in A.

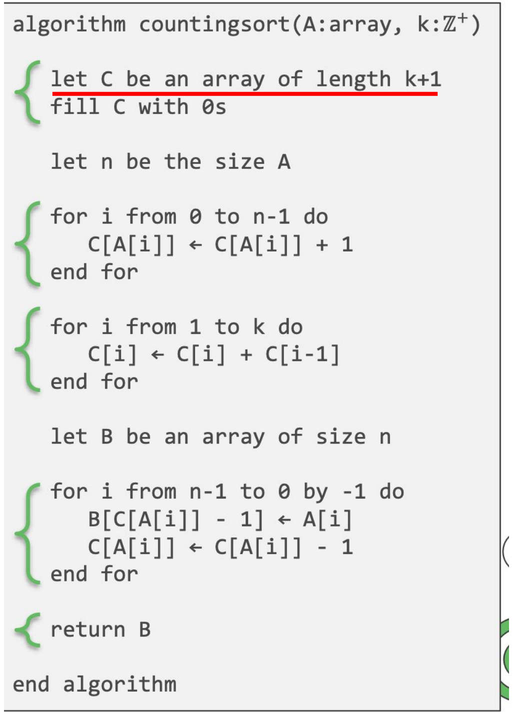
Step 5 (optional): Copy the elements from B to A if they must be in the original array.

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	0

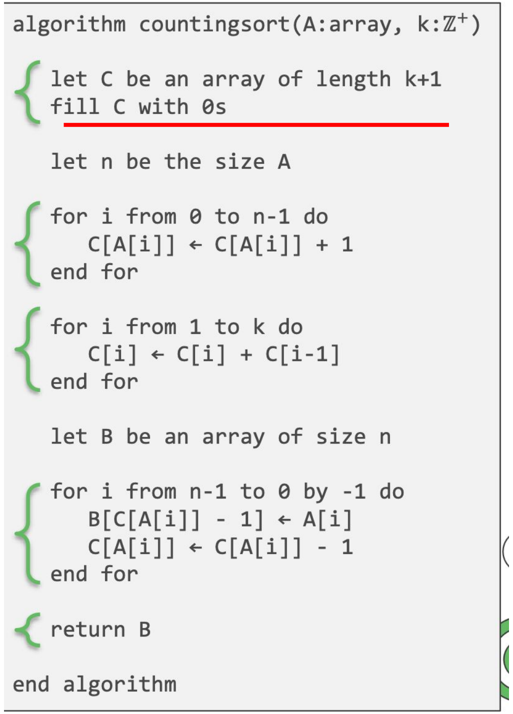


```
algorithm countingsort(A:array, k:ℤ+)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	0



```
algorithm countingsort(A:array, k:ℤ⁺)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	0

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

↑

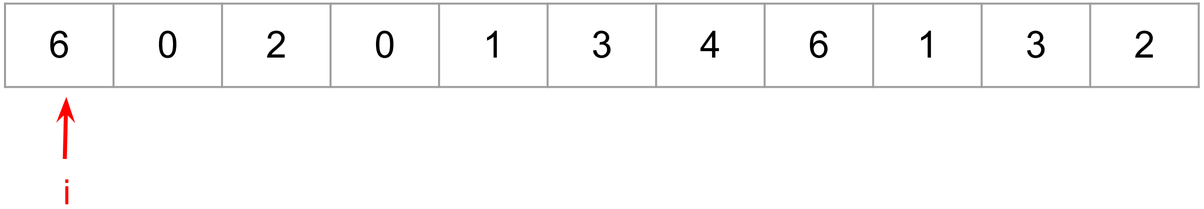
i

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	0

A[i] = 6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

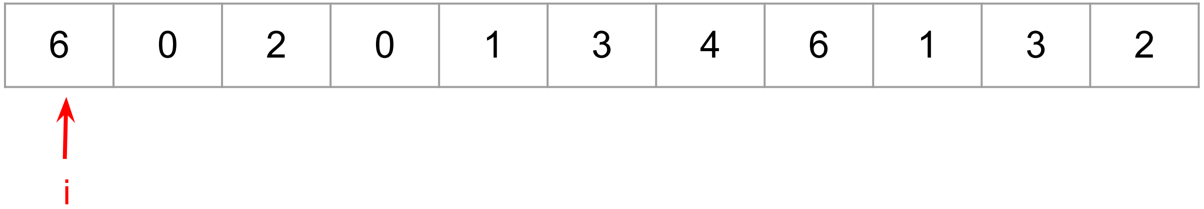
Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	0

A[i] = 6

C[A[i]] = C[6]

```
algorithm countingsort(A:array, k:ℤ⁺)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

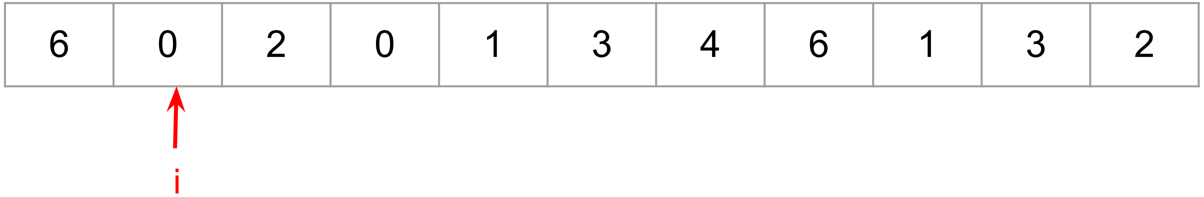
k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	1

A[i] = 6

C[A[i]] = C[6]

C[6] += 1

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```

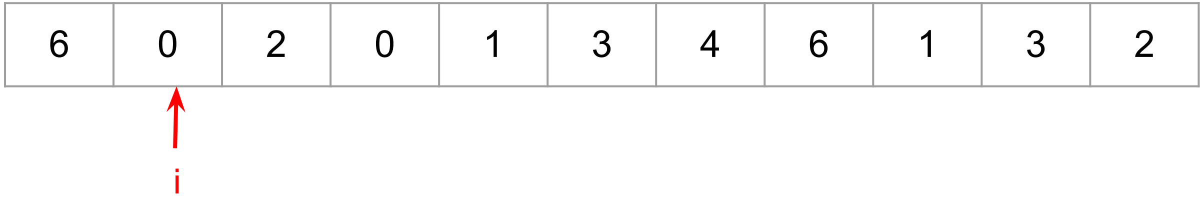


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	1

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



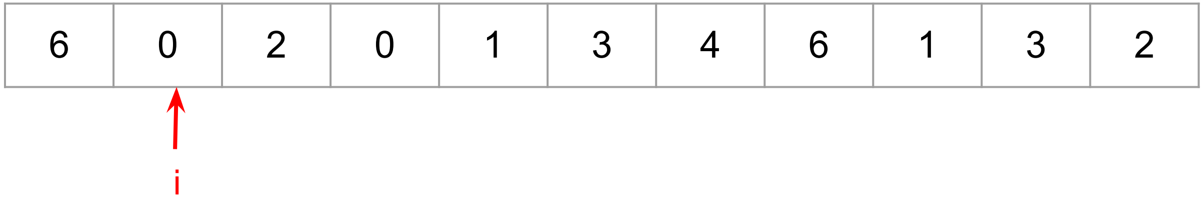
6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	0	0	0	0	0	0	1

A[i] = 0

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

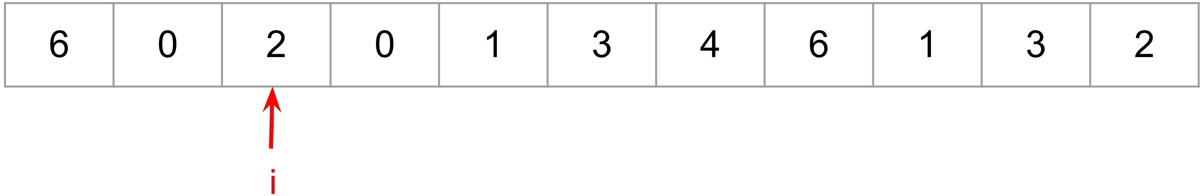
Initialize our C array

k	0	1	2	3	4	5	6
freq	1	0	0	0	0	0	1

$A[i] = 0$

$C[0] += 1$

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



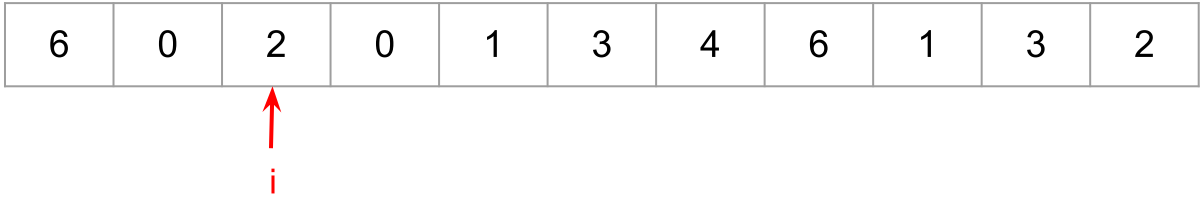
6	0	2	0	1	3	4	6	1	3	2

i

Initialize our C array

k	0	1	2	3	4	5	6
freq	1	0	0	0	0	0	1

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

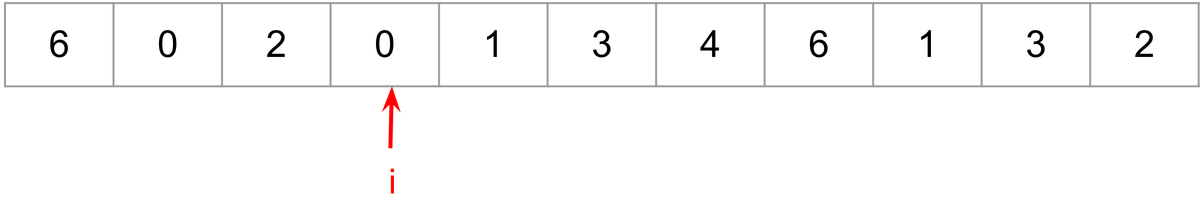


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	1	0	1	0	0	0	1

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

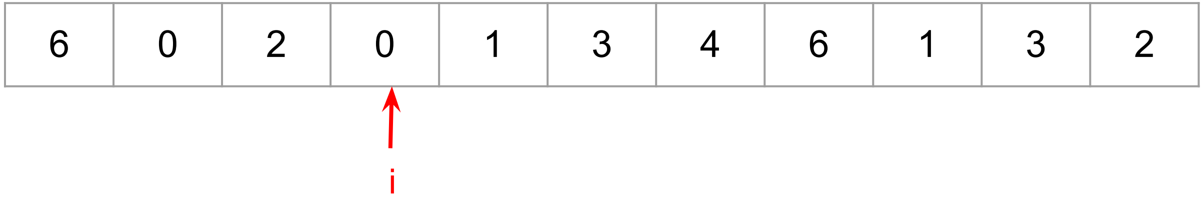


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	1	0	1	0	0	0	1

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



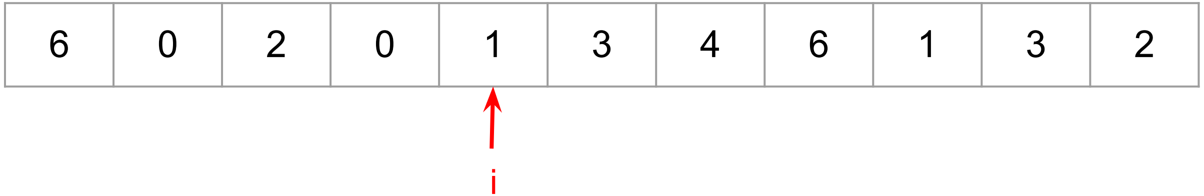
6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	0	1	0	0	0	1

Going faster..

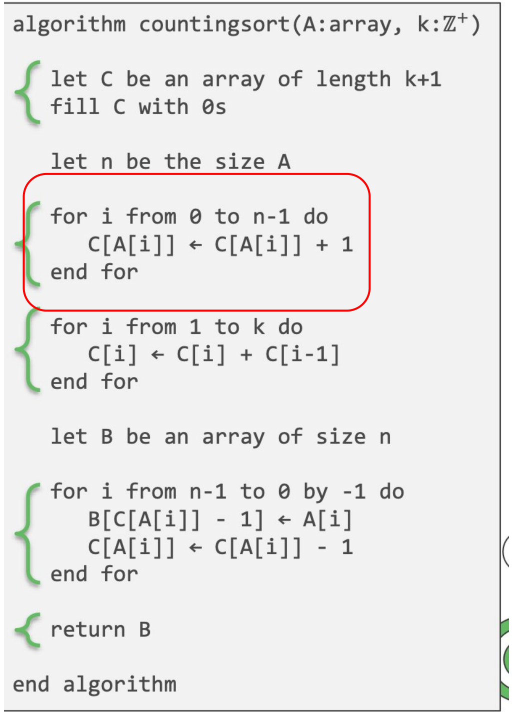
```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



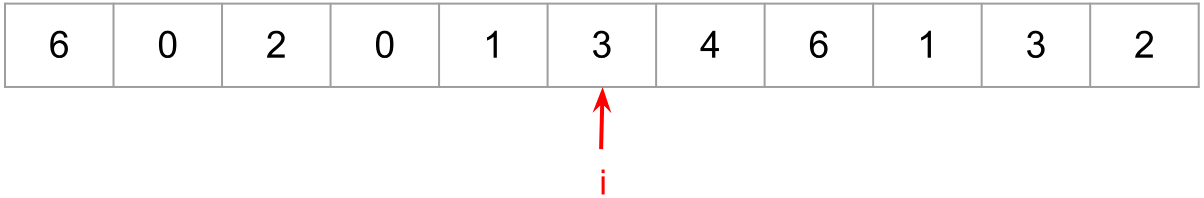
6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	1	1	0	0	0	1



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

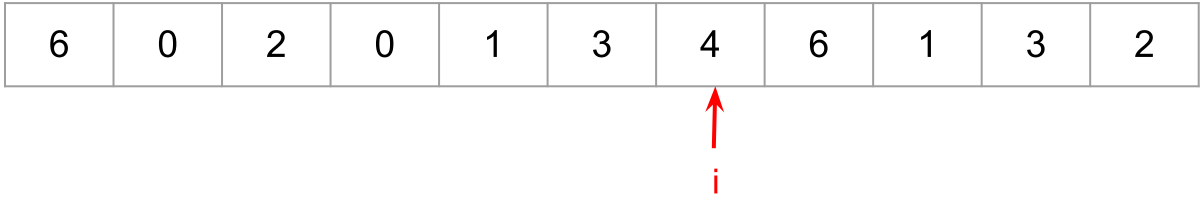


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	1	1	1	0	0	1

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

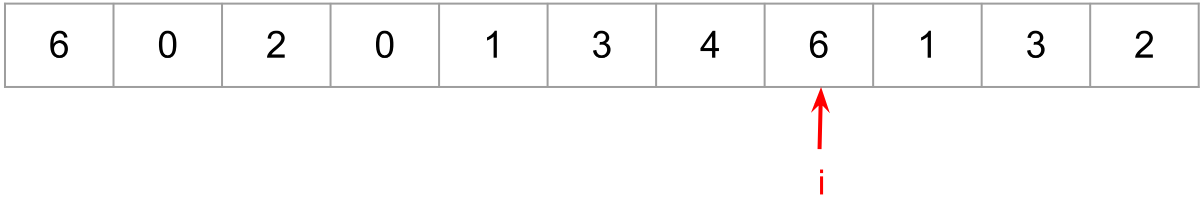


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	1	1	1	1	0	1

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

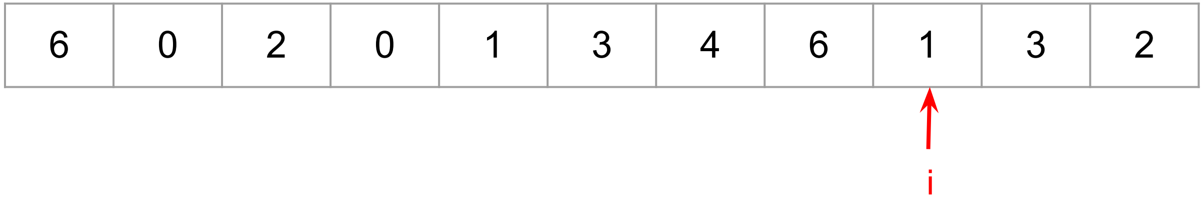


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	1	1	1	1	0	2

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

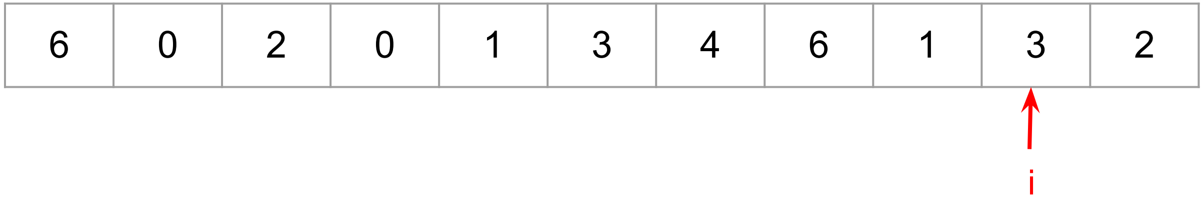


6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	2	1	1	1	0	2

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	2	1	2	1	0	2

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	2	2	2	1	0	2

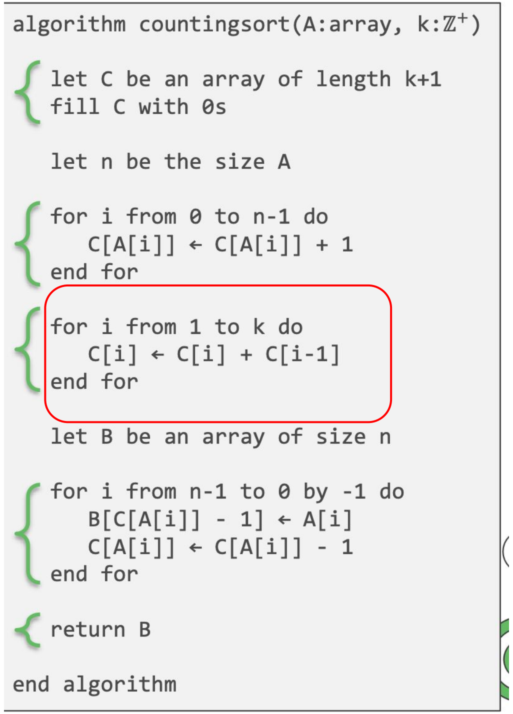
```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

Initialize our C array

k	0	1	2	3	4	5	6
freq	2	2	2	2	1	0	2

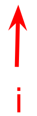
Next up



```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	2	2	2	1	0	2



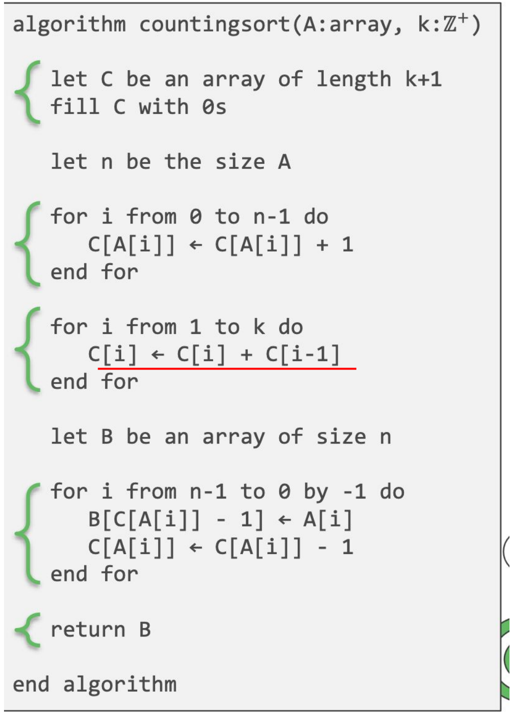
```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	2	2	2	1	0	2

i

C[1] = C[1] + C[0]



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	2	2	1	0	2

↑
i

$$C[1] = C[1] + C[0] = 2 + 2$$

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	2	2	1	0	2

↑

i

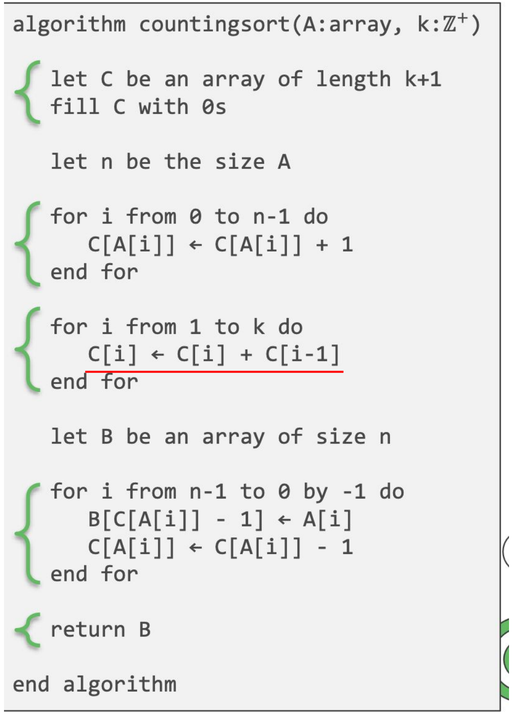
```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	2	2	1	0	2

↑
i

C[2] += C[1]



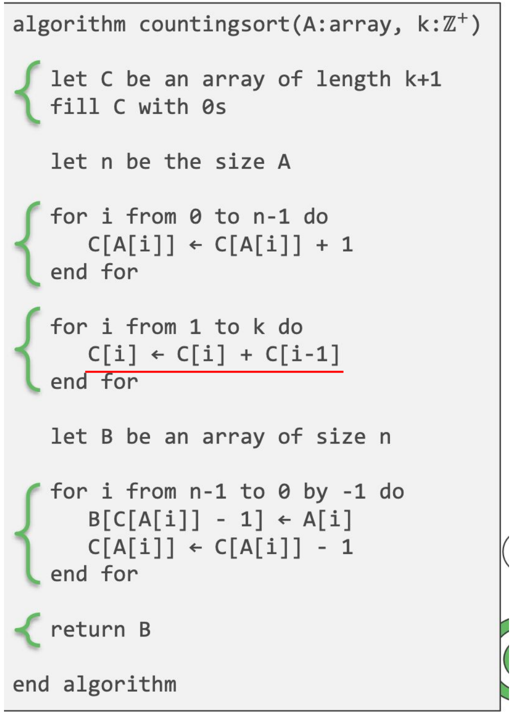
```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	2	1	0	2

i

C[2] += C[1] = 6



```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	2	1	0	2

↑

i

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

C[2] += C[1] = 6

C[i] = # of elements <= i in the sorted array

6	0	2	0	1	3	4	6	1	3	2

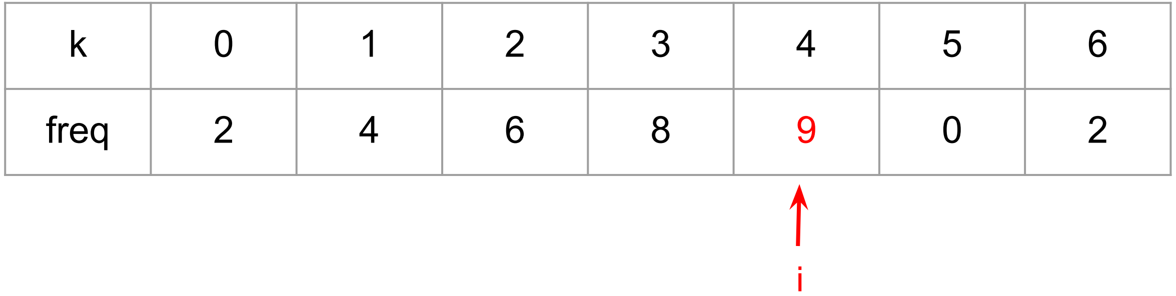
k	0	1	2	3	4	5	6
freq	2	4	6	8	1	0	2

↑

i

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	0	2

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	2

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

i

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

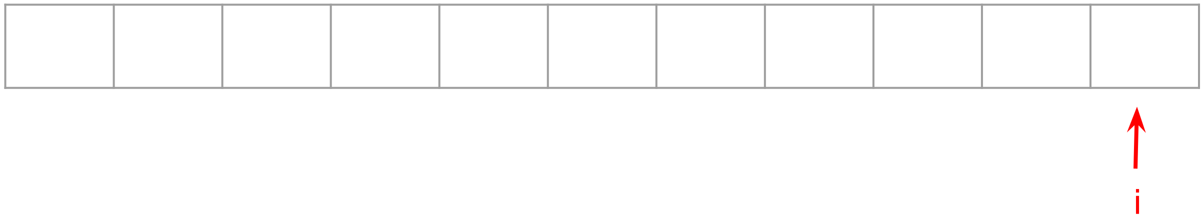
B


```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B



```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

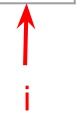
k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

A[i] = 2

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

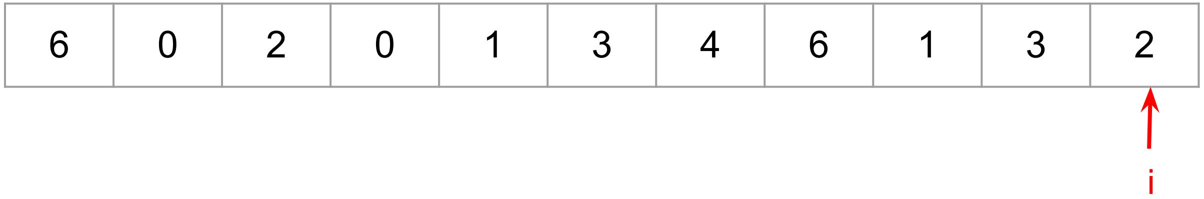


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

A[i] = 2
C[A[i]] = 6

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

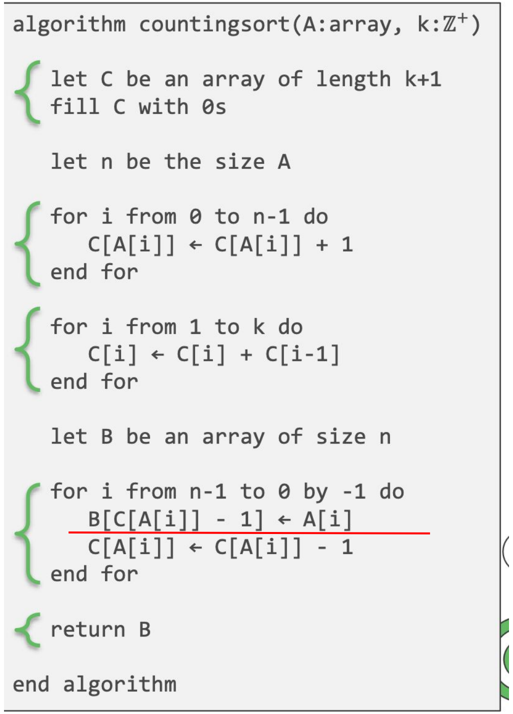


6	0	2	0	1	3	4	6	1	3	2

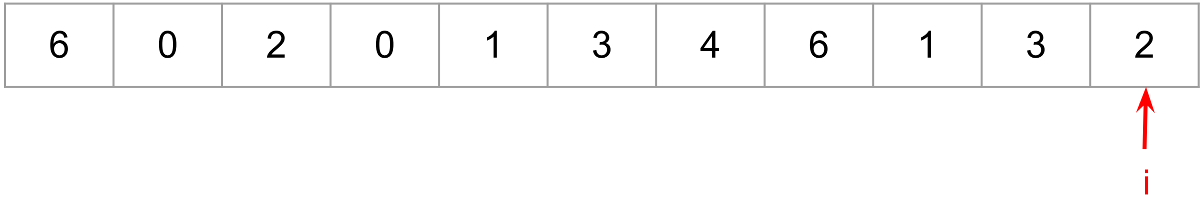
k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

A[i] = 2
C[A[i]] = 6



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

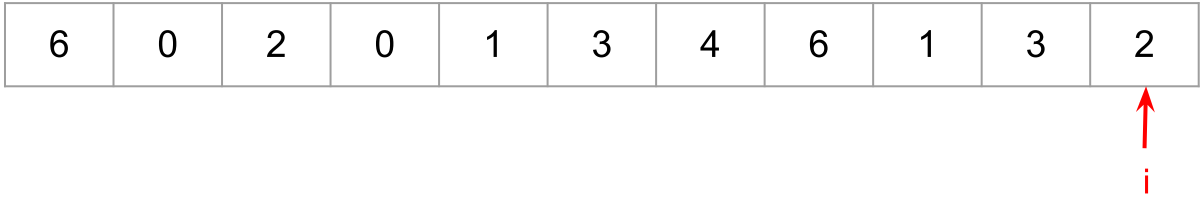
B

A[i] = 2

C[A[i]] = 6

B[C[A[i]] - 1] = B[5]

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

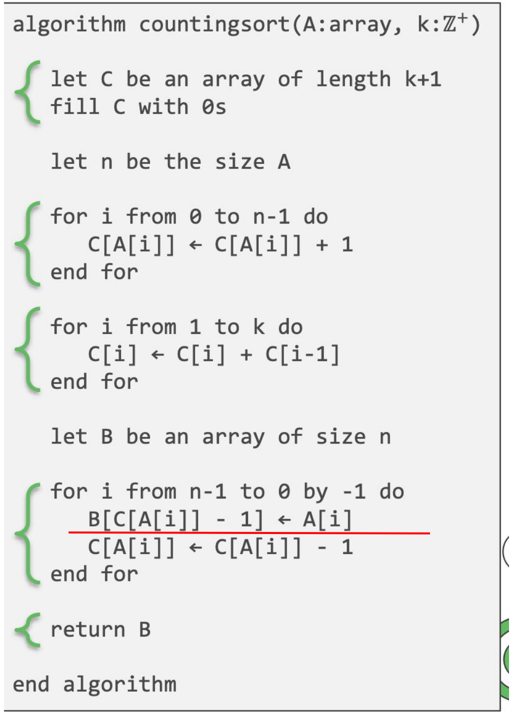
B

					2					

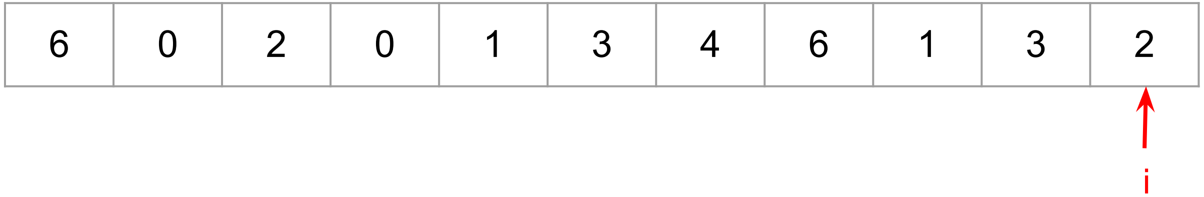
A[i] = 2

C[A[i]] = 6

B[C[A[i]] - 1] = B[5]   Set B[5] = A[i] = 2



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

C[i] = # elements less than or equal to i

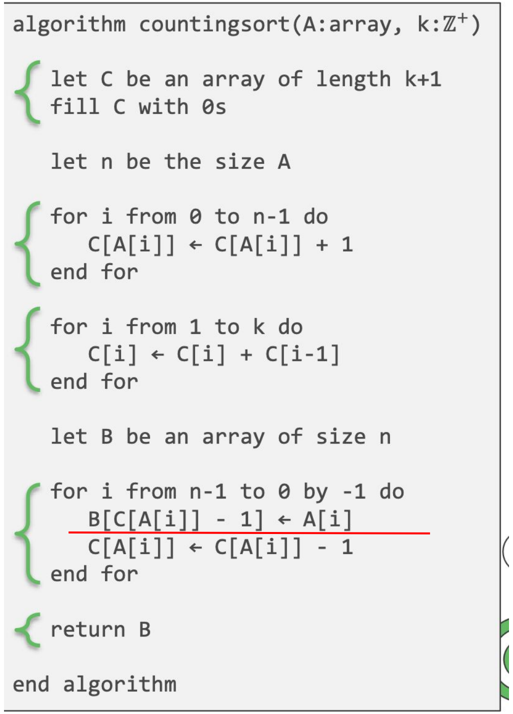
B

					2					

A[i] = 2

C[A[i]] = 6

B[C[A[i]] - 1] = B[5]   Set B[5] = A[i] = 2 **why?**



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

C[i] = # elements less than or equal to i

When sorted, elements before B[5] look like..

B

0	0	1	1	2	2					

A[i] = 2

C[A[i]] = 6

B[C[A[i]] - 1] = B[5]   Set B[5] = A[i] = 2 **why?**

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

0	0	1	1	2	2					

A[i] = 2

```
algorithm countingsort(A:array, k:ℤ⁺)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
       C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
       C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
       B[C[A[i]] - 1] ← A[i]
       C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```

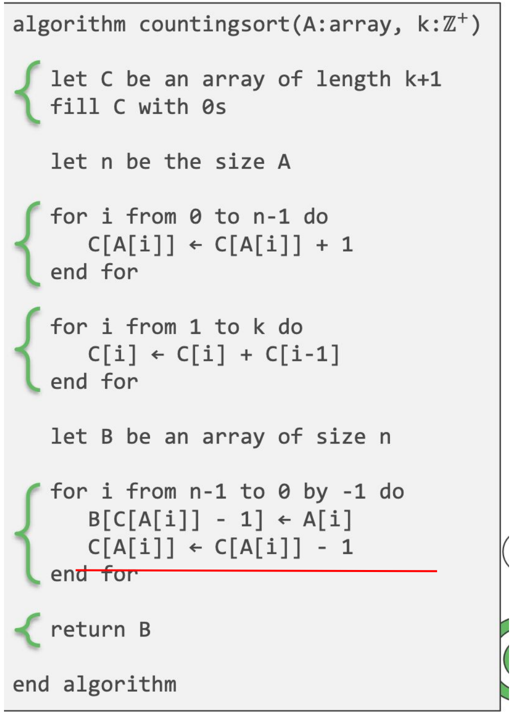
6	0	2	0	1	3	4	6	1	3	2
										↑ i

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

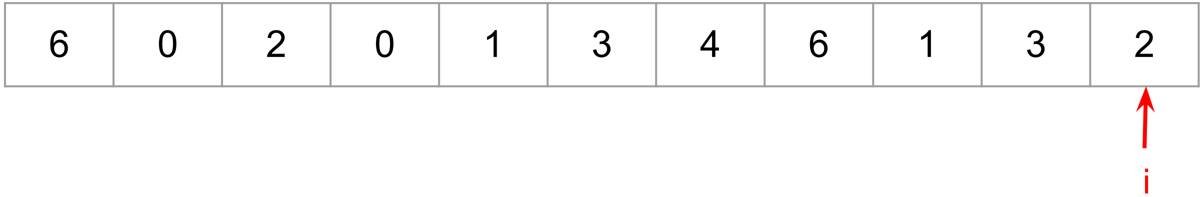
B

0	0	1	1	2	2					

A[i] = 2
C[A[i]] = C[2]



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2					

$A[i] = 2$

$C[A[i]] = C[2]$ **$C[2] \mathrel{-}= 1$**

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2
										↑ i

Intuition: We placed the first 2 down, only one 2 left

k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

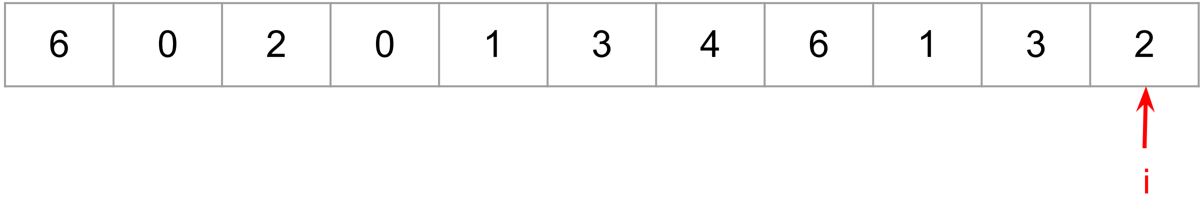
B

0	0	1	1	2	2					

A[i] = 2

C[A[i]] = C[2]   **C[2] -= 1**

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

Intuition: We placed the first 2 down, only **C[2] - C[1]** 2 left

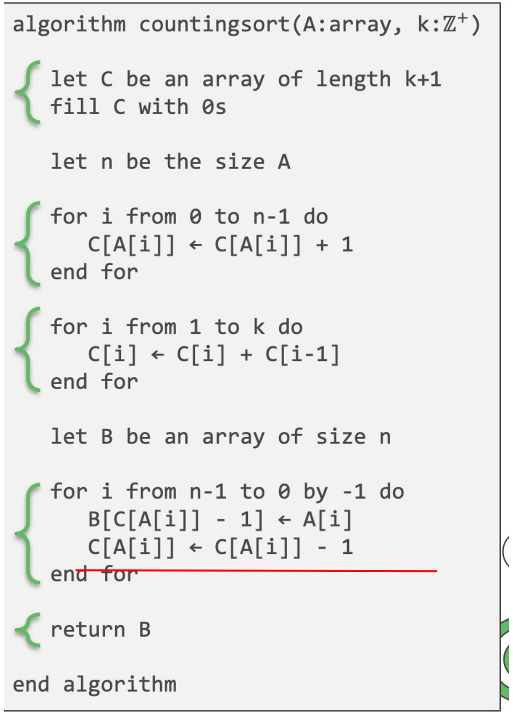
k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

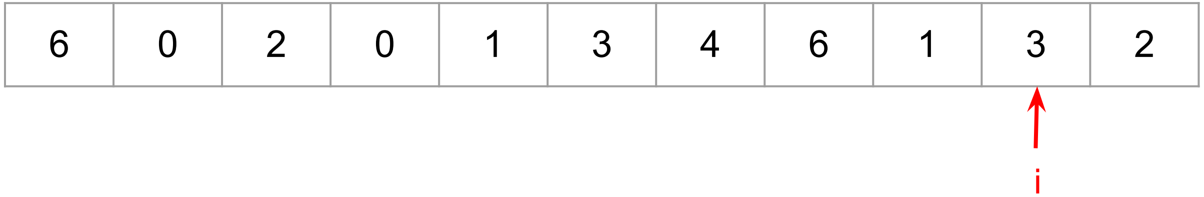
0	0	1	1	2	2					

A[i] = 2

C[A[i]] = C[2] **C[2] -= 1**



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2					

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

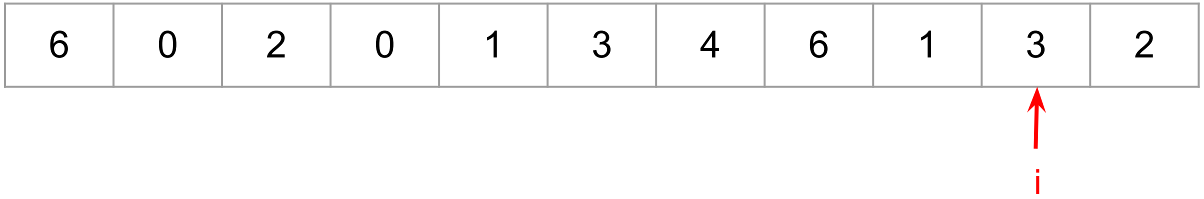
k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2					

A[i] = 3

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

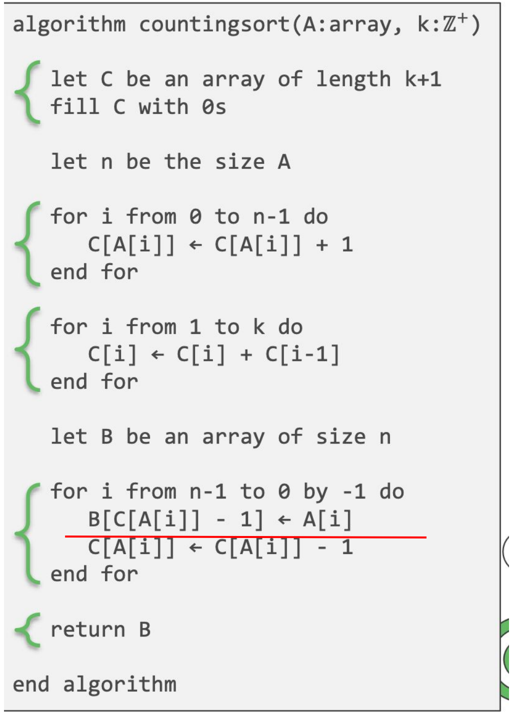
k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

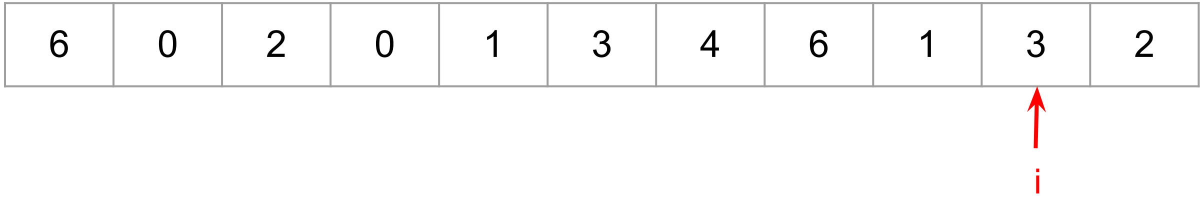
0	0	1	1	2	2					

A[i] = 3

C[A[i]] = C[3] = 8



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

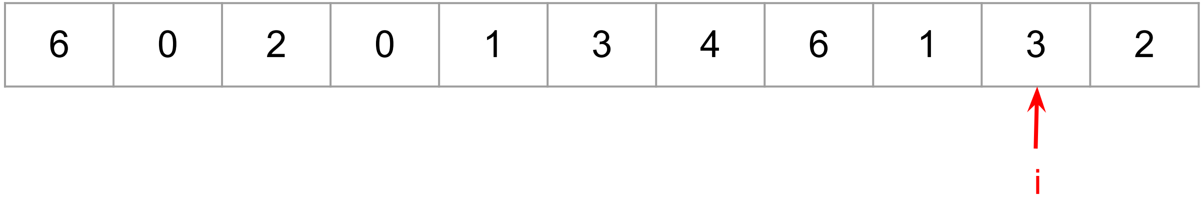
k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2					

A[i] = 3
C[A[i]] = C[3] = 8
Set B[8 - 1] = 3

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

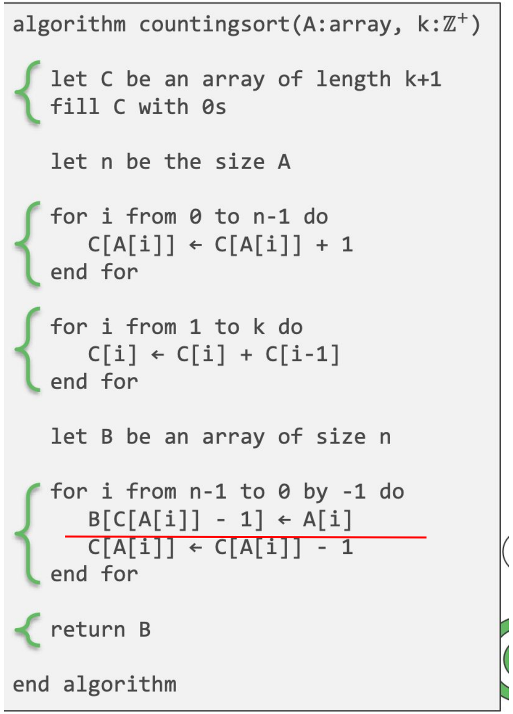
0	0	1	1	2	2		3			

A[i] = 3

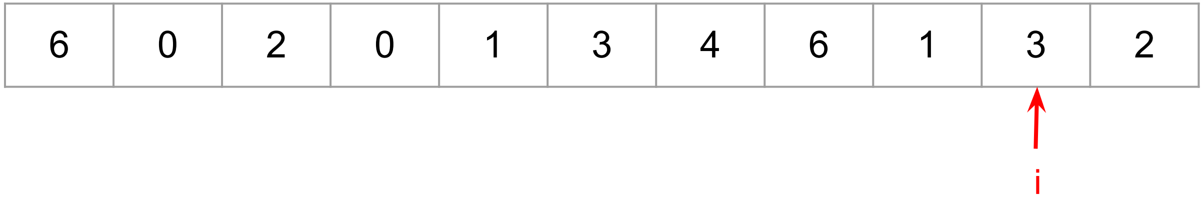
C[A[i]] = C[3] = 8

Set B[8 - 1] = 3

why?



```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2	3	3			

A[i] = 3
C[A[i]] = C[3] = 8
Set B[8 - 1] = 3

why?

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

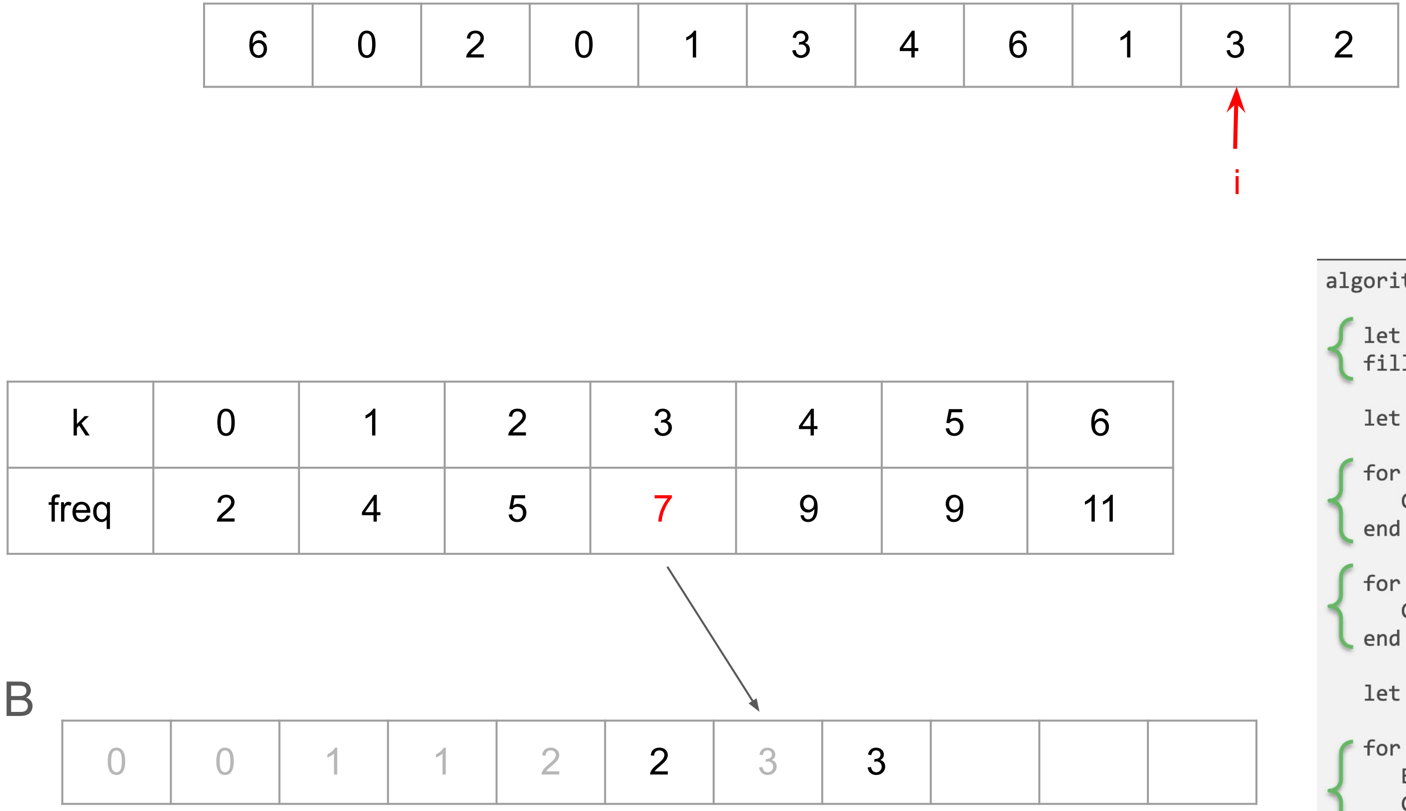
k	0	1	2	3	4	5	6
freq	2	4	5	8	9	9	11

B

0	0	1	1	2	2	3	3			

A[i] = 3
C[A[i]] = C[3]

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

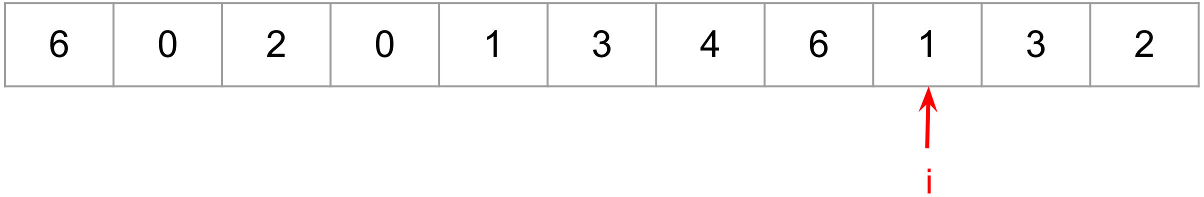
k	0	1	2	3	4	5	6
freq	2	4	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

A[i] = 3
C[A[i]] = C[3]
C[3] -= 1

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



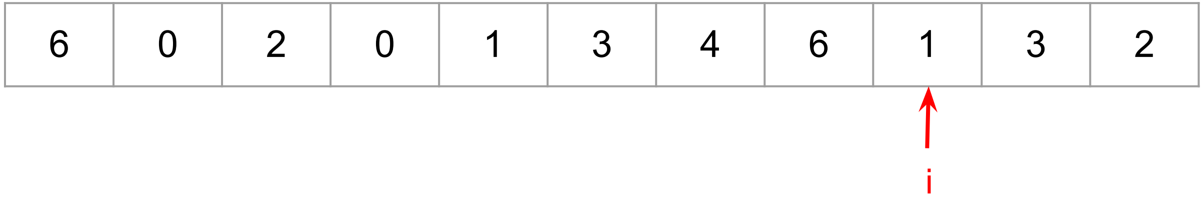
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

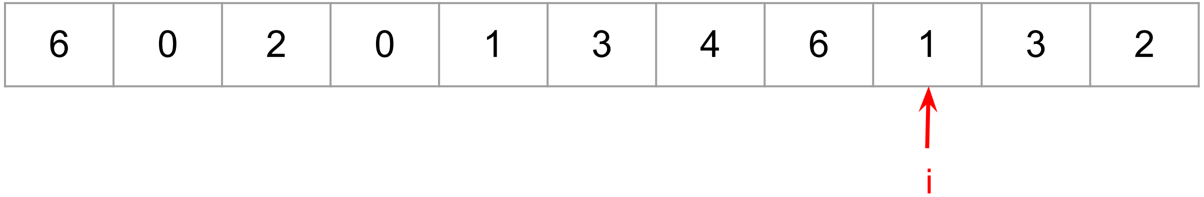
k	0	1	2	3	4	5	6
freq	2	4	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

This should be B[3] from our picture

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	4	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

This should be B[3] from our picture
A[i] = 1, C[A[i]] = 4, so true

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

↑
i

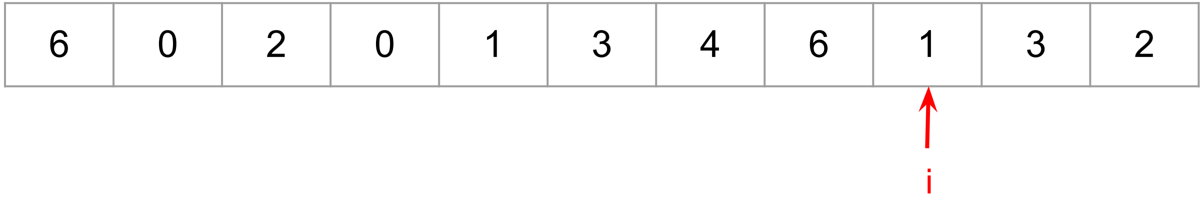
k	0	1	2	3	4	5	6
freq	2	4	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

This should be B[3] from our picture
A[i] = 1, C[A[i]] = 4, so true

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

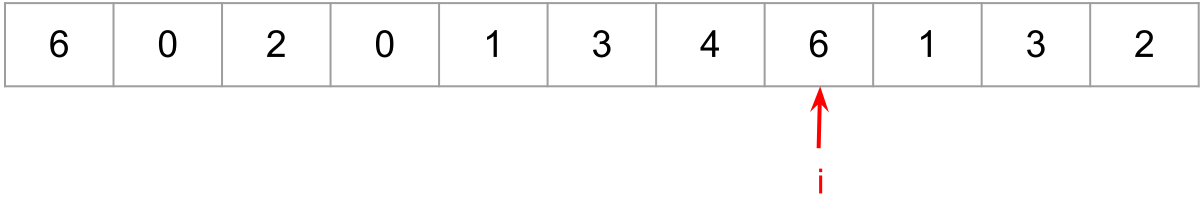
k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

This should be B[3] from our picture
A[i] = 1, C[A[i]] = 4, so true
Then decrement the count

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



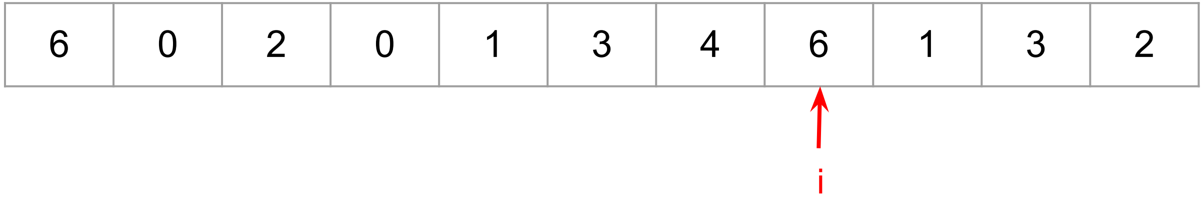
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

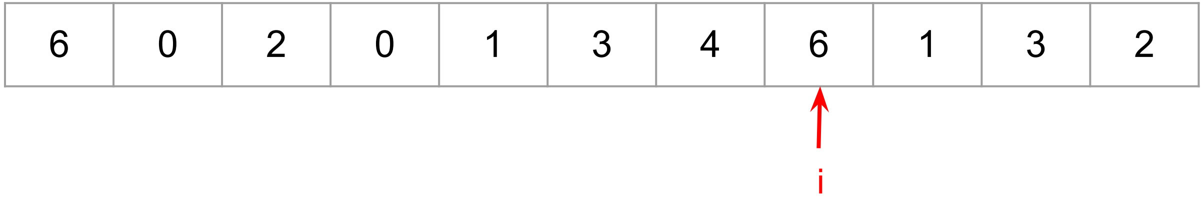
k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	11

B

0	0	1	1	2	2	3	3			

C[6] = 11

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	11

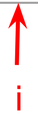
B

0	0	1	1	2	2	3	3			6

C[6] = 11

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2



k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	11

B

0	0	1	1	2	2	3	3	4	6	6

C[6] = 11

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

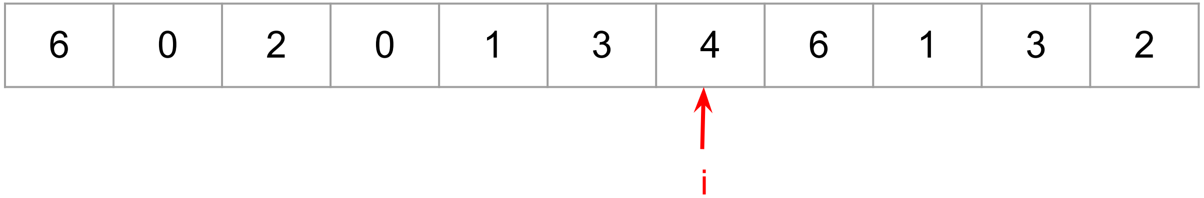
i

k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



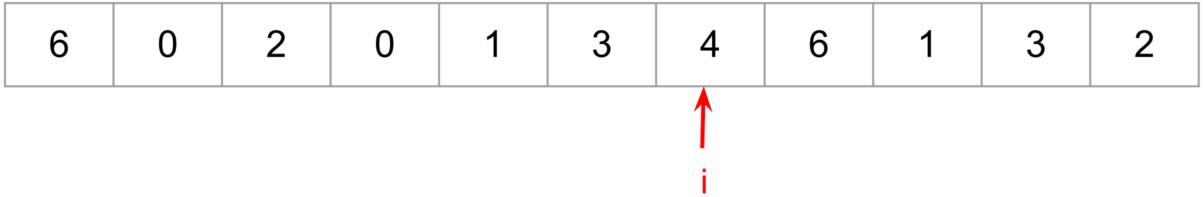
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



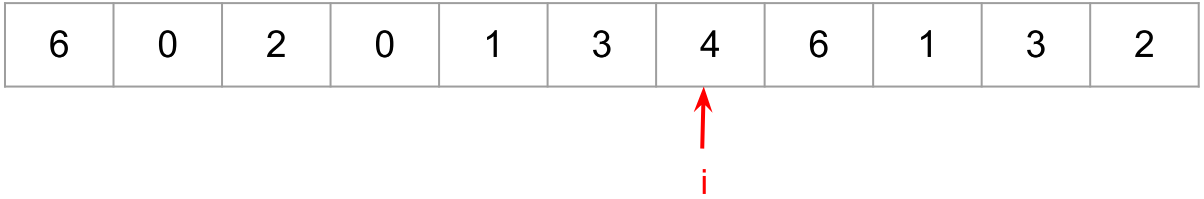
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	9	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

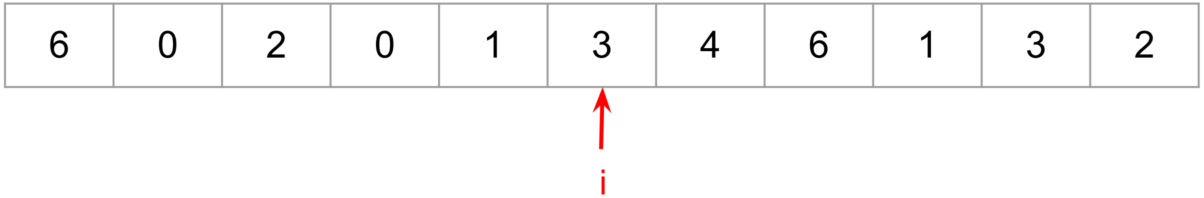
i

k	0	1	2	3	4	5	6
freq	2	3	5	7	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



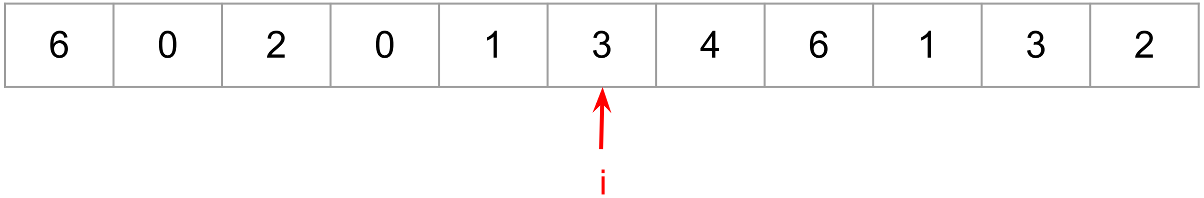
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	7	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

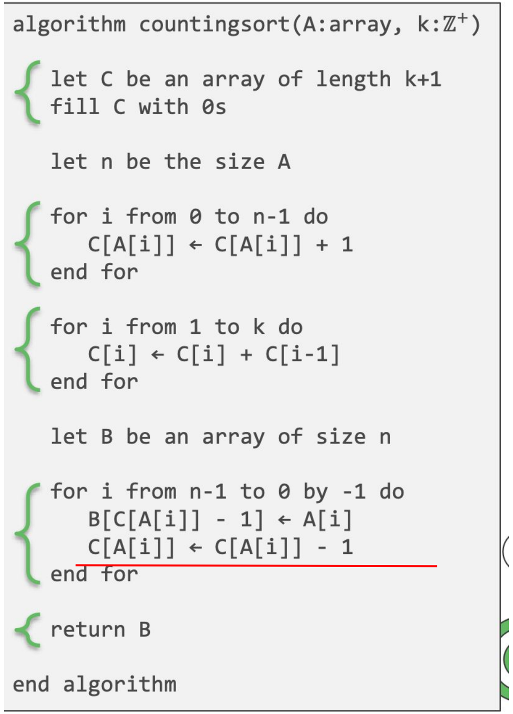


6	0	2	0	1	3	4	6	1	3	2

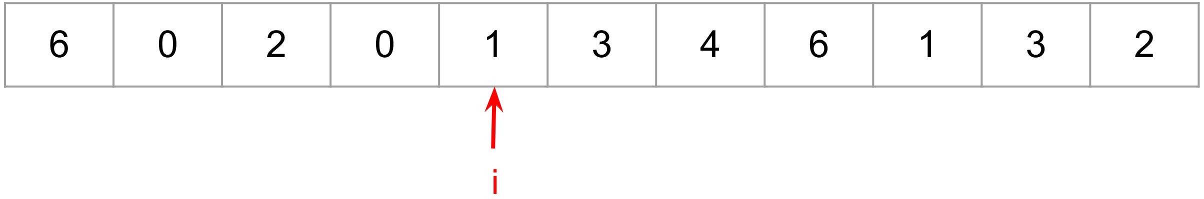
k	0	1	2	3	4	5	6
freq	2	3	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6



```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	3	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

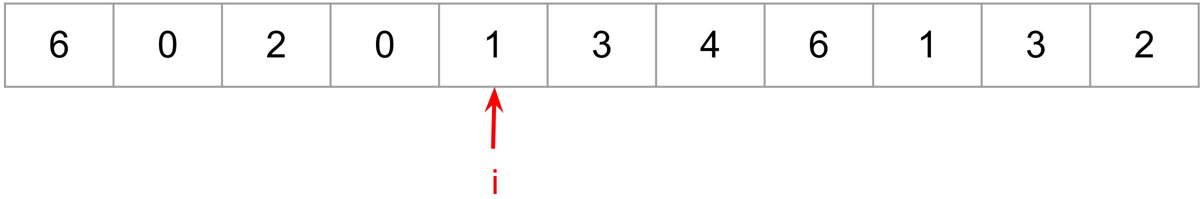
i

k	0	1	2	3	4	5	6
freq	2	3	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



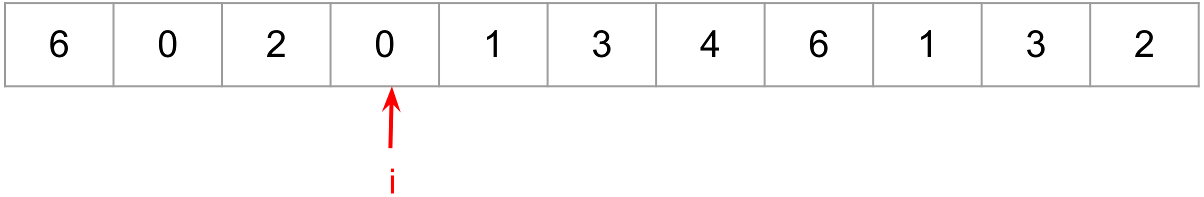
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	2	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



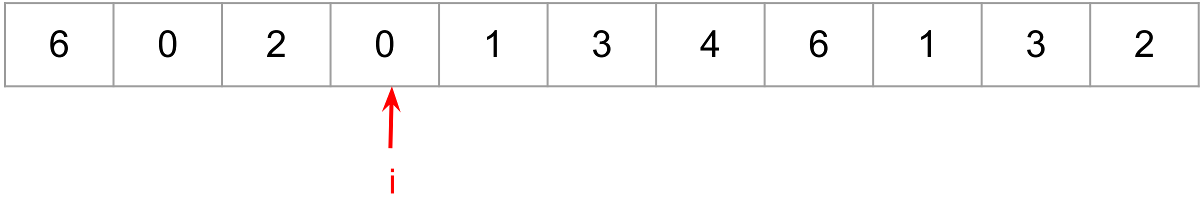
6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	2	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	2	2	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

let C be an array of length k+1
fill C with 0s

let n be the size A

for i from 0 to n-1 do
   C[A[i]] ← C[A[i]] + 1
end for

for i from 1 to k do
   C[i] ← C[i] + C[i-1]
end for

let B be an array of size n

for i from n-1 to 0 by -1 do
   B[C[A[i]] - 1] ← A[i]
   C[A[i]] ← C[A[i]] - 1
end for

return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

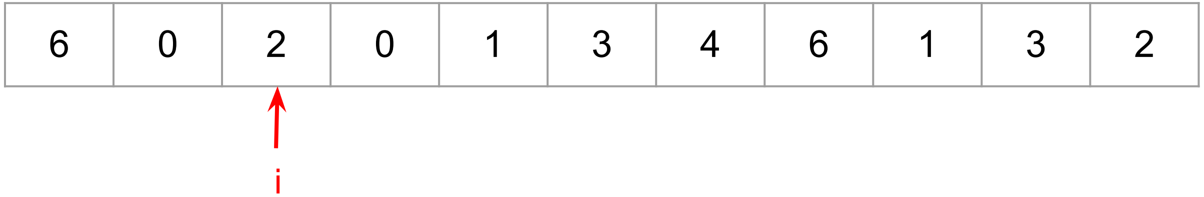
i

k	0	1	2	3	4	5	6
freq	1	2	5	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ+)

   let C be an array of length k+1
   fill C with 0s

   let n be the size A

   for i from 0 to n-1 do
      C[A[i]] ← C[A[i]] + 1
   end for

   for i from 1 to k do
      C[i] ← C[i] + C[i-1]
   end for

   let B be an array of size n

   for i from n-1 to 0 by -1 do
      B[C[A[i]] - 1] ← A[i]
      C[A[i]] ← C[A[i]] - 1
   end for

   return B

end algorithm
```



6	0	2	0	1	3	4	6	1	3	2

k	0	1	2	3	4	5	6
freq	1	2	4	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

    let C be an array of length k+1
    fill C with 0s

    let n be the size A

    for i from 0 to n-1 do
        C[A[i]] ← C[A[i]] + 1
    end for

    for i from 1 to k do
        C[i] ← C[i] + C[i-1]
    end for

    let B be an array of size n

    for i from n-1 to 0 by -1 do
        B[C[A[i]] - 1] ← A[i]
        C[A[i]] ← C[A[i]] - 1
    end for

    return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

i

k	0	1	2	3	4	5	6
freq	0	2	4	6	8	9	10

B

0	0	1	1	2	2	3	3	4	6	6

```
algorithm countingsort(A:array, k:ℤ⁺)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

6	0	2	0	1	3	4	6	1	3	2

↑
i

k	0	1	2	3	4	5	6
freq	0	2	4	6	8	9	9

B

0	0	1	1	2	2	3	3	4	6	6

Done!

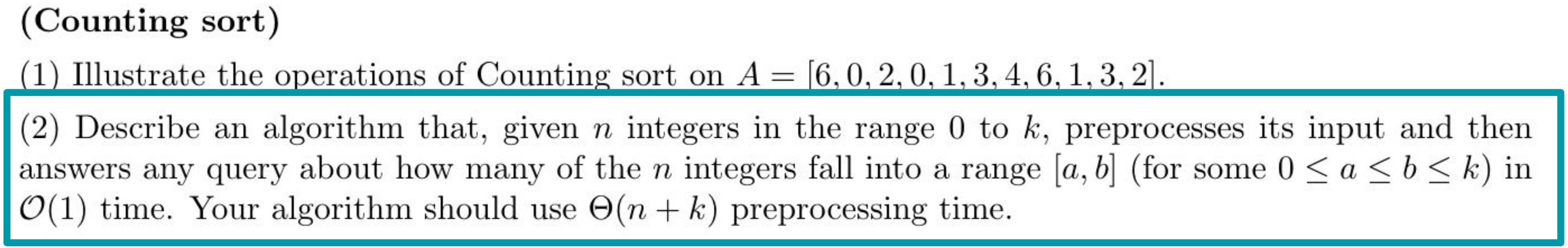
```
algorithm countingsort(A:array, k:$\mathbb{Z}^+$)

  let C be an array of length k+1
  fill C with 0s

  let n be the size A

  for i from 0 to n-1 do
     C[A[i]] ← C[A[i]] + 1
  end for

  for i from 1 to k do
     C[i] ← C[i] + C[i-1]
  end for

  let B be an array of size n

  for i from n-1 to 0 by -1 do
     B[C[A[i]] - 1] ← A[i]
     C[A[i]] ← C[A[i]] - 1
  end for

  return B

end algorithm
```

Question 3

(Counting sort)

(1) Illustrate the operations of Counting sort on $A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]$.

(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

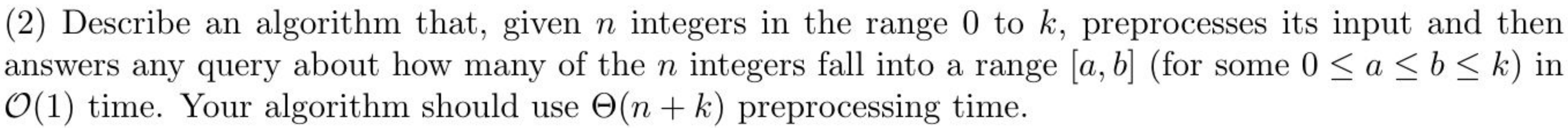


(Counting sort)

(1) Illustrate the operations of Counting sort on $A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]$.

(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

Wait! Sounds familiar..



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

A

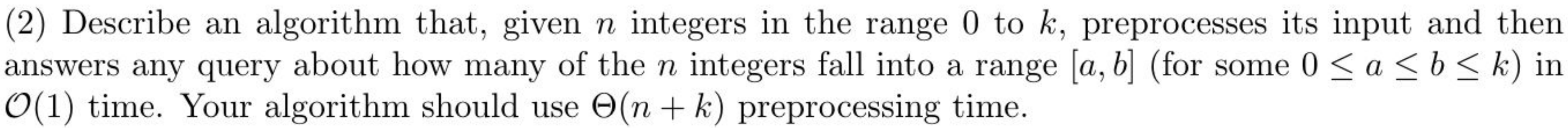
6	0	2	0	1	3	4	6	1	3	2

The counting array kept track of **C[i] = # elements less than or equal to i**

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

0	0	1	1	2	2	3	3	4	6	6



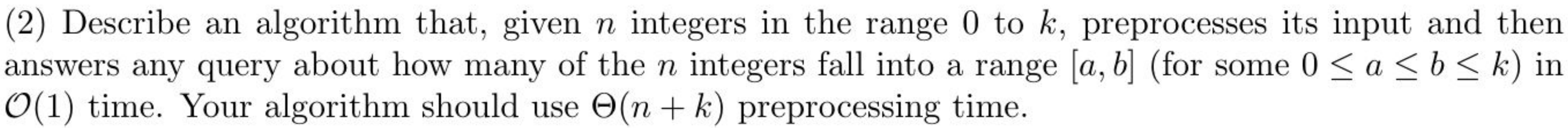
(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

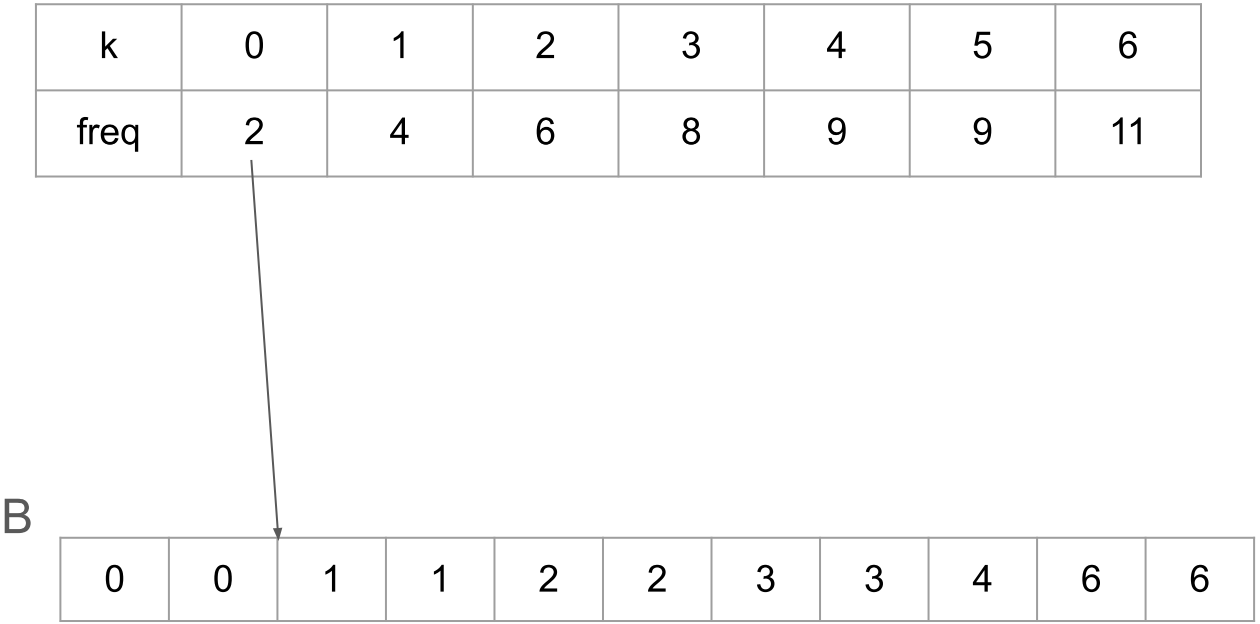
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

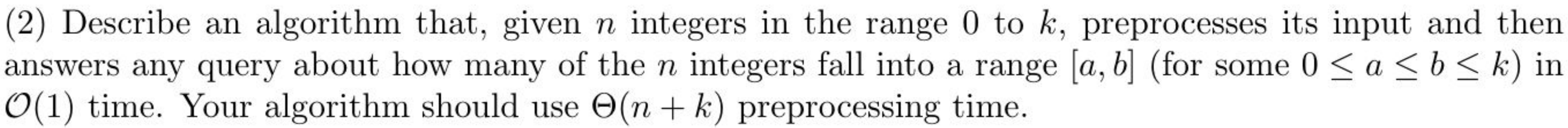
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

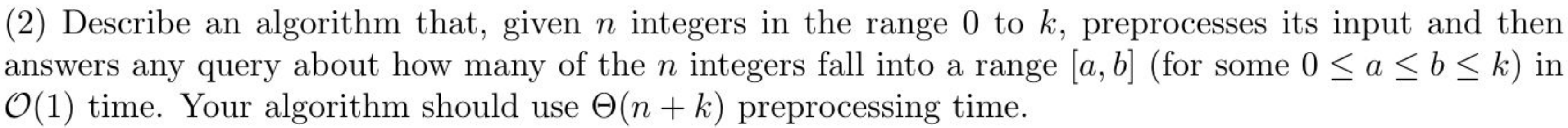
The counting array kept track of **C[i] = # elements less than or equal to i**

k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [0,0] = C[0]

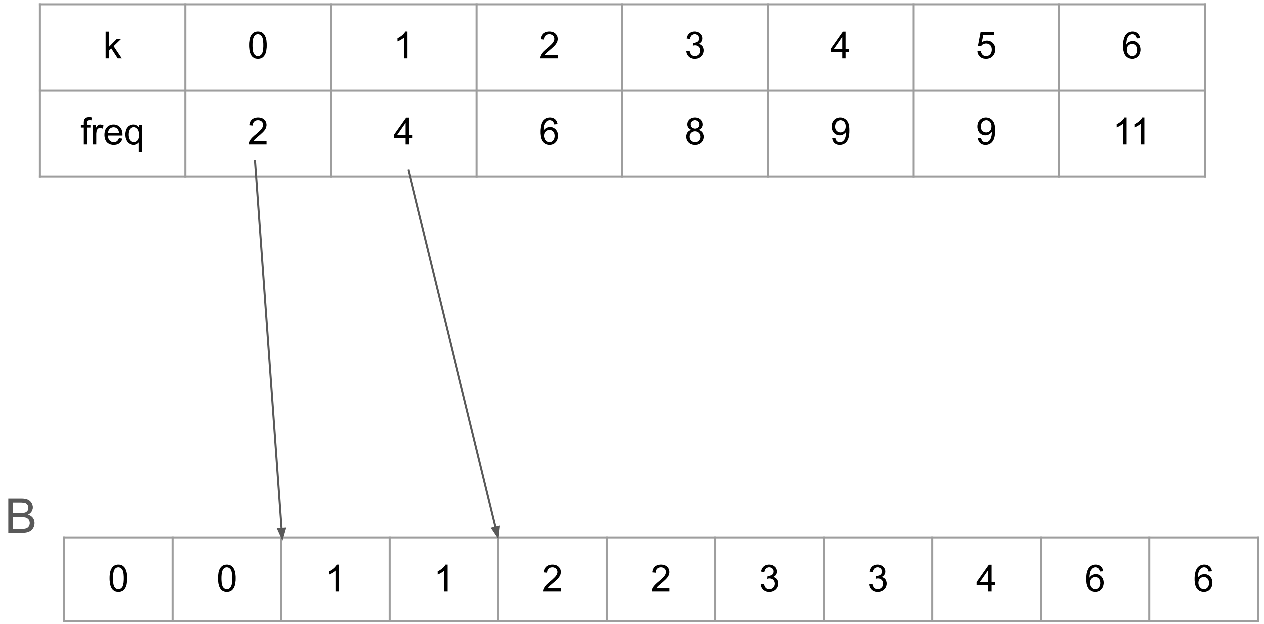
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

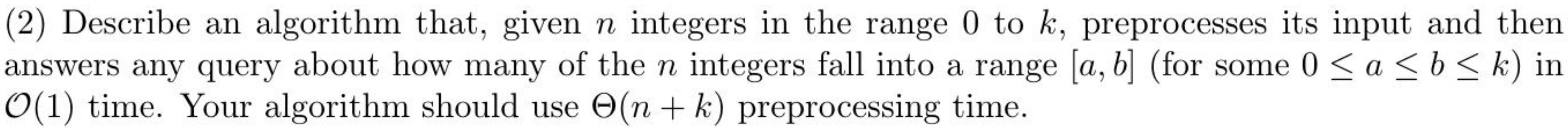
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

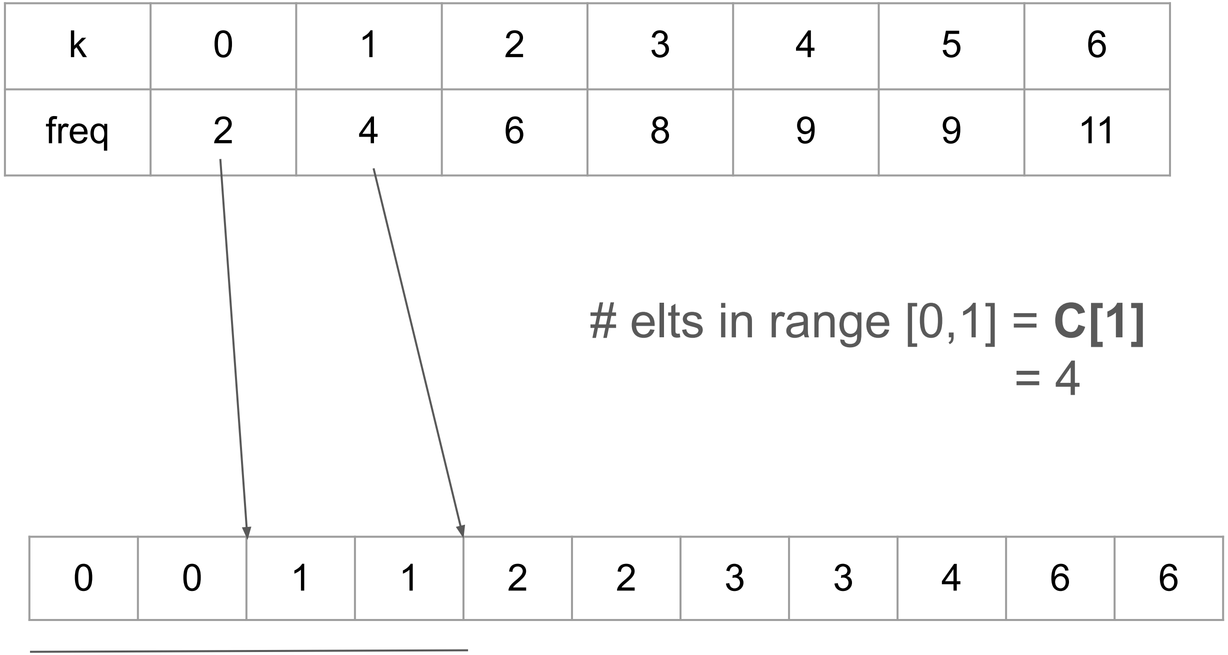
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \le a \le b \le k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

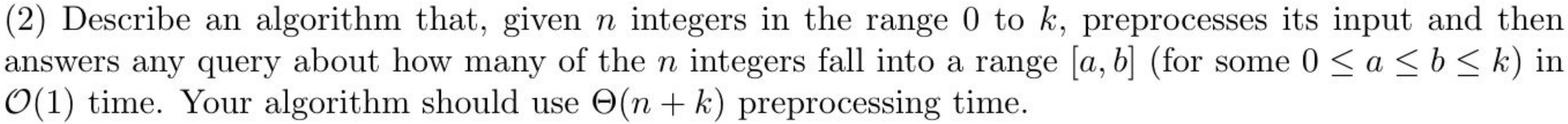


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [0,1] = **C[1]**
= 4

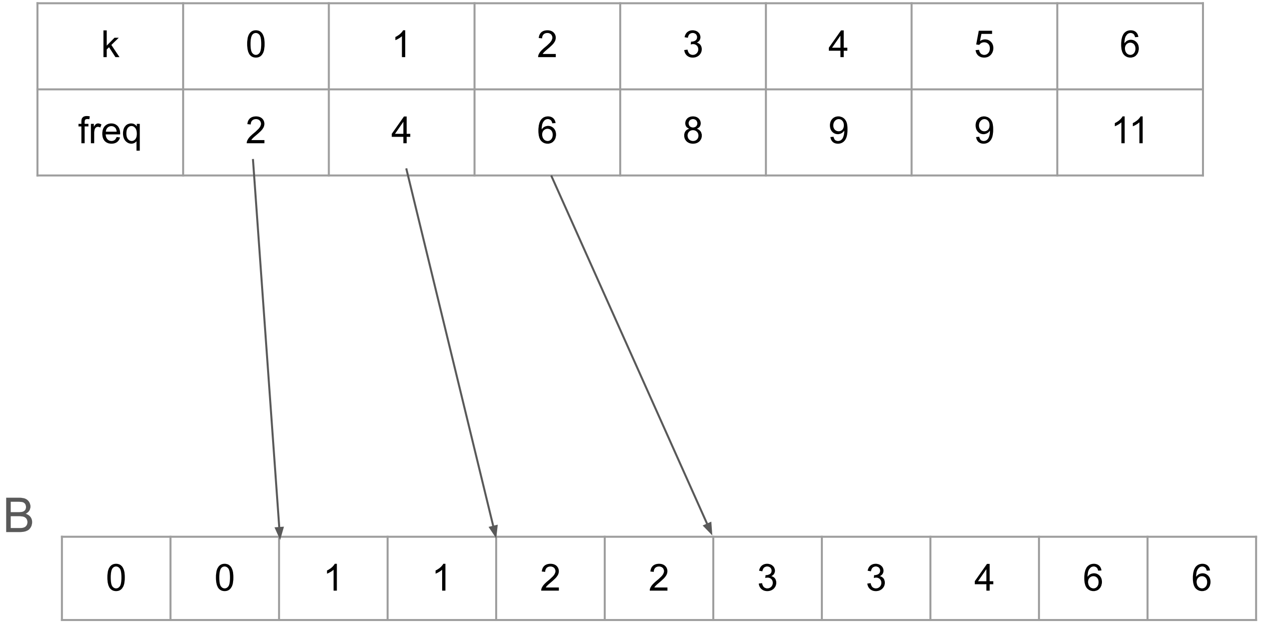
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

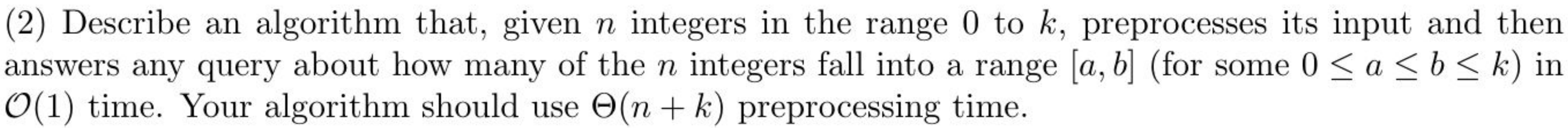
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

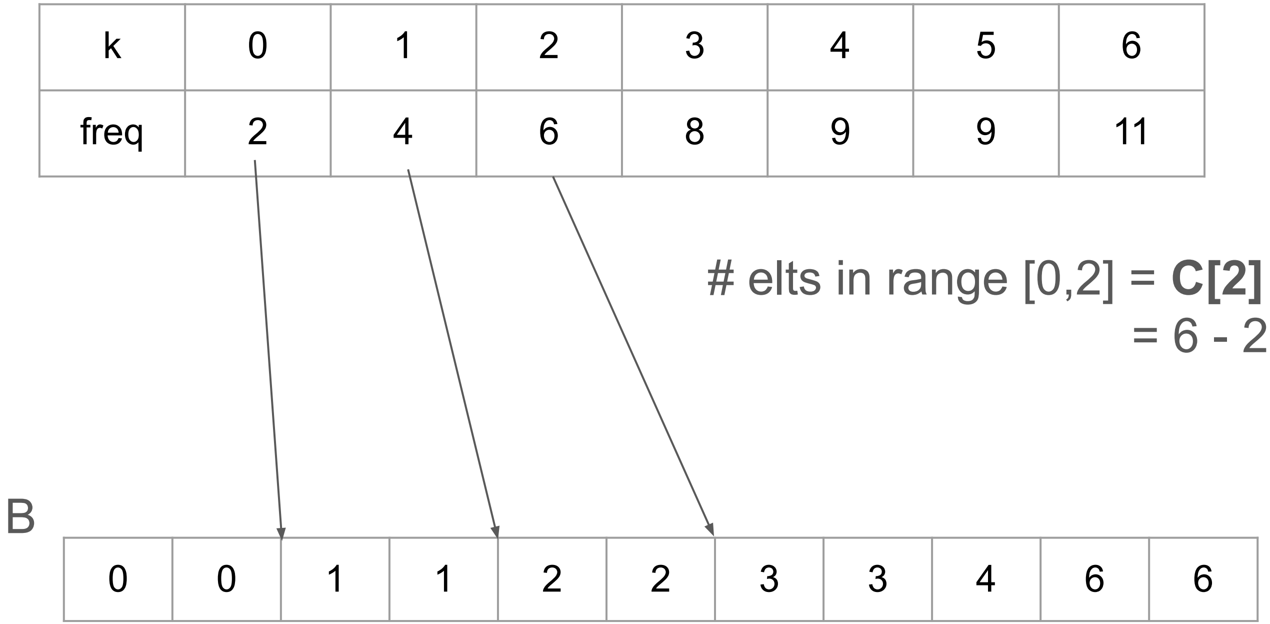
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

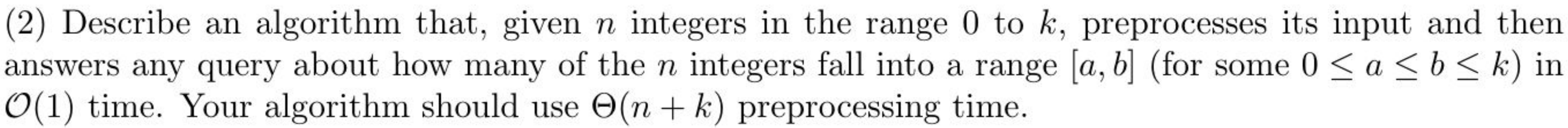


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

0	0	1	1	2	2	3	3	4	6	6

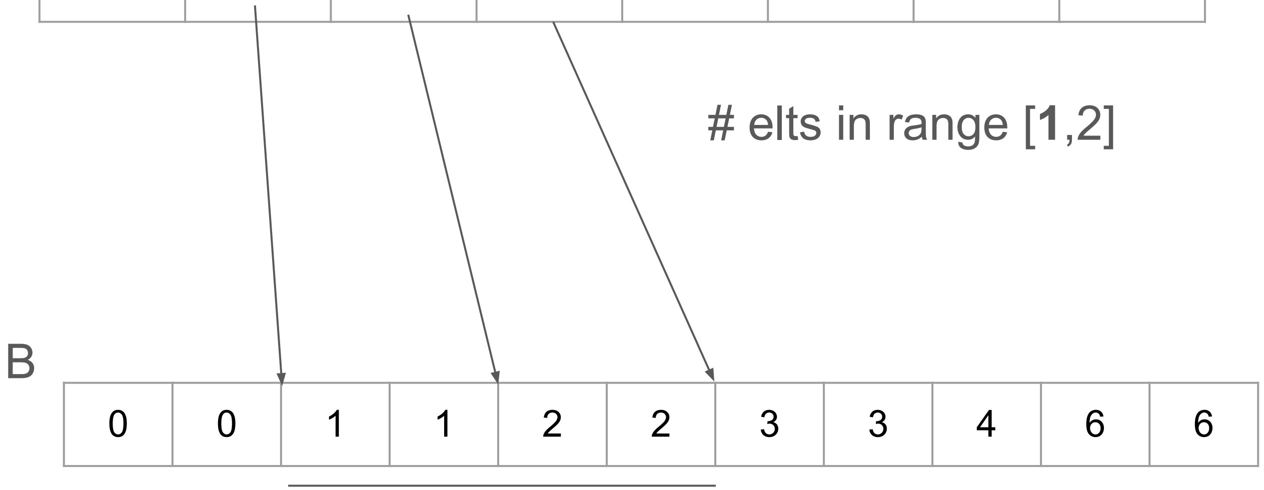
# elts in range [0,2] = **C[2]**
= 6 - 2 + 2 = 6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

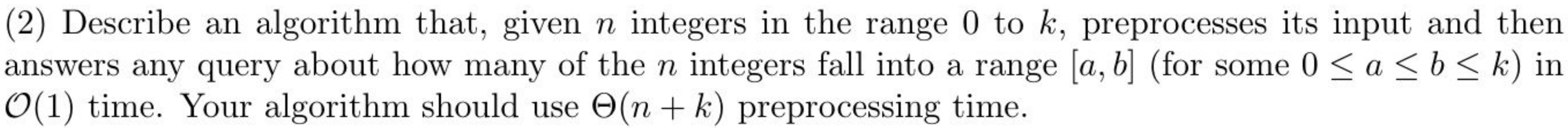
k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11



# elts in range [**1**,2]

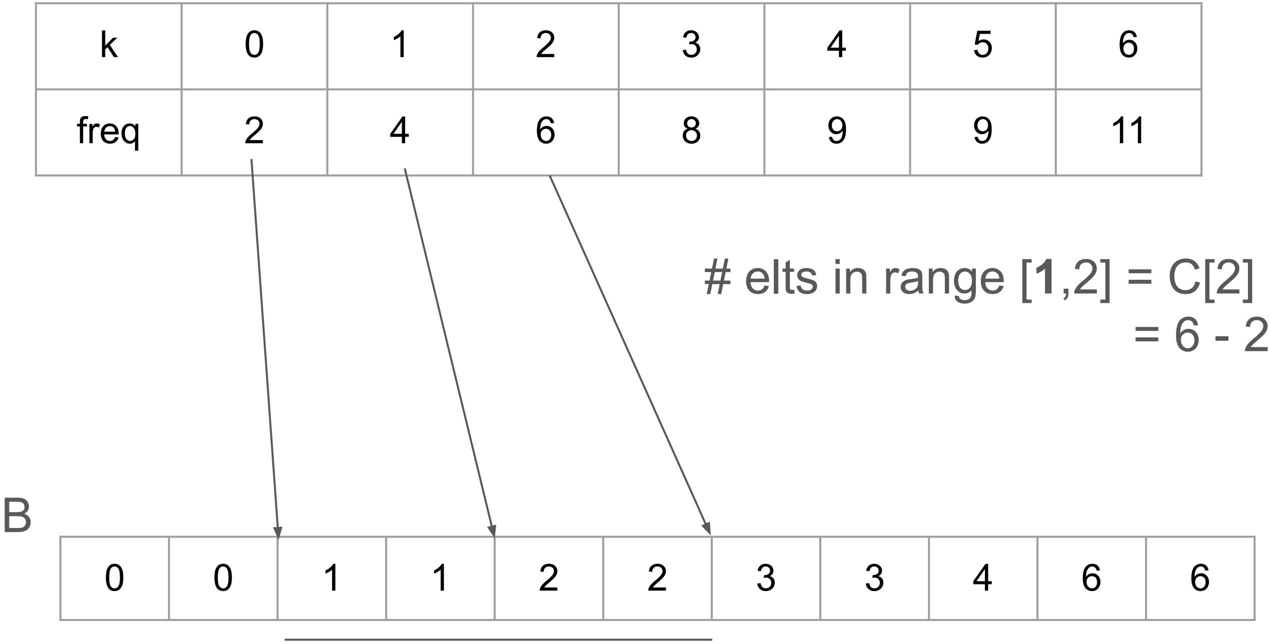
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

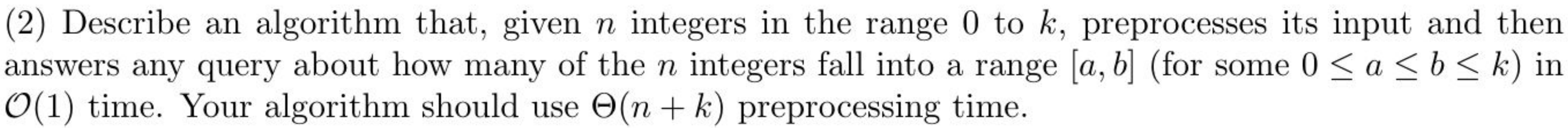


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

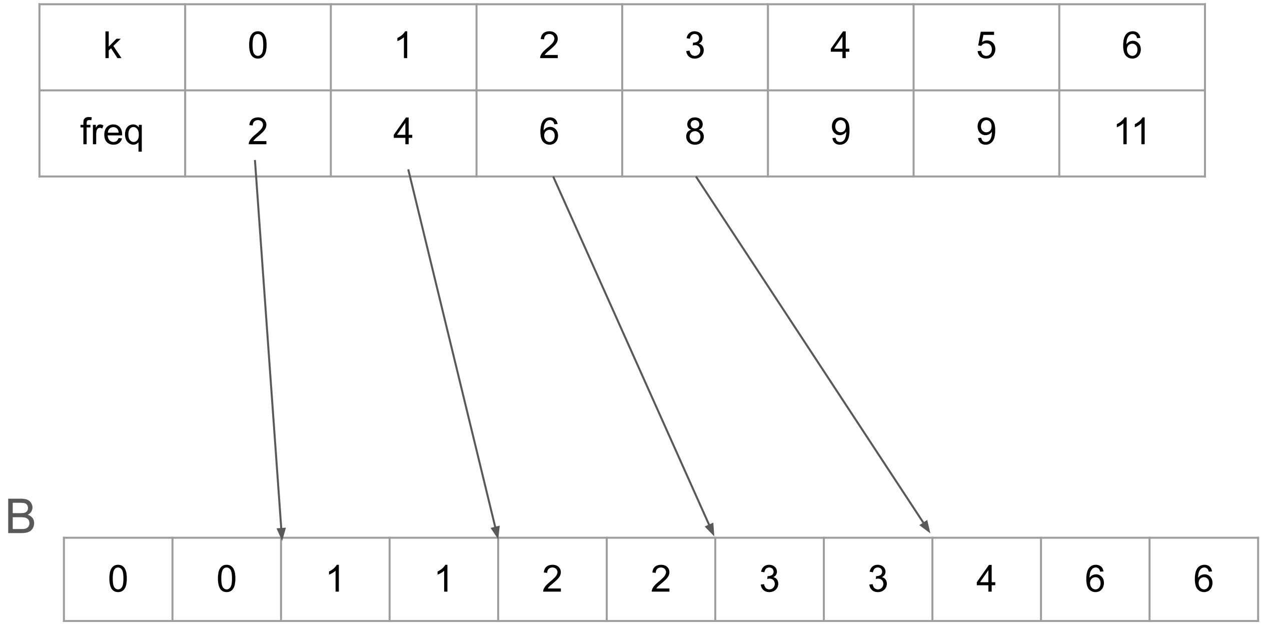
0	0	1	1	2	2	3	3	4	6	6

# elts in range [**1**,2] = C[2] - C[0]
= 6 - 2 = 4



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

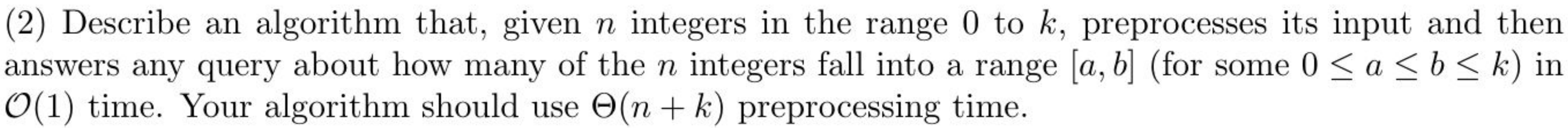
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

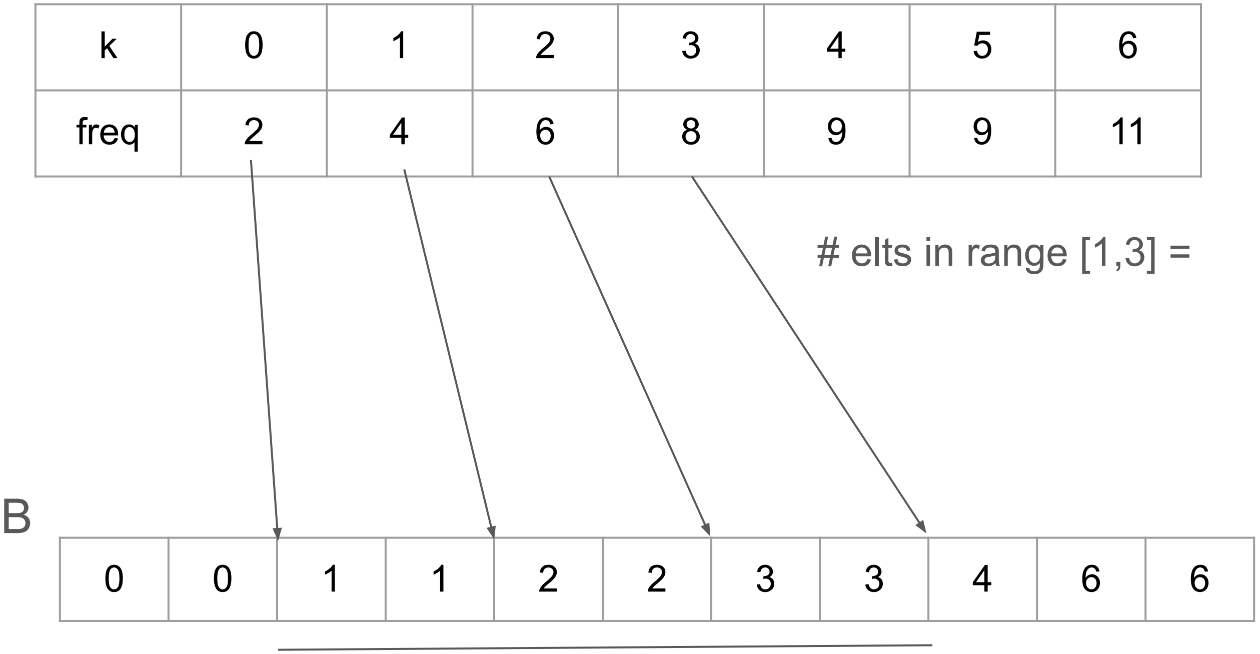
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

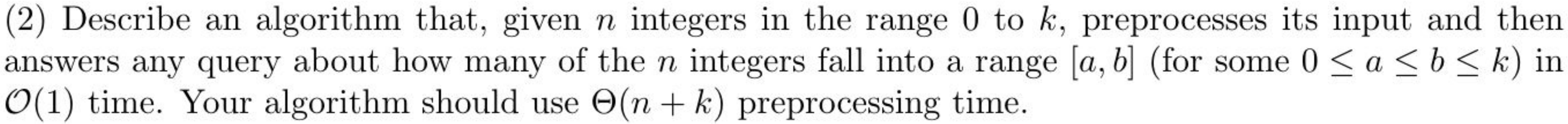


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [1,3] =

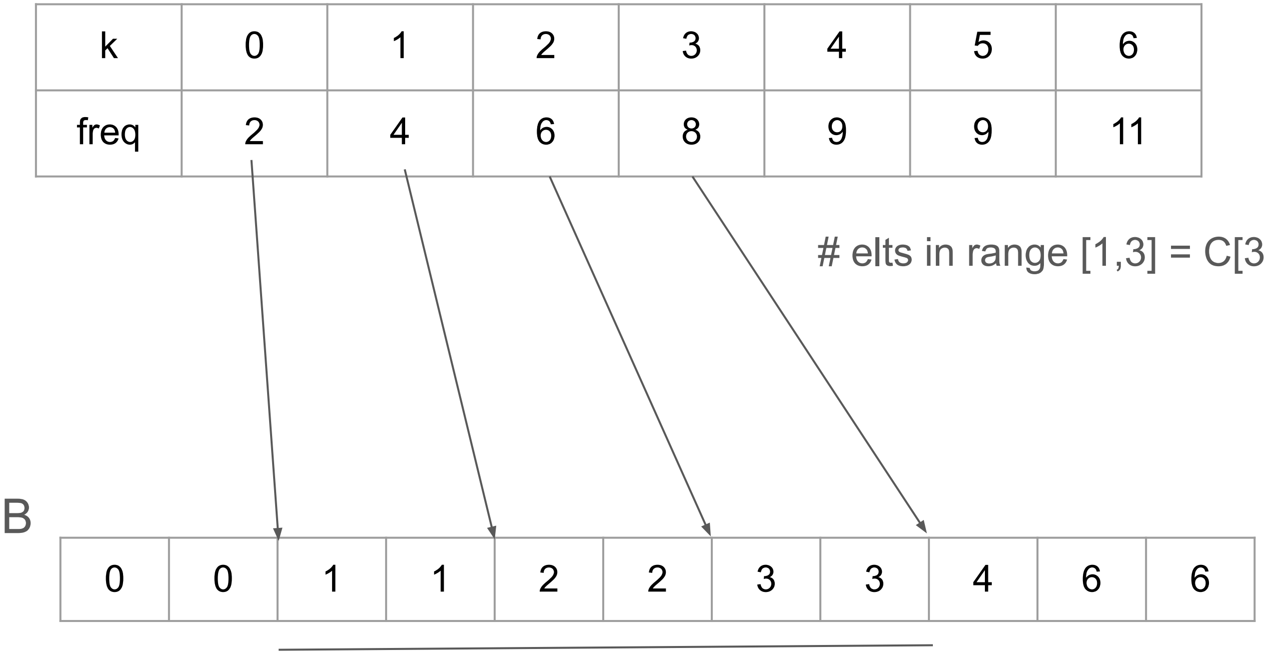
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

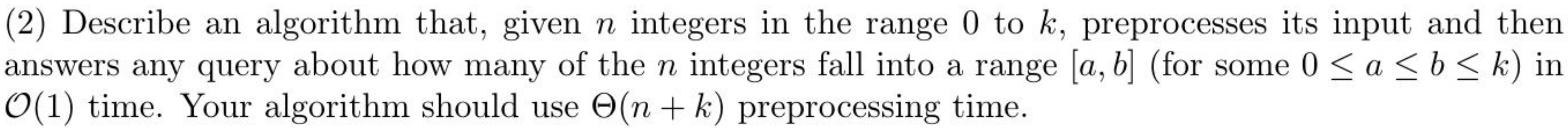


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [1,3] = C[3] - C[0]

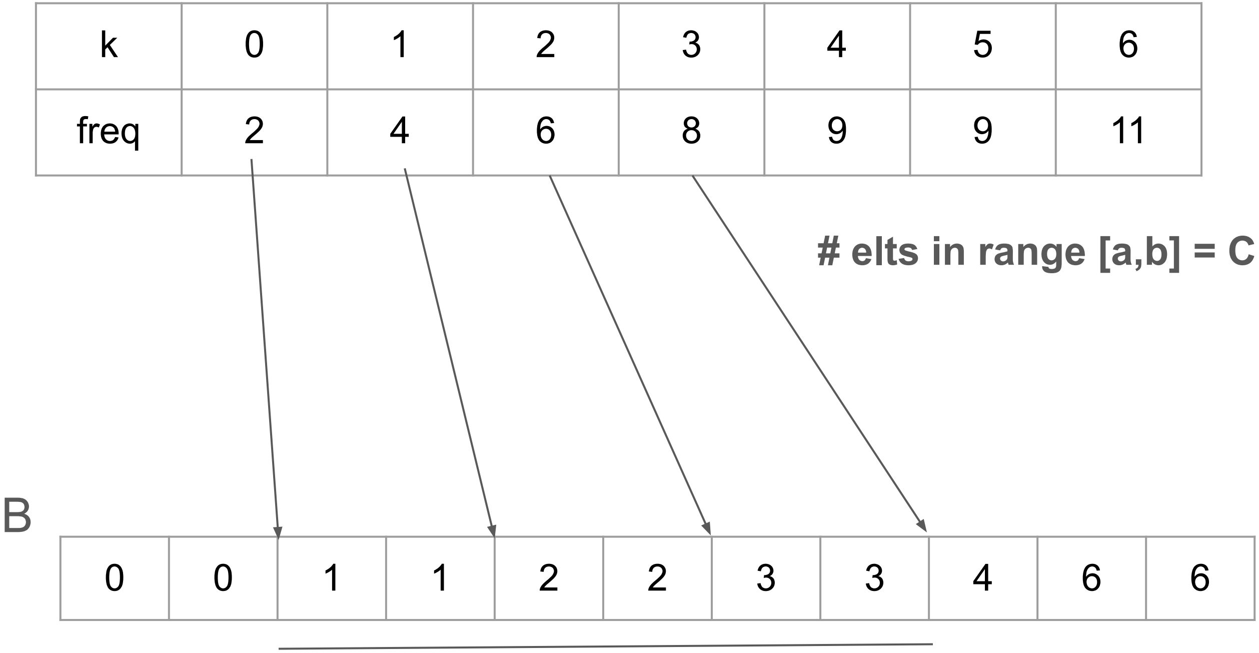
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**

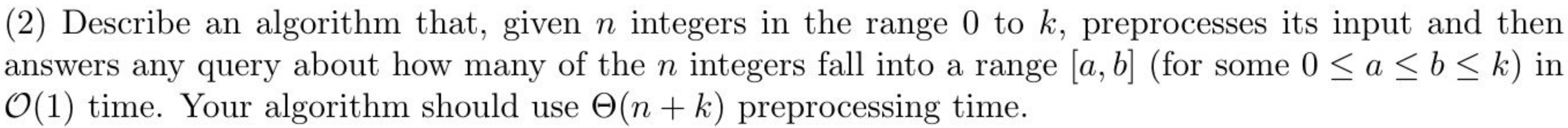


k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [a,b] = C[b] - C[a - 1]

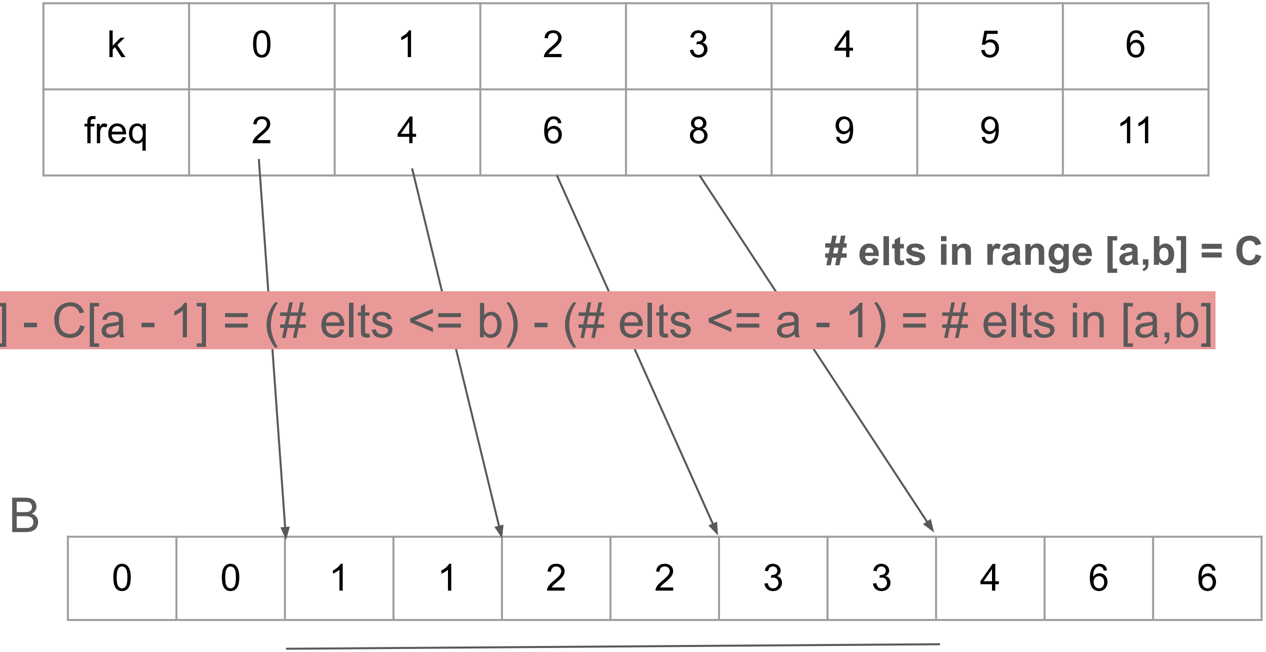
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

The counting array kept track of **C[i] = # elements less than or equal to i**



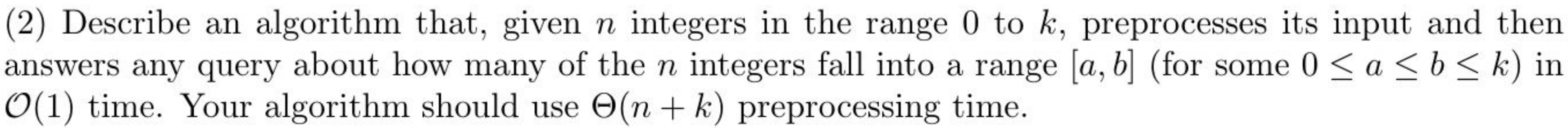
k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

# elts in range [a,b] = C[b] - C[a - 1]

C[b] - C[a - 1] = (# elts <= b) - (# elts <= a - 1) = # elts in [a,b]

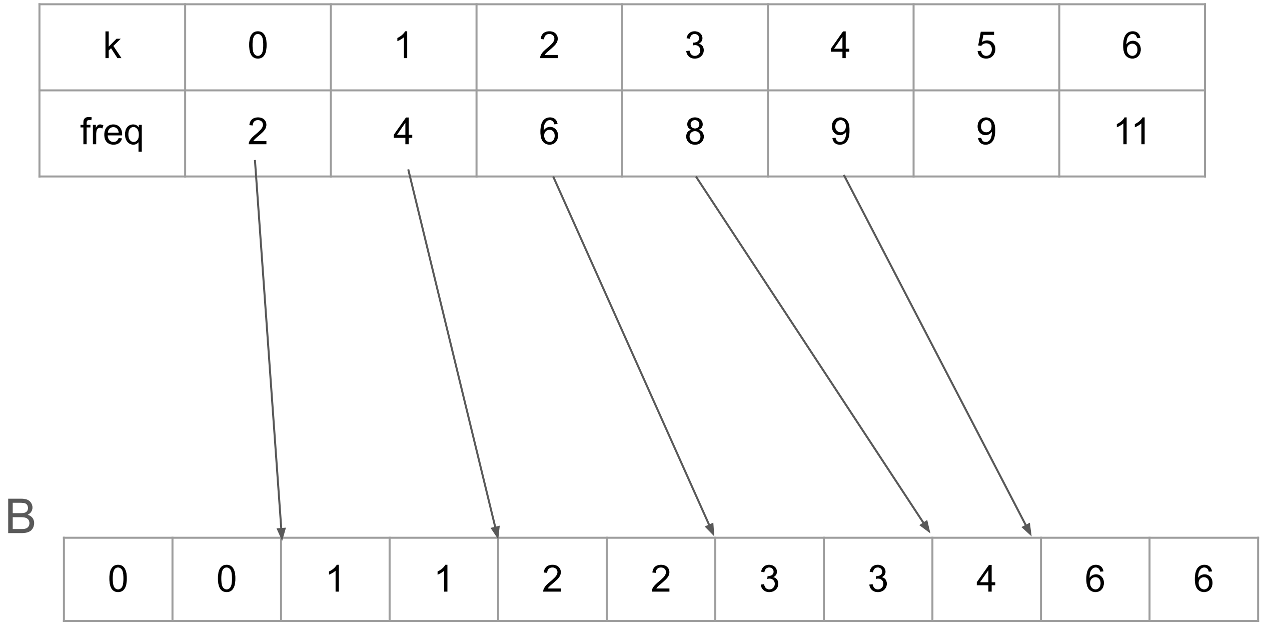
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

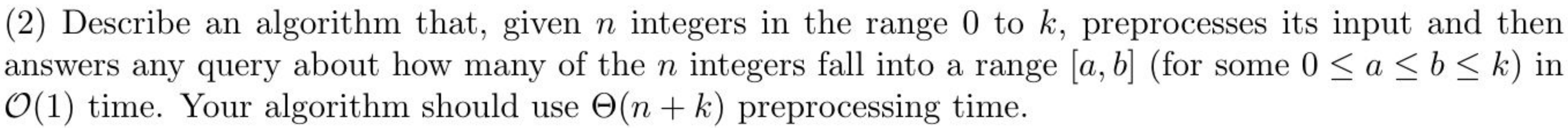
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

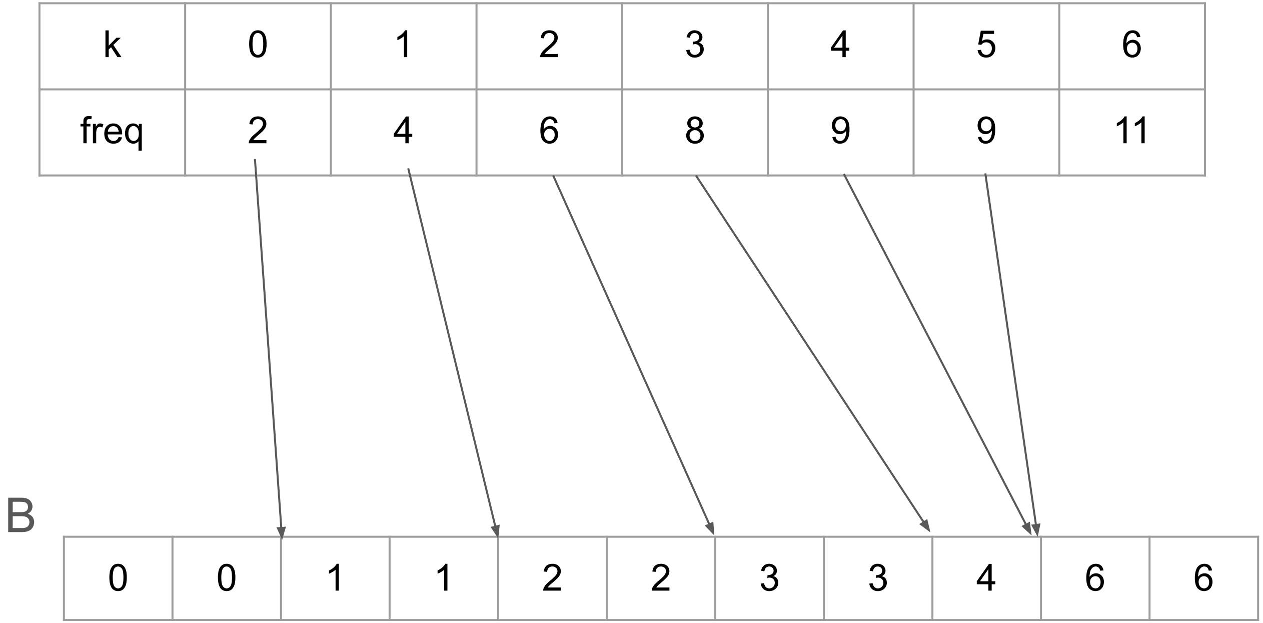
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \le a \le b \le k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

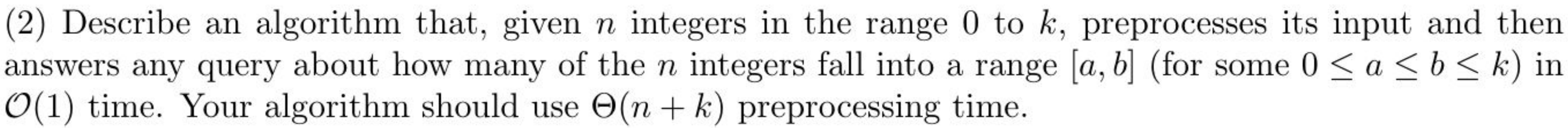
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

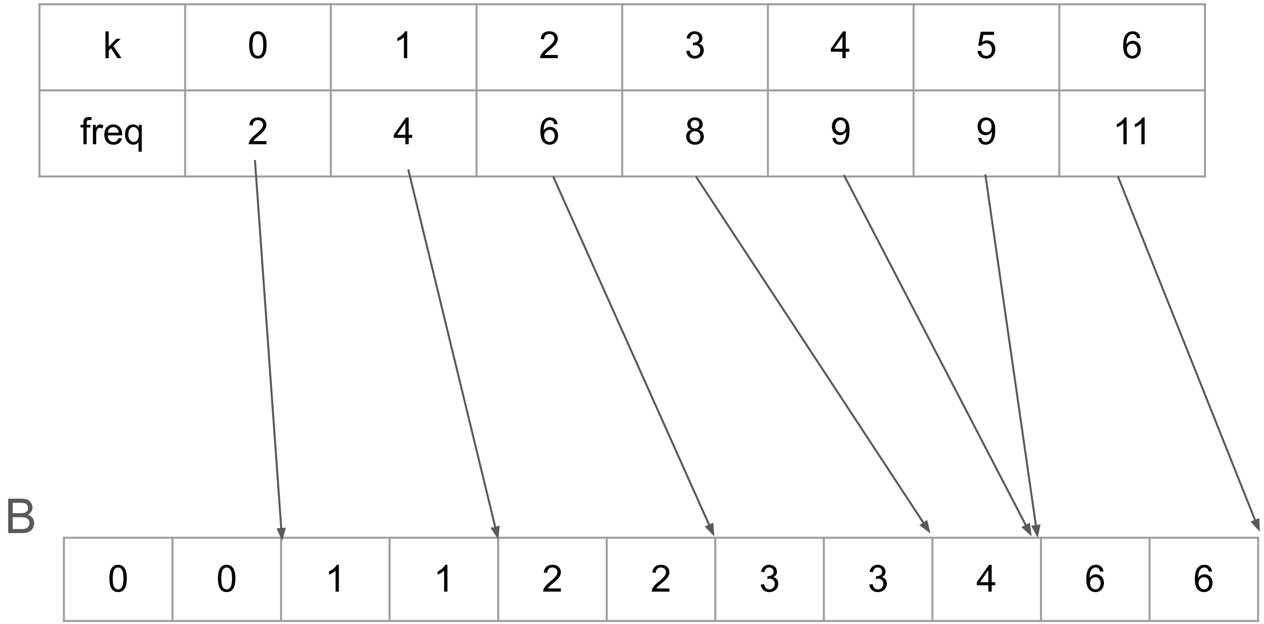
B

0	0	1	1	2	2	3	3	4	6	6



(2) Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (for some $0 \leq a \leq b \leq k$) in $\mathcal{O}(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time.

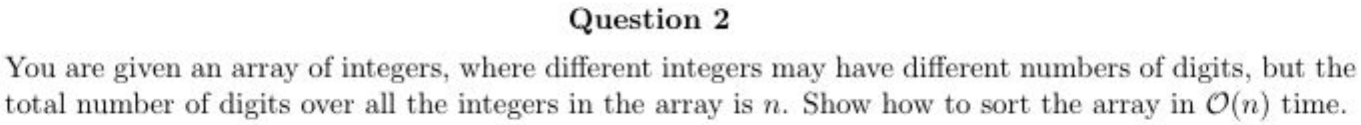
The counting array kept track of **C[i] = # elements less than or equal to i**



k	0	1	2	3	4	5	6
freq	2	4	6	8	9	9	11

B

0	0	1	1	2	2	3	3	4	6	6



Question 2

You are given an array of integers, where different integers may have different numbers of digits, but the total number of digits over all the integers in the array is n. Show how to sort the array in $\mathcal{O}(n)$ time.

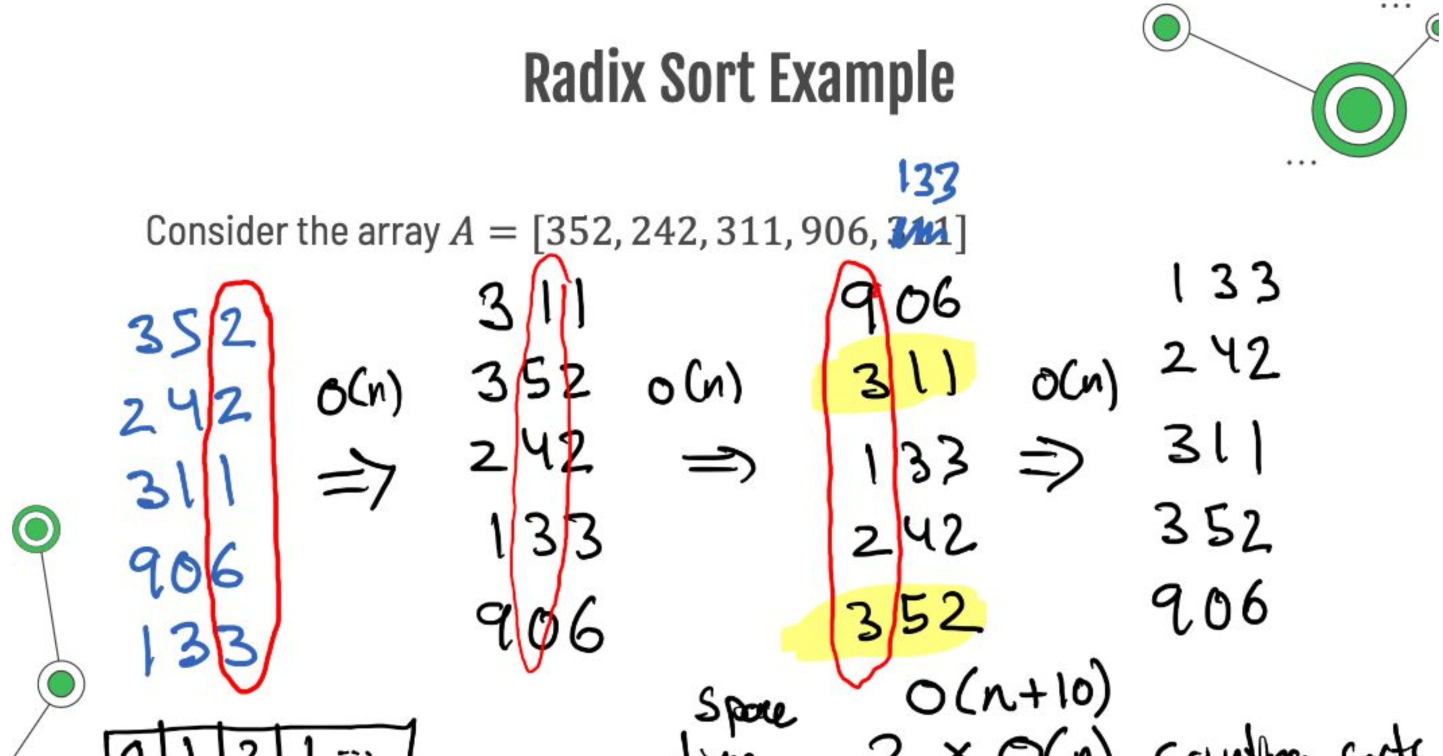
What is n here?

12	5	17	13	15	4	100	4

What are the two linear sorts we learn and how do they work

Why not just use radix sort?

12	5	17	13	15	4	100	4



Radix Sort Example

Consider the array $A = [352, 242, 311, 906, 133]$

352
242
311
906
133

O(n) ⇒

311
352
242
133
906

O(n) ⇒

906
311
133
242
352

O(n) ⇒

133
242
311
352
906

Space O(n+10)

time 3 × O(n) counting sorts

Sorting in O(n) time

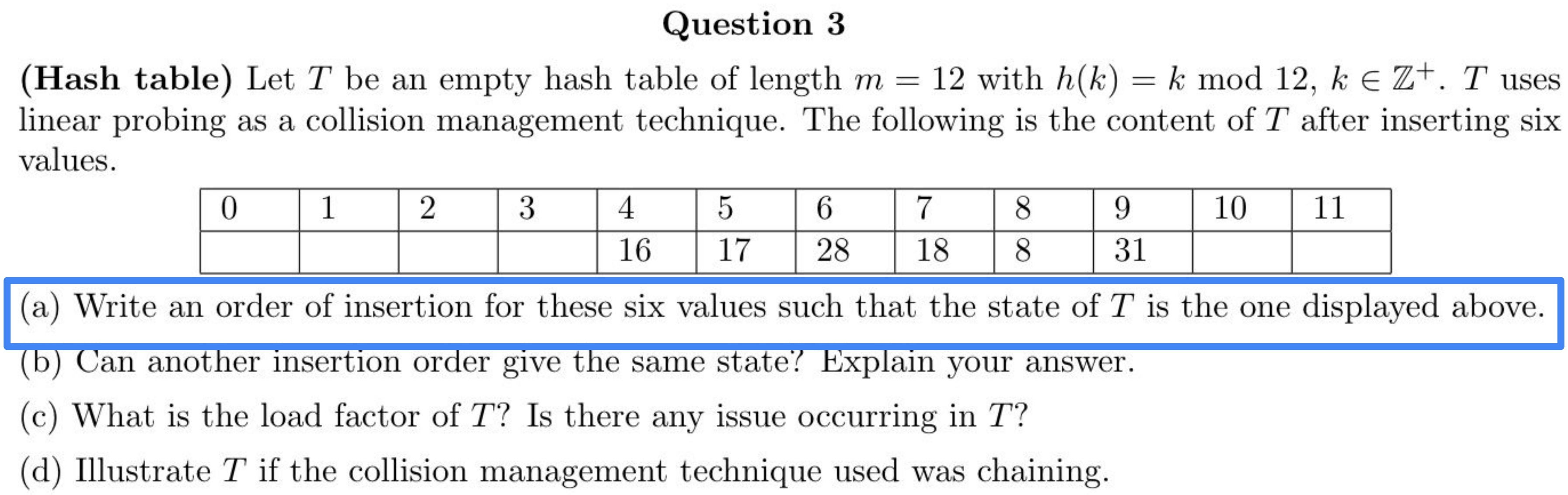
12	5	17	13	15	4	100	4

1. “Bucket” by lengths

5	4	4	12	17	13	15	100

2. Sort each length bucket

4	4	4	12	17	13	15	100



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

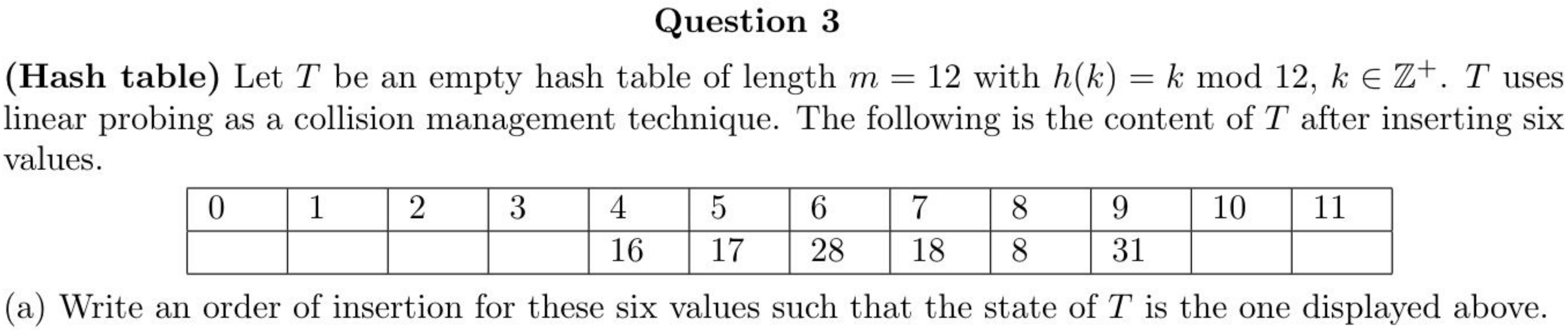
(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

(b) Can another insertion order give the same state? Explain your answer.

(c) What is the load factor of T? Is there any issue occurring in T?

(d) Illustrate T if the collision management technique used was chaining.

Linear Probing: If collision, check next box



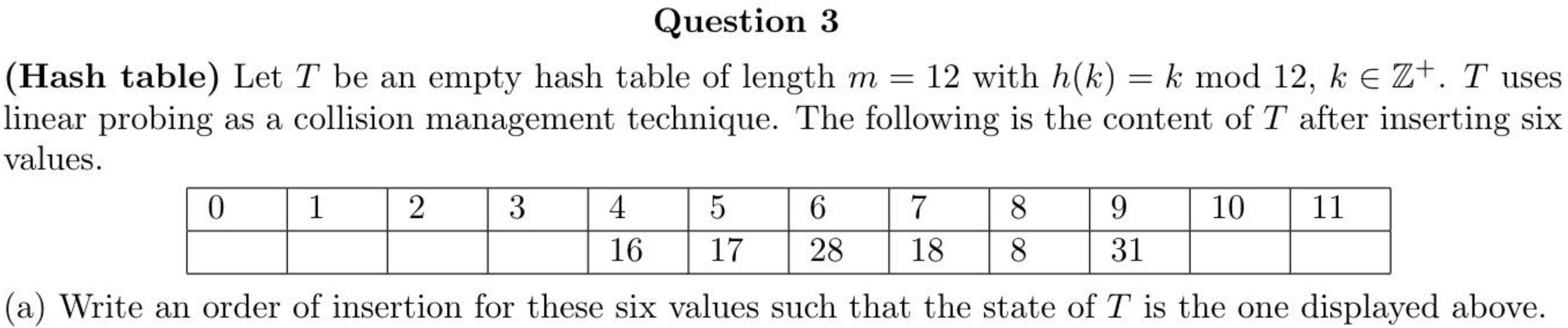
Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

k	h(k) = k mod 12
16	
17	
28	
28	
8	
31	



Question 3

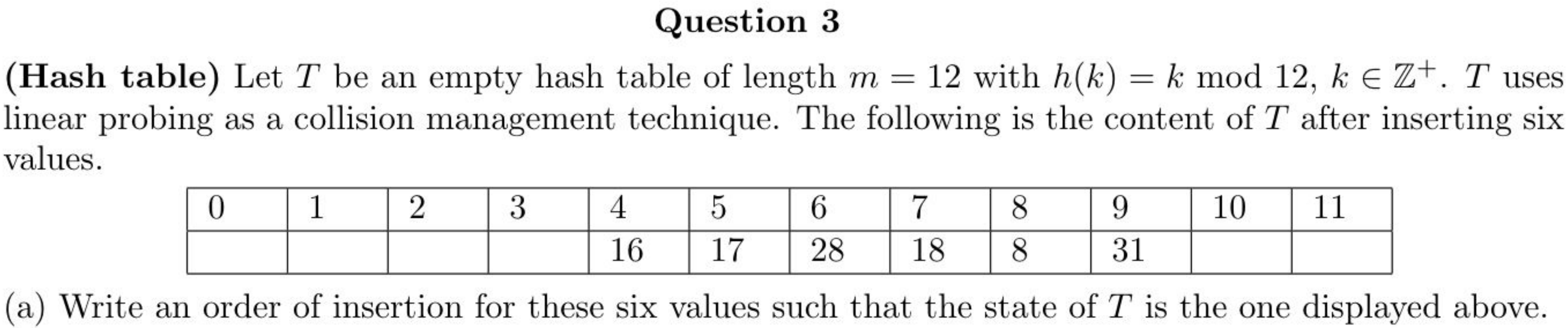
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

Which ones are in the right place?



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

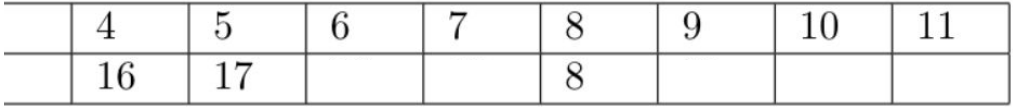
0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

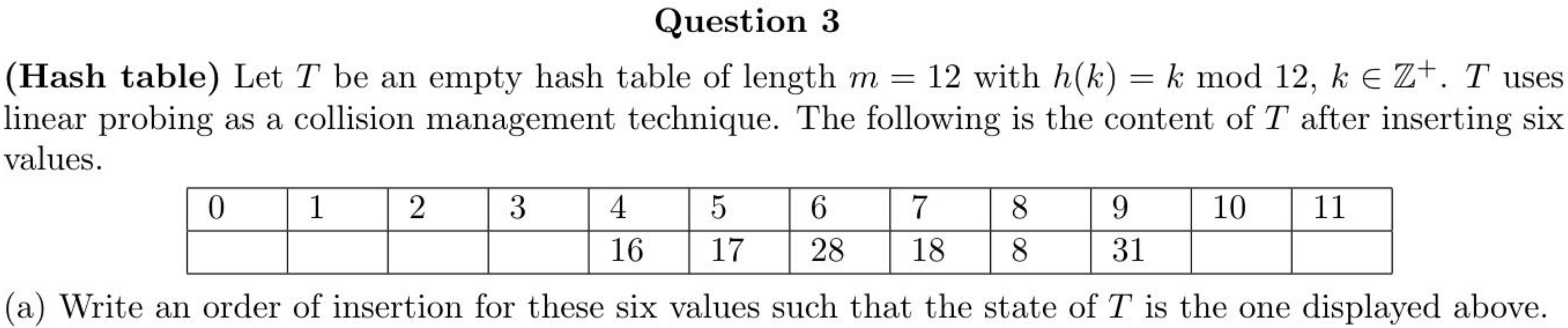
k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17			8			



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

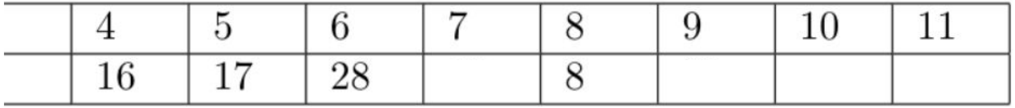
0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

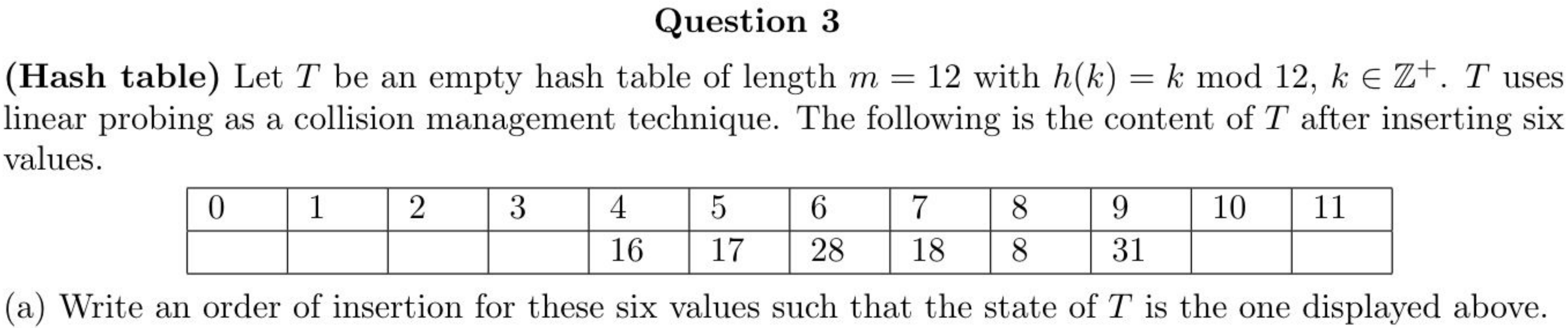
Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28		8			

Next, 28



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

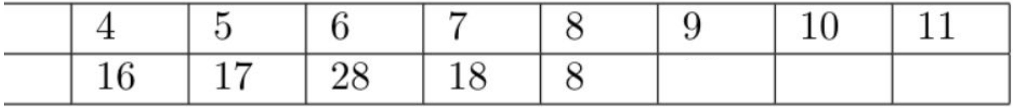
0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

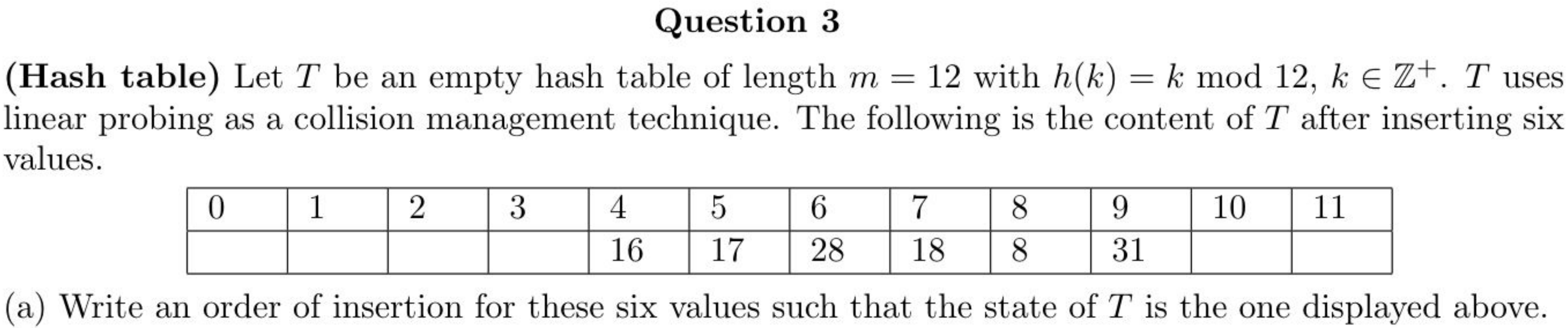
Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28	18	8			

Next, 28, 18



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

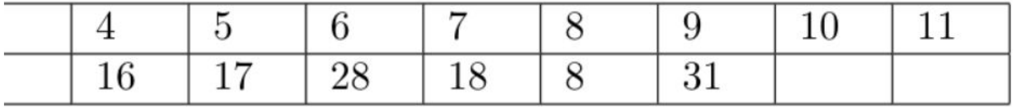
0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28	18	8	31		

Next, 28, 18, 31

Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

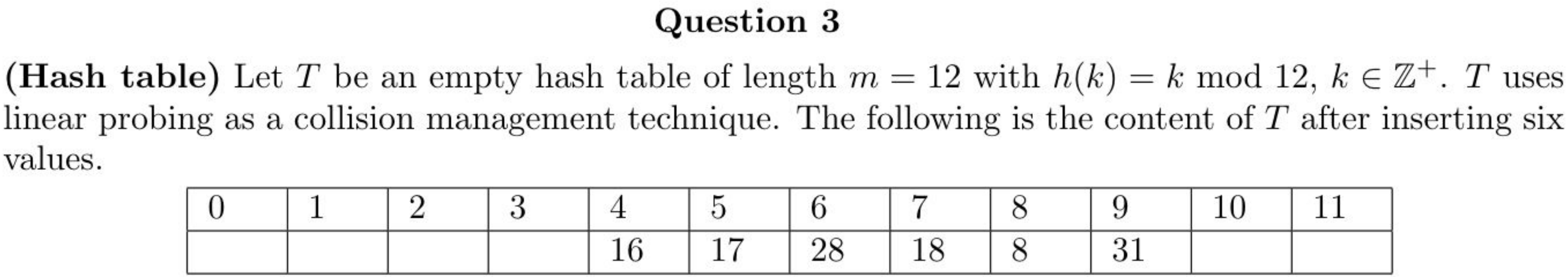
0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

(b) Can another insertion order give the same state? Explain your answer.

(c) What is the load factor of T? Is there any issue occurring in T?

(d) Illustrate T if the collision management technique used was chaining.



Question 3

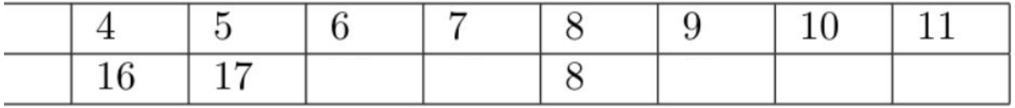
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

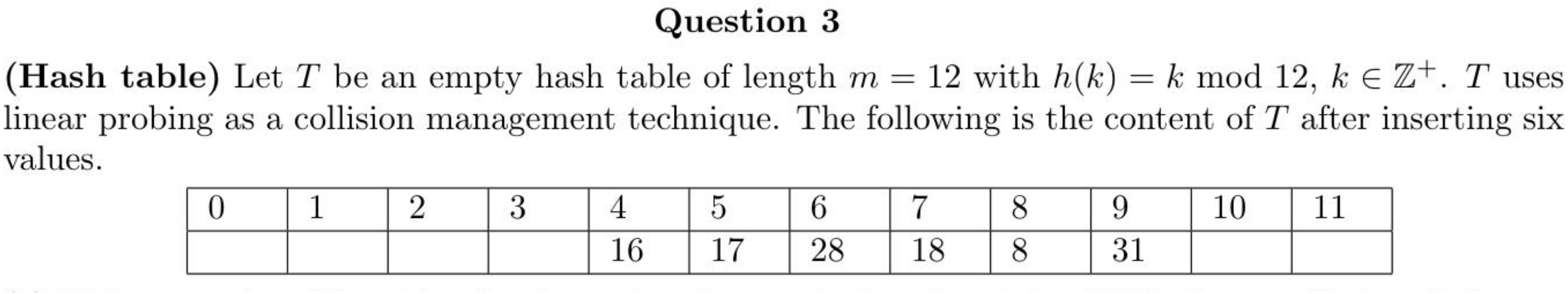
k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

Which ones are in the right place?

Insert 16,17,8 first



4	5	6	7	8	9	10	11
16	17			8			



Question 3

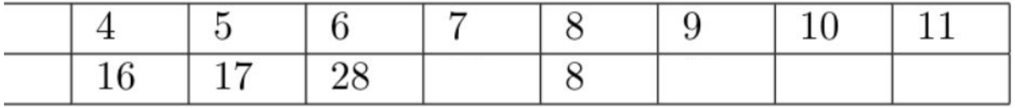
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

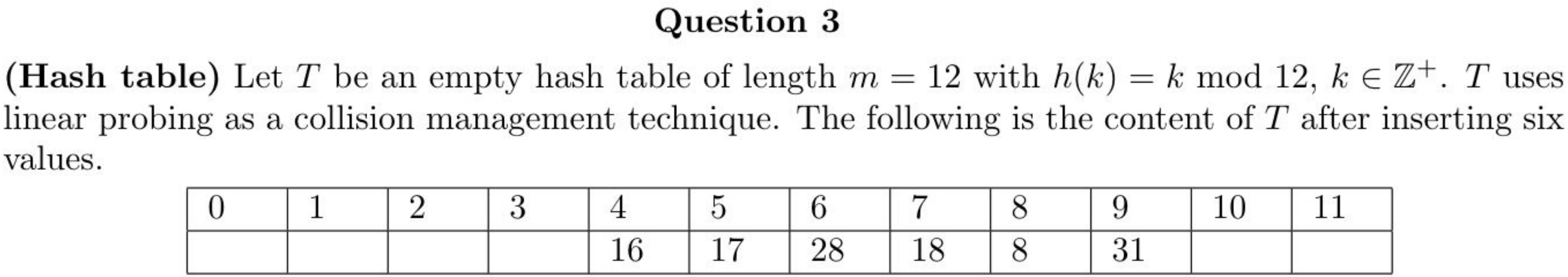
Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28		8			

Next, 28



Question 3

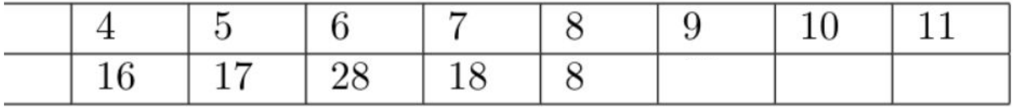
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

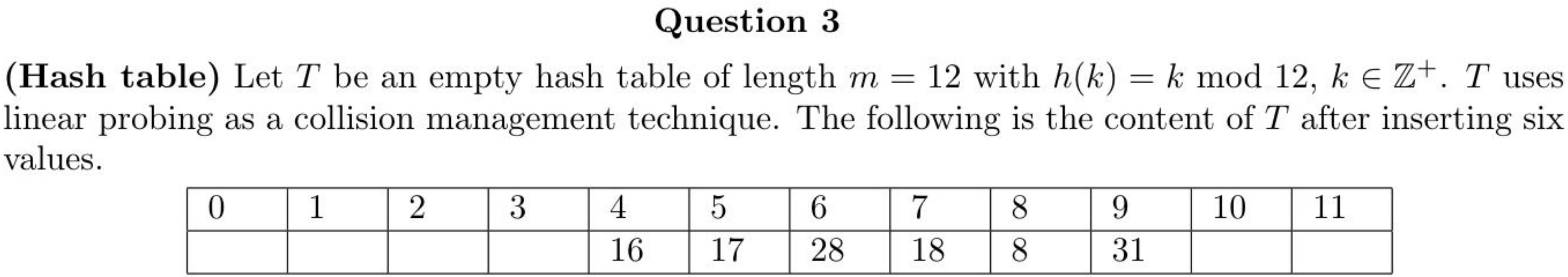
Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28	18	8			

Next, 28, 18



Question 3

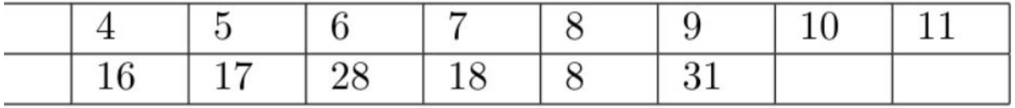
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

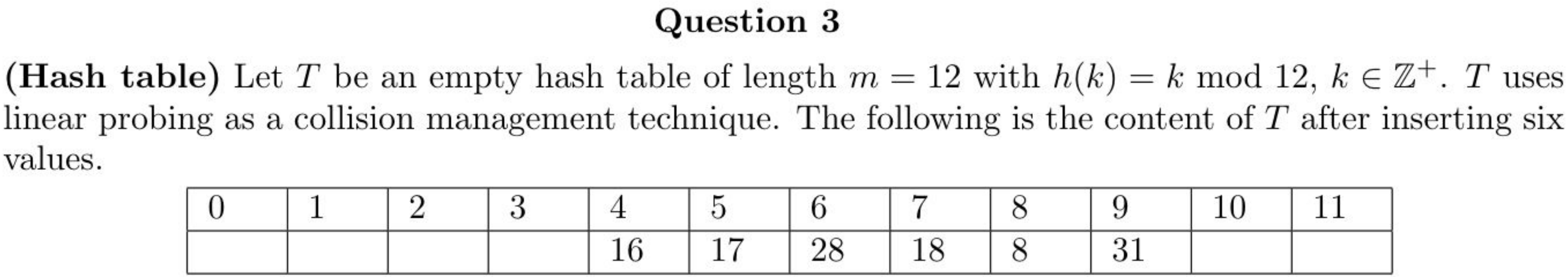
Which ones are in the right place?

Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28	18	8	31		

Next, 28, 18, 31



Question 3

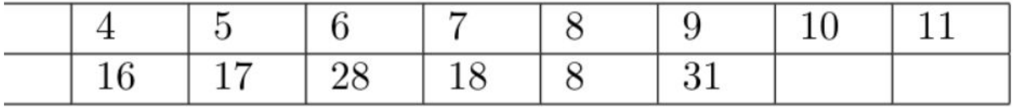
(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

Which ones are in the right place?

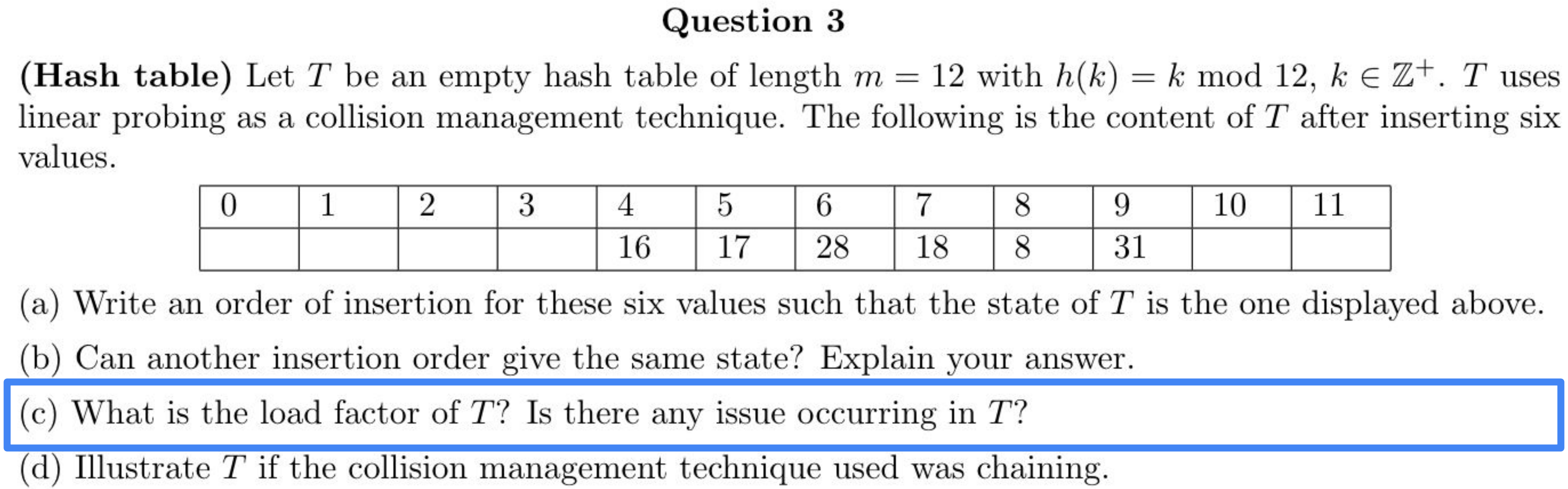
Insert 16,17,8 first



	4	5	6	7	8	9	10	11
	16	17	28	18	8	31		

Next, 28, 18, 31

I can enter 16,17,8 in any order



Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

(b) Can another insertion order give the same state? Explain your answer.

(c) What is the load factor of T? Is there any issue occurring in T?

(d) Illustrate T if the collision management technique used was chaining.

Load factor =

Question 3

(Hash table) Let T be an empty hash table of length $m = 12$ with $h(k) = k \bmod 12$, $k \in \mathbb{Z}^+$. T uses linear probing as a collision management technique. The following is the content of T after inserting six values.

0	1	2	3	4	5	6	7	8	9	10	11
				16	17	28	18	8	31		

(a) Write an order of insertion for these six values such that the state of T is the one displayed above.

(b) Can another insertion order give the same state? Explain your answer.

(c) What is the load factor of T? Is there any issue occurring in T?

(d) Illustrate T if the collision management technique used was chaining.

(d) Illustrate T if the collision management technique used was chaining.

Insertion order: 16, 17, 28, 18, 8, 31

k	h(k) = k mod 12
16	4
17	5
28	4
18	6
8	8
31	7

	4	5	6	7	8	9	10	11