Ayudantía 2

Merge
MergeSort
QuickSort

Merge

Algoritmo que mezcla 2 conjuntos ordenados en un tercero de forma ordenada, se puede resumir en los siguientes pasos:

1. Se toma el primer elemento de A y el primer elementos de B
2. Se comparan y el menor se ingresa al final de C
3. Si quedan elementos en A y B, ir a 1
4. Anexar a C la lista que aún tenga elementos

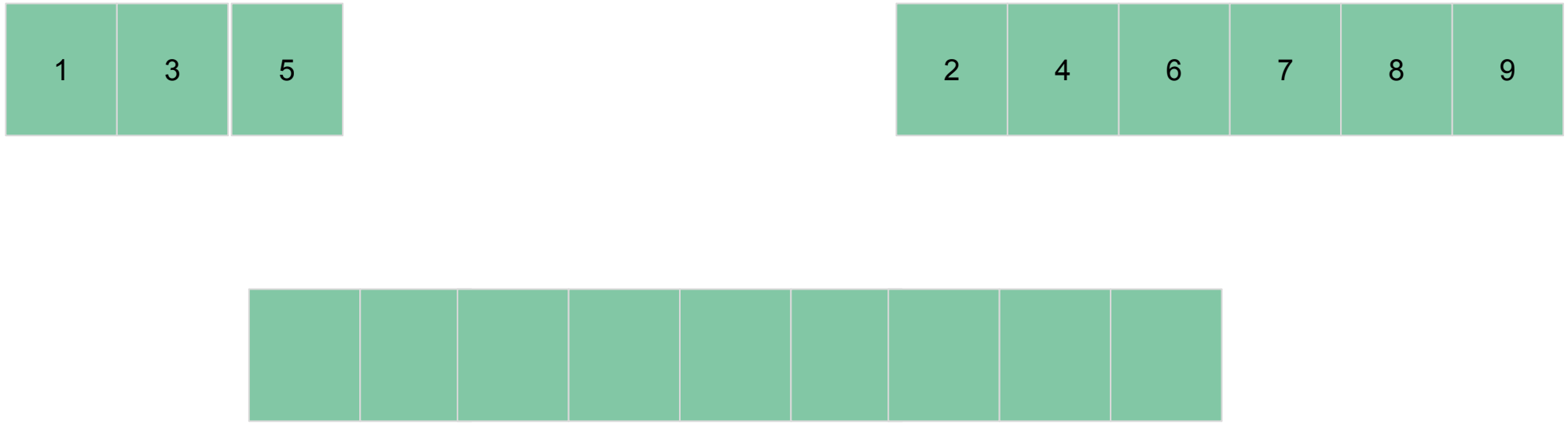
Merge

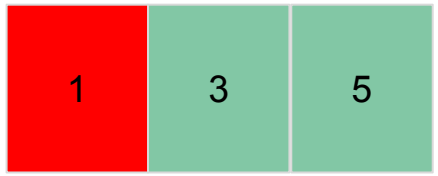
En pseudocódigo

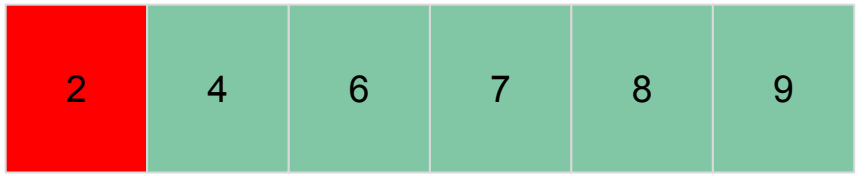
```
merge(A, B):
    C = Arreglo vacío
    while A y B no estén vacíos:
        a = primer elemento de A
        b = primer elemento de B
        if a <= b:
            C.append(a)
            Eliminamos a de A
        else if a > b:
            C.append(b)
            Eliminamos b de B
    if A está vacío:
        C.append(B)
    else if B.está vacío:
        C.append(A)
    return C
```

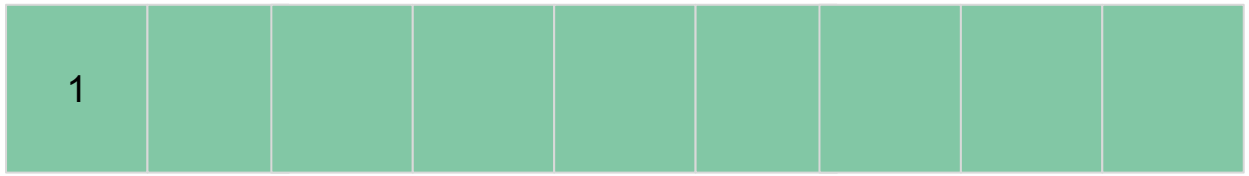
Veamos un ejemplo

Se tiene 2 arreglos, A = [1, 3, 5] y B= [2, 4, 6, 7, 8, 9], el resultado de aplicar un merge se guarda en otro arreglo C

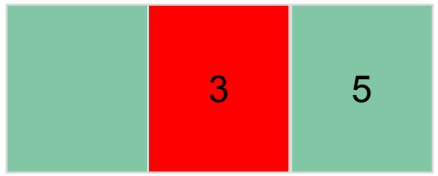


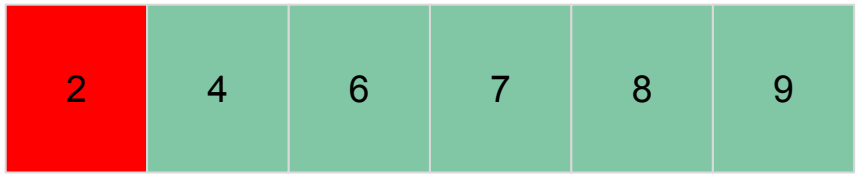


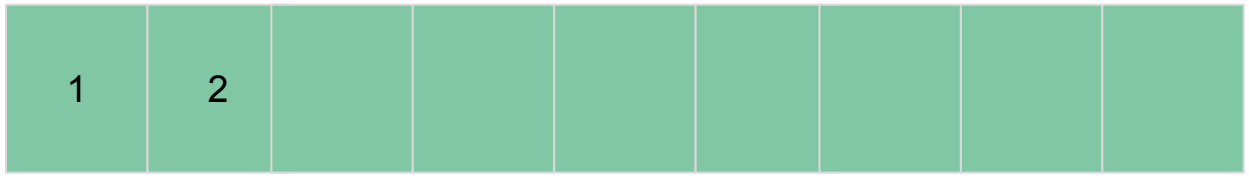




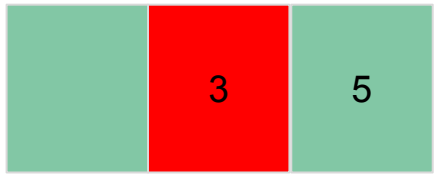
- Se selecciona el menor elemento entre A y B
- Se ingresa a C

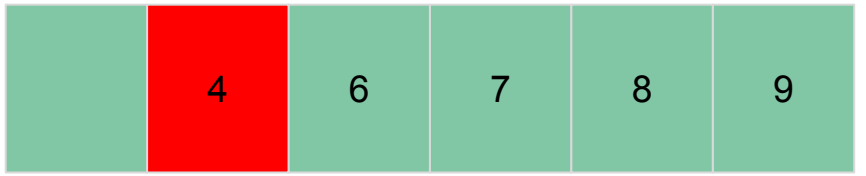


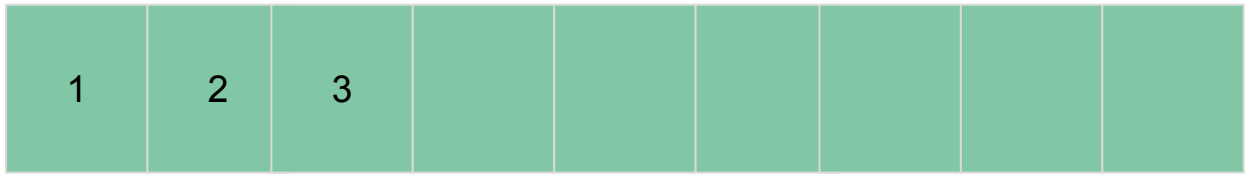




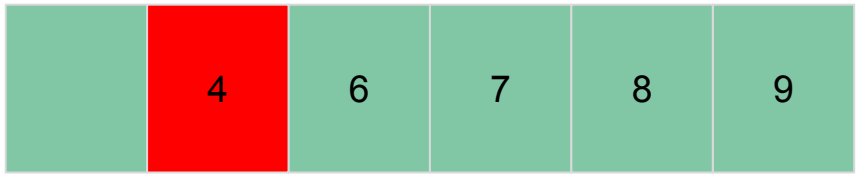
- Se selecciona el menor elemento entre A y B
- Se ingresa a C

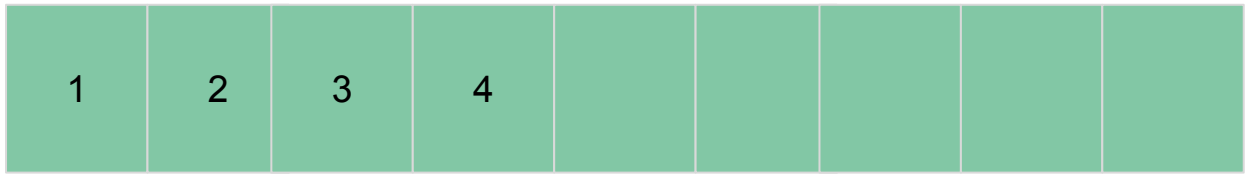




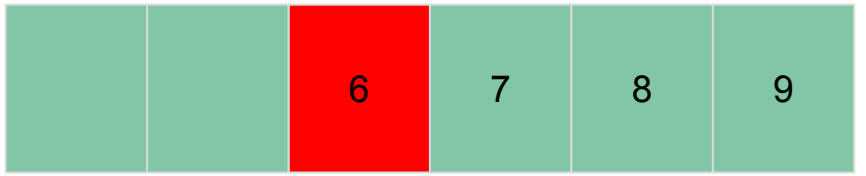


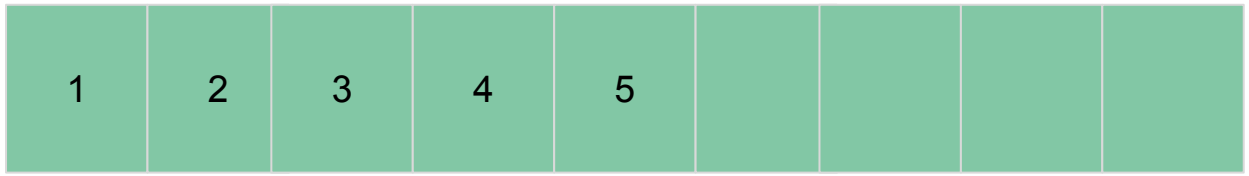
- Se selecciona el menor elemento entre A y B
- Se ingresa a C



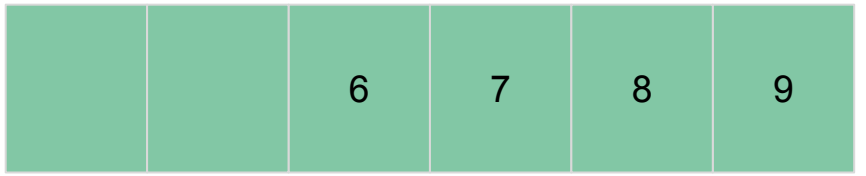


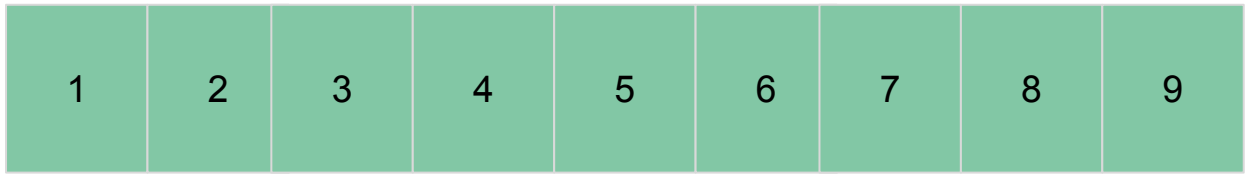
- Se selecciona el menor elemento entre A y B
- Se ingresa a C





- Se selecciona el menor elemento entre A y B
- Se ingresa a C





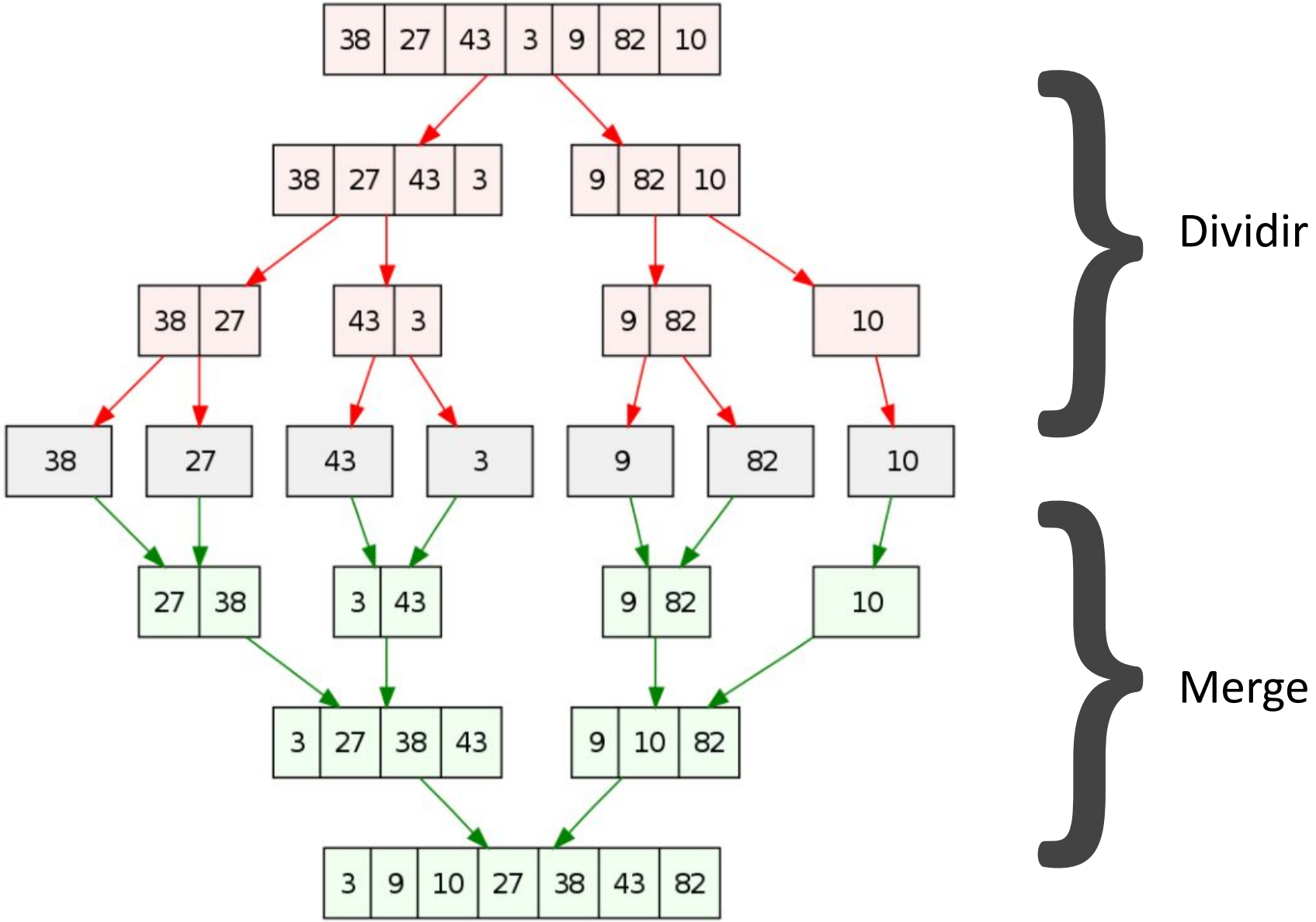
- Como A quedó vacío se ingresa el resto de B

MergeSort

Algoritmo se basa en el paradigma de dividir para reinar

- **Divide:** Divide la secuencia de n-elementos en 2 subsecuencias
- **Conquista:** Ordena las 2 subsecuencias recursivamente con merge sort
- **Combina**: Realiza un Merge a las 2 subsecuencias ordenadas

MergeSort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82
Dividir
Merge

Su pseudocódico

```
mergeSort(A, i, f):
    if f - i > 1:
        m = round((f - i)/2)
        B = mergeSort(A, i, m)
        C = mergeSort(A, m, f)
        A[i:f] = merge(B, C)
    return A[i:f]
```

- A un arreglo
- i un índice indicando el inicio
- f un índice indicando el término

Demostremos la correctitud del algoritmo

Caso base: Para un arreglo de largo 1 este ya se encuentra ordenado y es retornado, por tanto mergeSort es correcto.

HI: Asumimos que MergeSort ordena correctamente un arreglo de largo k.

PD: MergeSort ordena un arreglo de largo k+1.

Recordando el algoritmo:

```
mergeSort(A, i, f):
    if f - i > 1:
        m = round((f - i)/2)
        B = mergeSort(A, i, m)
        C = mergeSort(A, m, f)
        A[i:f] = merge(B, C)
    return A[i:f]
```

TI: Para un arreglo de largo k+1, en la primera iteración se divide este arreglo en dos arreglos de largo (k+1)/2 a los que se les aplicará mergeSort. Como cada subarreglo es de un largo menor que k, por HI mergeSort ordena los arreglos.

Cumpliendo su objetivo

¿Tiene fin este algoritmo?

Problemas:

Implementa en C un merge de 2 arreglos ordenados ascendentemente y retorna un arreglo en forma decreciente

Implementa en C un merge sort para ordenar el arreglo [3; 41; 52; 26; 38; 57; 9; 49]

QuickSort

Pseudocódigo

```
quicksort(A, i, f):
    if i ≤ f:
        p ← partition(A, i, f)
        quicksort(A, i, p − 1)
        quicksort(A, p + 1, f)
```

```
partition(A, i, f):
    x ← un indice aleatorio en [i, f],    p ← A[x]
    A[x] ⇄ A[f]
    j ← i
    for k ∈ [i, f − 1]:
        if A[k] < p:
            A[j] ⇄ A[k]
            j ← j + 1
    A[j] ⇄ A[f]
    return j
```


UNO


1
1
2
2
3
3
4
4
5
5
6
6


3
3
6
6
1
1
4
4
2
2
5
5

i x f

A[x]

A[f]

i

x

f

p

A[f]

i

x

f

A[f]

p

i

x

f

p

i

f

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

p

i

f

j

k

A[j]

A[k]

A[f]

i

f

j

k

A[j]

A[k]

A[f]

i

f

j

k

A[j]

p

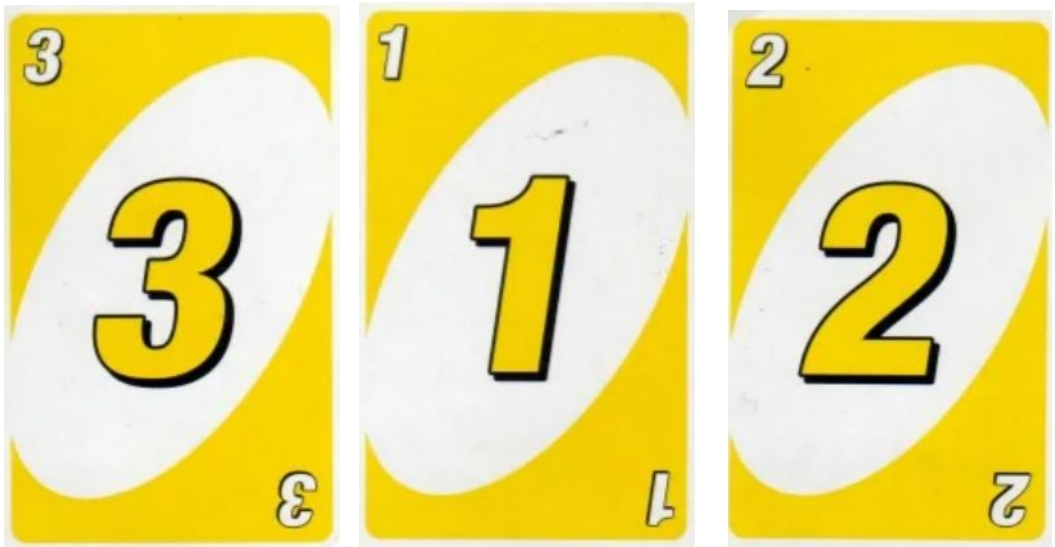
i

f

j

$quicksort(A,\ i,\ p-1)$ p $quicksort(A,\ p+1,\ f)$

i x f

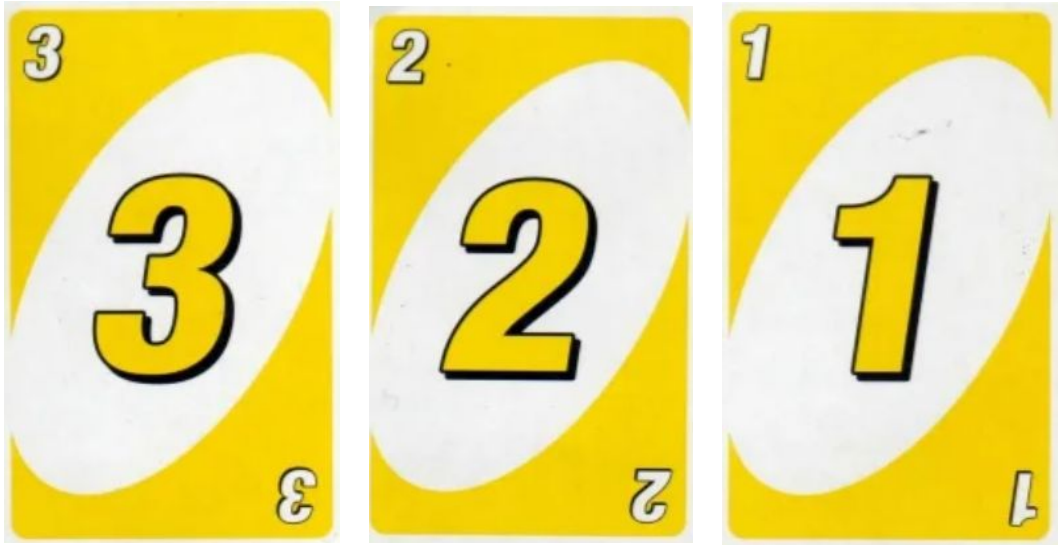


	A[x]	A[f]
i	x	f

	p	A[f]
i	x	f

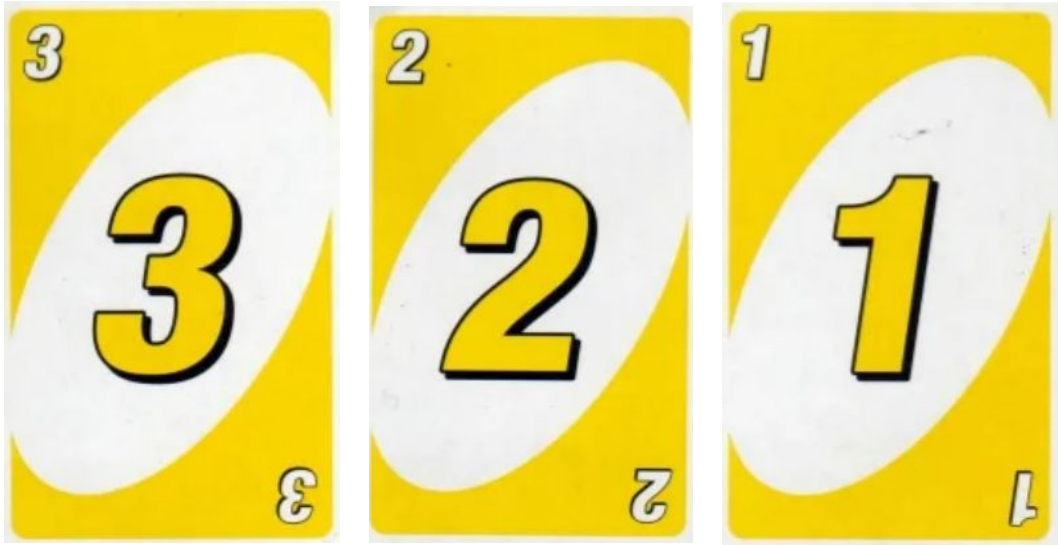
A[f] p

i x f



p

i f



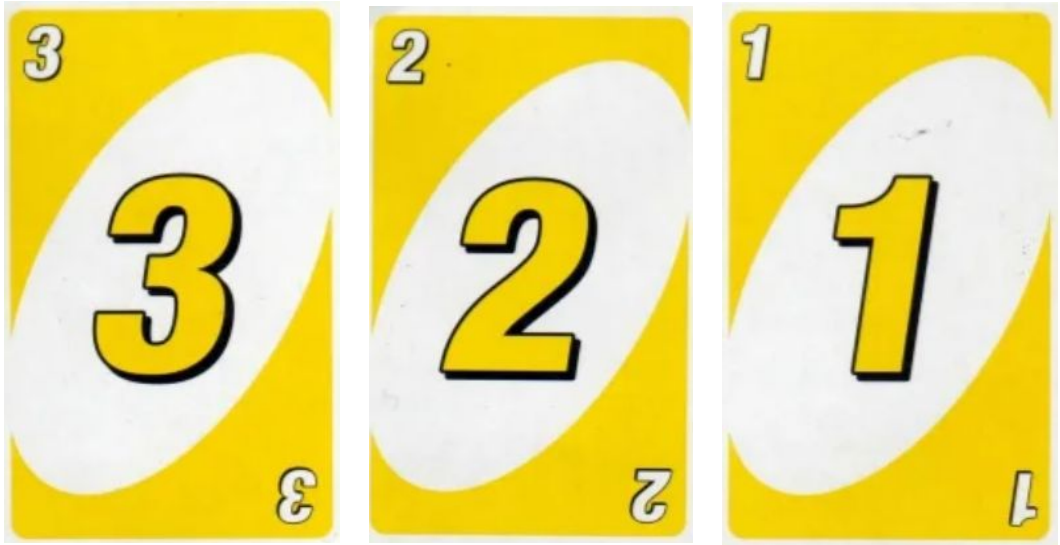
A[j]

A[k]

p

i

f



j

k

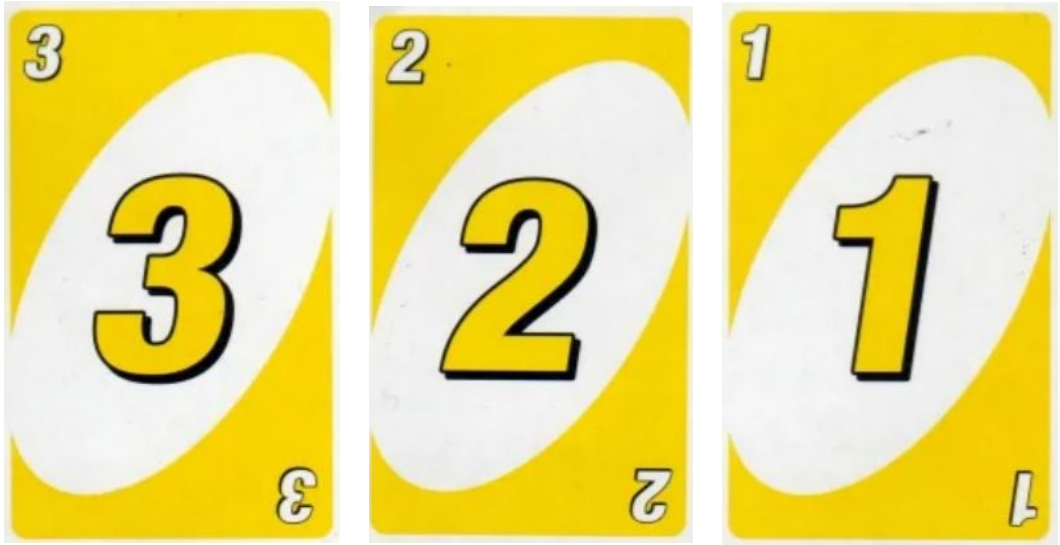
A[j]

A[k]

p

i

f



j

k

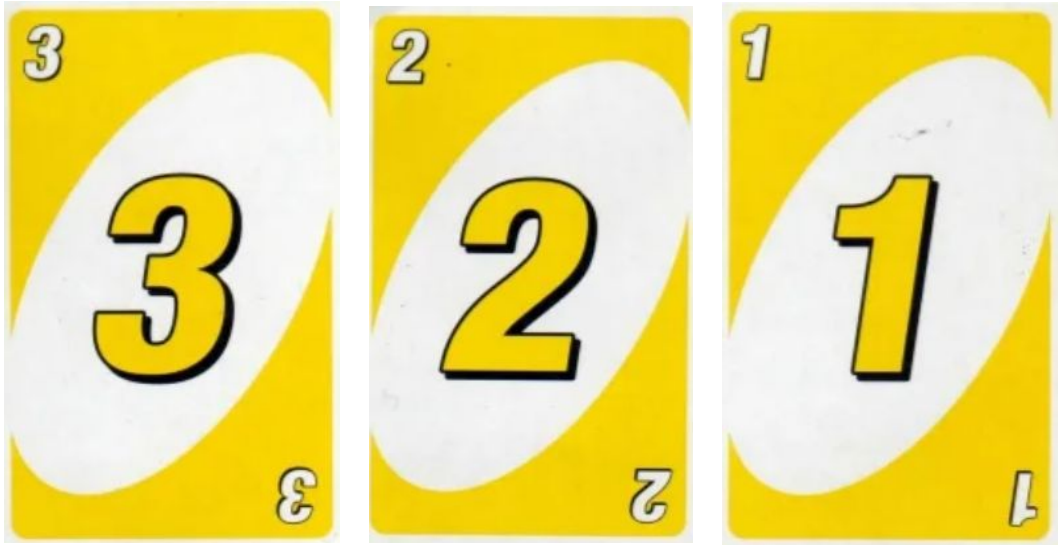
A[j]

A[k]

p

i

f



j

k

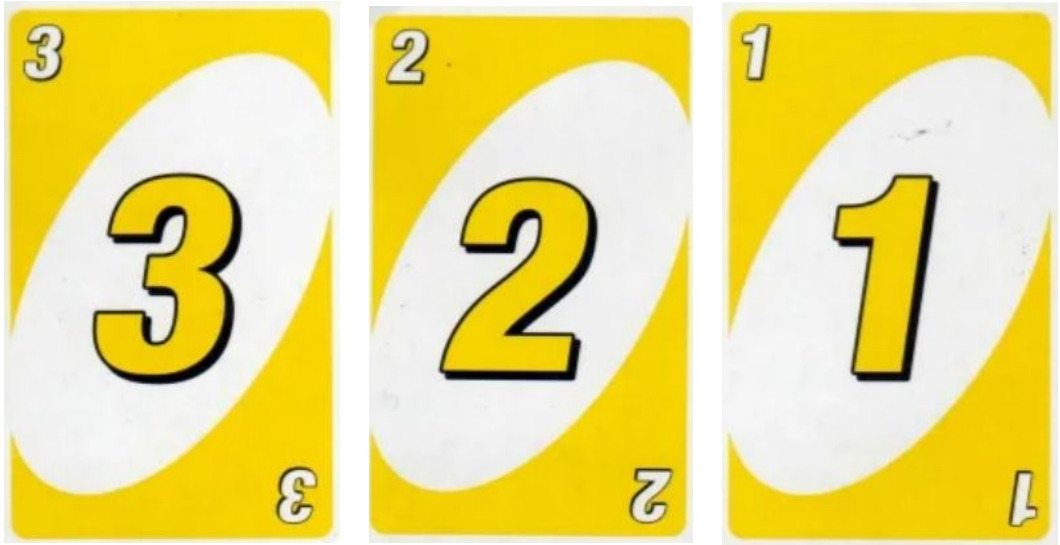
A[j]

A[k]

p

i

f



j

k

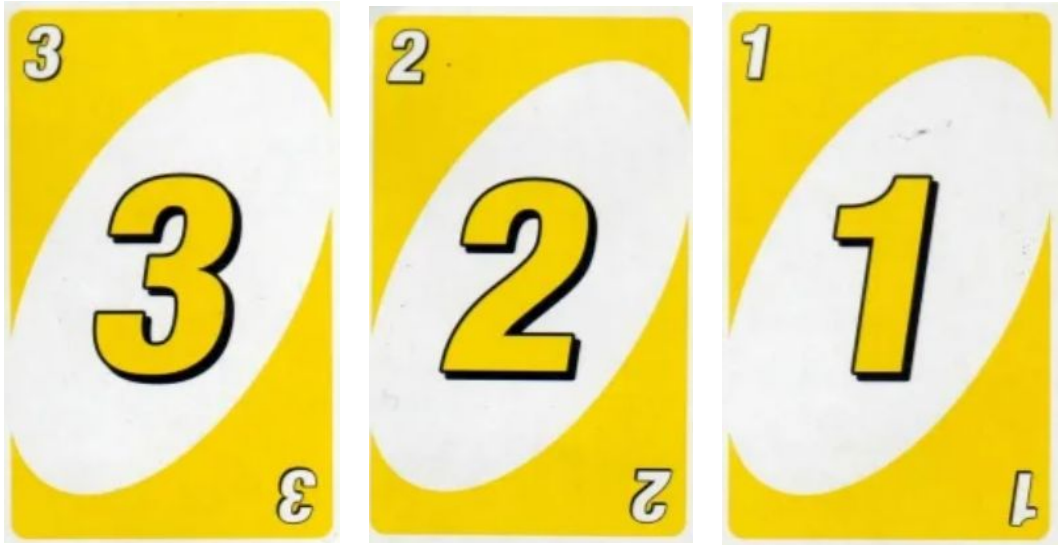
A[j]

A[k]

p

i

f



j

k

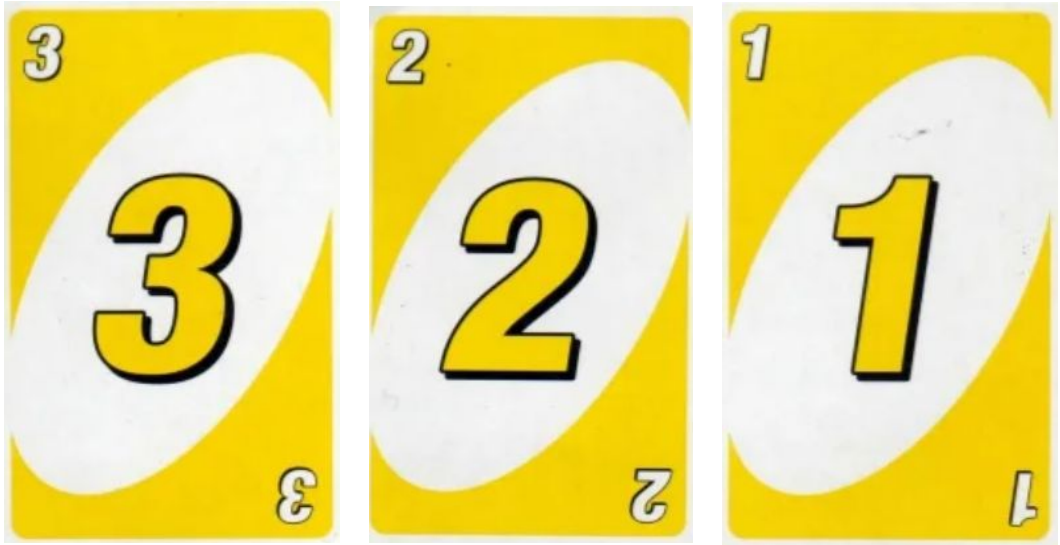
A[j]

A[k]

A[f]

i

f



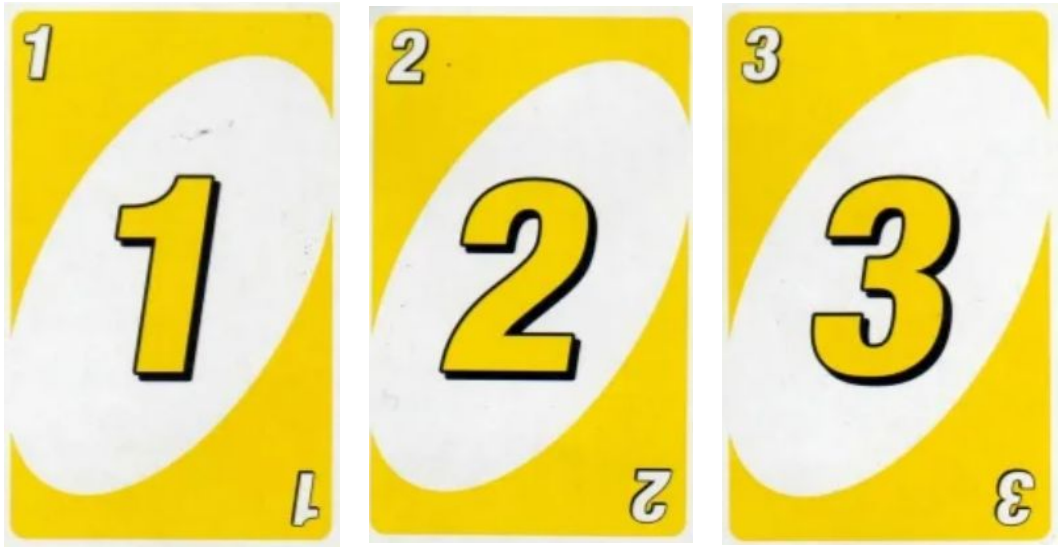
j

k

A[j]

A[f] A[k]

i f



j

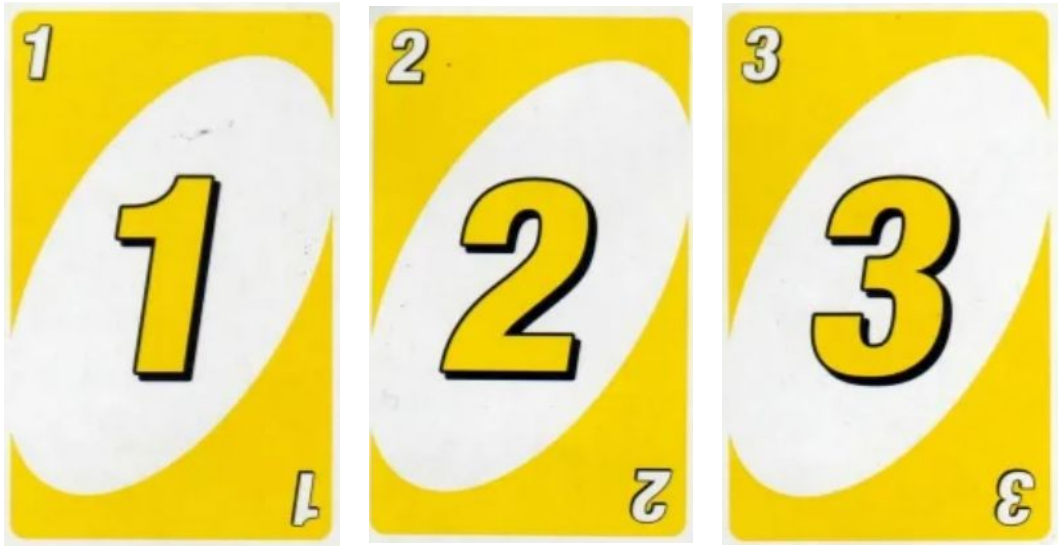
k

A[j]

p

i

f



j

p’

$quicksort(A,\ p'+1,\ f')$

p

$quicksort(A,\ p+1,\ f)$

f’

2
2
2

3
3
3

6
6
6

5
5
5

2
2
2

3
3
3

5
5
5

6
6
6

p’

$quicksort(A,\ p'+1,\ f')$

p

$quicksort(A,\ p+1,\ f)$

$quicksort(A,\ i,\ p-1)$ p $quicksort(A,\ p+1,\ f)$


1
1
2
2
3
3
4
4
5
5
6
6


1
2
3
4
5
6