

Деякі методи сортування масивів


## Сортування бульбашкою

Алгоритм працює таким чином:

- у поданому наборі даних (списку чи масиві) порівнюються два сусідні елементи (якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями.
- прохід по списку продовжується доти, доки дані не будуть відсортованими.

<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
[ 3,	5,	1,	4,	2 ]

<sup>0</sup>   <sup>1</sup>   <sup>2</sup>   <sup>3</sup>   <sup>4</sup>  
[ 3, 5, 1, 4, 2 ]



$a[0] > a[1] ?$



*0*   *1*   *2*   *3*   *4*  
[ 3 , 5 , 1 , 4 , 2 ]



a[1] > a[2] ?

$0 \quad 1 \quad 2 \quad 3 \quad 4$   
[ 3, 5, 1, 4, 2 ]

1 2

a[1] > a[2] ?

<sup>0</sup>   <sup>1</sup>   <sup>2</sup>   <sup>3</sup>   <sup>4</sup>  
[ 3, 5, 1, 4, 2 ]



<sup>0</sup>   <sup>1</sup>   <sup>2</sup>   <sup>3</sup>   <sup>4</sup>  
[ 3, 1, 5, 4, 2 ]

$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 5, & 1, & 4, & 2 & ] \end{matrix}$



$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 1, & 5, & 4, & 2 & ] \end{matrix}$

$\begin{matrix} \uparrow & \uparrow \\ 2 & 3 \end{matrix}$

$a[2] > a[3] ?$



<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 5, 1, 4, 2 ]



<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 5, 4, 2 ]

A red curved arrow points from the element 5 at index 2 to the element 4 at index 3. Below the array, a red horizontal line with upward-pointing arrows at each end connects the indices 2 and 3, with the numbers 2 and 3 written below the arrows respectively.

a[2] > a[3] ?

$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 5, & 1, & 4, & 2 & ] \end{matrix}$



$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 1, & 5, & 4, & 2 & ] \end{matrix}$



$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 1, & 4, & 5, & 2 & ] \end{matrix}$

$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 5, & 1, & 4, & 2 & ] \end{matrix}$

$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 1, & 5, & 4, & 2 & ] \end{matrix}$

$\begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [ & 3, & 1, & 4, & 5, & 2 & ] \end{matrix}$

$\begin{matrix} \uparrow & \uparrow \\ 3 & 4 \end{matrix}$

$a[3] > a[4] ?$

<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 5, 1, 4, 2 ]

<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 5, 4, 2 ]

<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 4, 5, 2 ]

3 4

a[3] > a[4] ?

<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 5, 1, 4, 2 ]



<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 5, 4, 2 ]



<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 4, 5, 2 ]



<sup>0</sup> <sup>1</sup> <sup>2</sup> <sup>3</sup> <sup>4</sup>  
[ 3, 1, 4, 2, 5 ]

# Сортування бульбашкою

## Алгоритм

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

/\* якщо ця двійка невідсортована \*/

якщо  $A[i-1] > A[i]$  тоді

/\* поміняти місцями  $i$  запам'ятати, що щось змінилось \*/

переставити(  $A[i-1]$ ,  $A[i]$  )

переставлені = істина

кінець якщо

кінець для

доки переставлені

[https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F\\_%D0%B1%D1%83%D0%BB%D1%8C%D0%B1%D0%B0%D1%88%D0%BA%D0%BE%D1%8E](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D1%83%D0%BB%D1%8C%D0%B1%D0%B0%D1%88%D0%BA%D0%BE%D1%8E)

повторювати

доки      переставлені

повторювати

доки

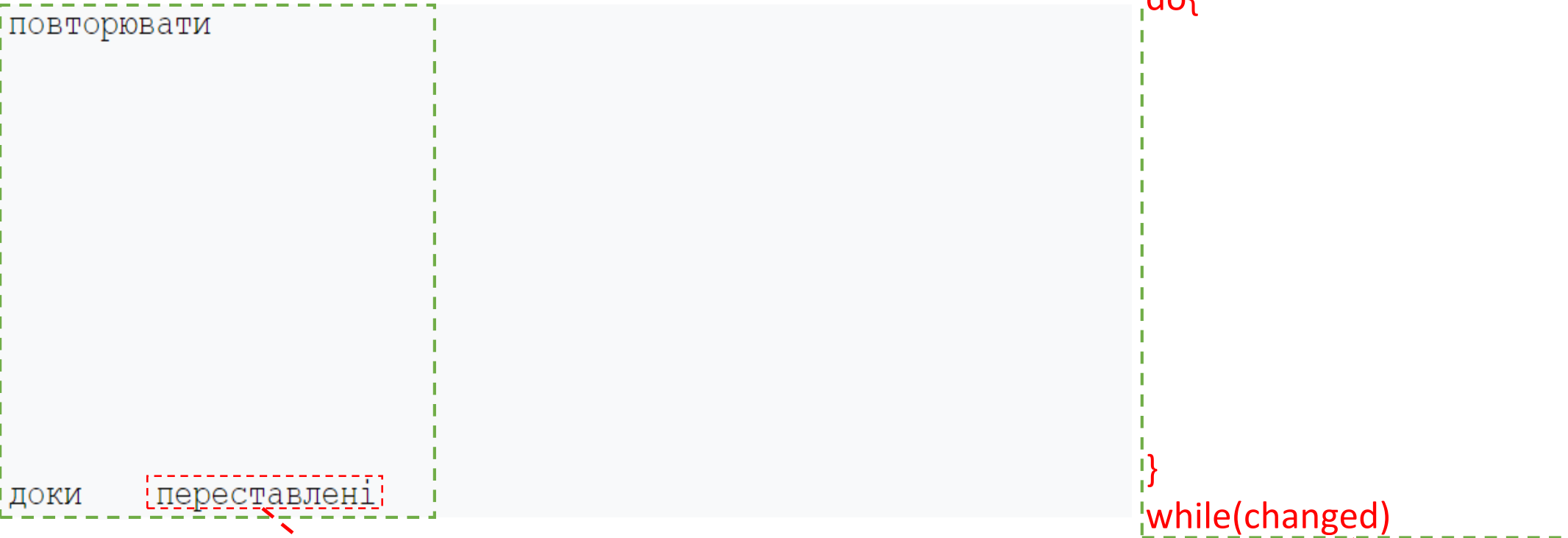
переставлені

let changed

do{

}

while(changed)





повторювати

переставлені = хиба

доки переставлені

let changed

do{



}

while(changed)

```
повторювати  
  переставлені = хиба
```

```
доки    переставлені
```

```
let changed
```

```
do{
```

```
  changed = false
```

```
}
```

```
while(changed)
```

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

кінець для

доки переставлені

let changed

do{

changed = false

}

while(changed)

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

кінець для

доки переставлені

let changed

do{

changed = false

for(let i=1; i< a.length; i++){

}

}

while(changed)

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

/\* якщо ця двійка невідсортована \*/

якщо  $A[i-1] > A[i]$  тоді

кінець якщо

кінець для

доки переставлені

let changed

do{

changed = false

for(let i=1; i< a.length; i++){



}

}

while(changed)

```
повторювати
  переставлені = хиба
  для i = 1 включно до довжина(A) - 1 робити:
    /* якщо ця двійка невідсортована */
    якщо A[i-1] > A[i] тоді

      кінець якщо
    кінець для
доки переставлені
```

```
let changed
do{
  changed = false
  for(let i=1; i< a.length; i++){
    if( a[i-1] > a[i] ){

    }
  }
}
while(changed)
```

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

/\* якщо ця двійка невір'якована \*/

якщо  $A[i-1] > A[i]$  тоді

/\* поміняти місцями  $i$  запам'ятати, що щось змінилось \*/  
переставити(  $A[i-1]$ ,  $A[i]$  )

кінець якщо

кінець для

доки переставлені

let changed

do{

changed = false

for(let i=1; i< a.length; i++){

if(  $a[i-1] > a[i]$  ){

}

}

}

while(changed)

повторювати

переставлені = хиба

для  $i = 1$  включно до довжина(A) - 1 робити:

/\* якщо ця двійка невір'якована \*/

якщо  $A[i-1] > A[i]$  тоді

/\* поміняти місцями  $i$  запам'ятати, що щось змінилось \*/  
переставити(  $A[i-1]$ ,  $A[i]$  )

кінець якщо

кінець для

доки переставлені

let changed

do{

changed = false

for(let i=1; i< a.length; i++){

if(  $a[i-1] > a[i]$  ){

let tmp =  $a[i-1]$ ;

$a[i-1] = a[i]$

$a[i] = tmp$

}

}

}

while(changed)



```
повторювати
  переставлені = хиба
  для i = 1 включно до довжина(A) - 1 робити:
    /* якщо ця двійка невірнорядкована */
    якщо A[i-1] > A[i] тоді
      /* поміняти місцями i запам'ятати, що щось змінилось */
      переставити( A[i-1], A[i] )
      переставлені = істина
    кінець якщо
  кінець для
доки переставлені
```

```
let changed
do{
  changed = false
  for(let i=1; i< a.length; i++){

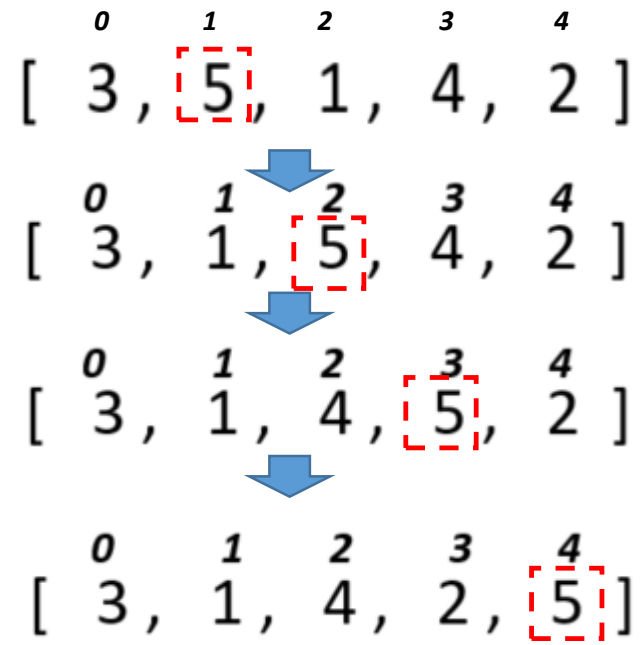
    if( a[i-1] > a[i] ){
      let tmp = a[i-1];
      a[i-1] = a[i]
      a[i] = tmp
    }
  }
}
while(changed)
```

```
повторювати
  переставлені = хиба
  для i = 1 включно до довжина(A) - 1 робити:
    /* якщо ця двійка невідсортована */
    якщо A[i-1] > A[i] тоді
      /* поміняти місцями i запам'ятати, що щось змінилось */
      переставити( A[i-1], A[i] )
      переставлені = істина
    кінець якщо
  кінець для
доки    переставлені
```

```
let changed
do{
  changed = false
  for(let i=1; i< a.length; i++){

    if( a[i-1] > a[i] ){
      let tmp = a[i-1];
      a[i-1] = a[i]
      a[i] = tmp
      changed = true;
    }
  }
}
while(changed)
```

Алгоритм за один прохід дозволяє перемістити найбільший елемент у області сортування у свою правильну позицію



Початковий масив

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>5, <sup>4</sup>1 ]

Початковий масив

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>5, <sup>4</sup>1 ]

Результат 1-шого проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>1, <sup>4</sup>5 ]

Початковий масив

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>5, <sup>4</sup>1 ]

Результат 1-шого проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>1, <sup>4</sup>5 ]

Результат 2-го проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>1, <sup>3</sup>4, <sup>4</sup>5 ]

Початковий масив

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>5, <sup>4</sup>1 ]

Результат 1-шого проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>1, <sup>4</sup>5 ]

Результат 2-го проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>1, <sup>3</sup>4, <sup>4</sup>5 ]

Результат 3-го проходу

[ <sup>0</sup>2, <sup>1</sup>1, <sup>2</sup>3, <sup>3</sup>4, <sup>4</sup>5 ]

Початковий масив

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>5, <sup>4</sup>1 ]

Результат 1-шого проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>4, <sup>3</sup>1, <sup>4</sup>5 ]

Результат 2-го проходу

[ <sup>0</sup>2, <sup>1</sup>3, <sup>2</sup>1, <sup>3</sup>4, <sup>4</sup>5 ]

Результат 3-го проходу

[ <sup>0</sup>2, <sup>1</sup>1, <sup>2</sup>3, <sup>3</sup>4, <sup>4</sup>5 ]

Результат 4-го проходу

[ <sup>0</sup>1, <sup>1</sup>2, <sup>2</sup>3, <sup>3</sup>4, <sup>4</sup>5 ]



**Сортування змішуванням** ([англ. Cocktail sort](#)) — один із різновидів алгоритму [сортування бульбашкою](#).  
Відрізняється від [сортування бульбашкою](#) тим, що сортування відбувається в обох напрямках, міняючи напрямки при кожному проході

```
let leftIndex = 0
let rightIndex = array.length - 1

while (leftIndex < rightIndex) {
  for (let idx = leftIndex; idx < rightIndex; idx++) {
    if ( array[idx] > array[idx + 1]) {
      swap(array, idx, idx + 1)
    }
  }
  rightIndex--;

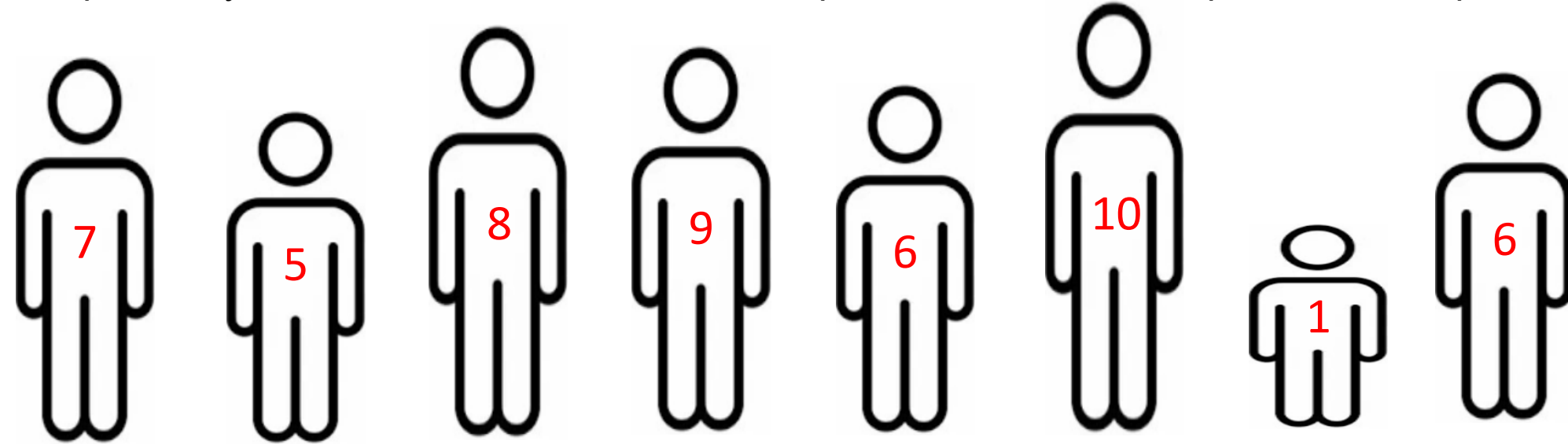
  for (let idx = rightIndex; idx > leftIndex; idx--) {
    if ( array[idx] < array[idx - 1]) {
      swap(array, idx, idx - 1)
    }
  }
  leftIndex++;
}
```

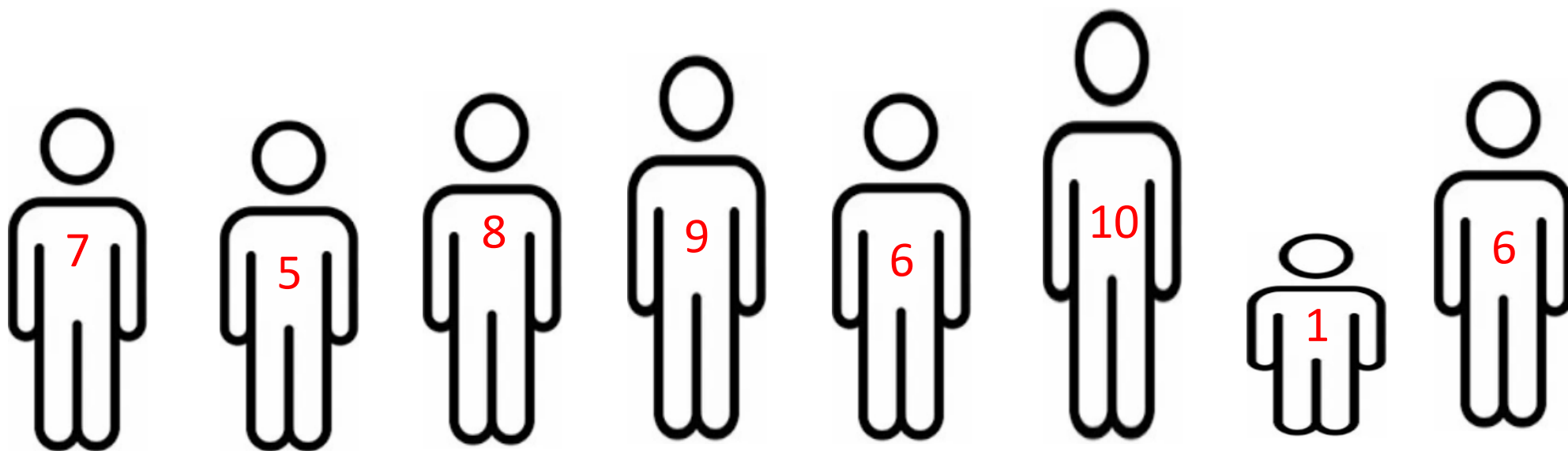
[https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F\\_%D0%B7%D0%BC%D1%96%D1%88%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%BC%D1%96%D1%88%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC)

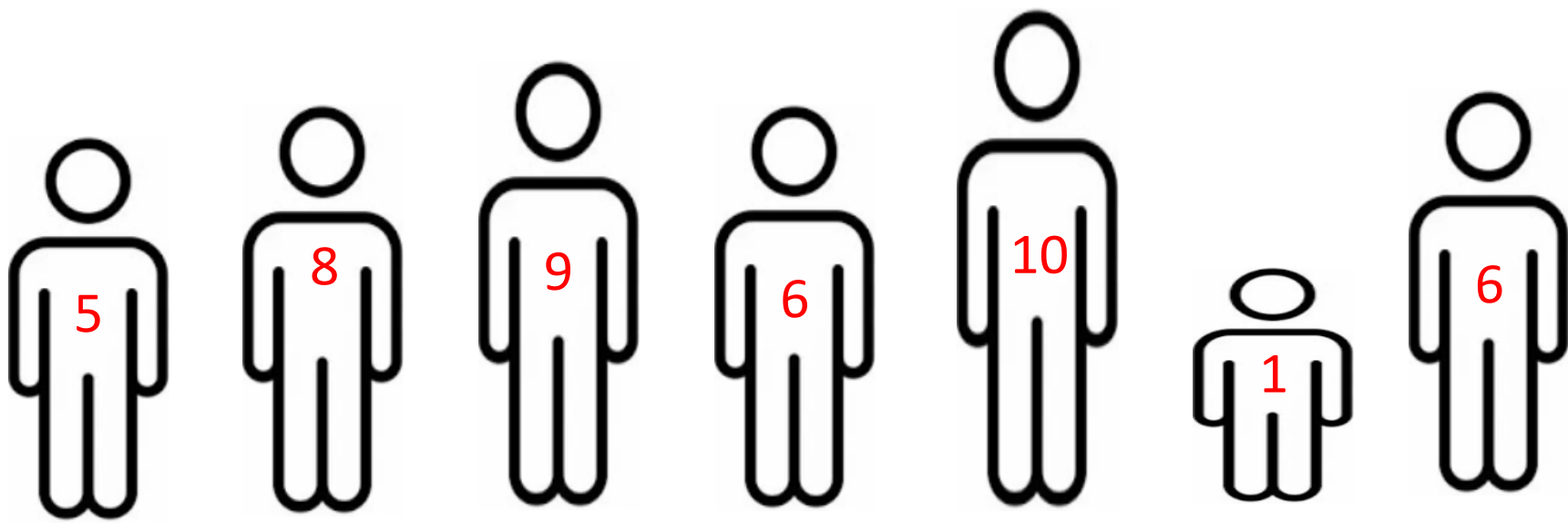
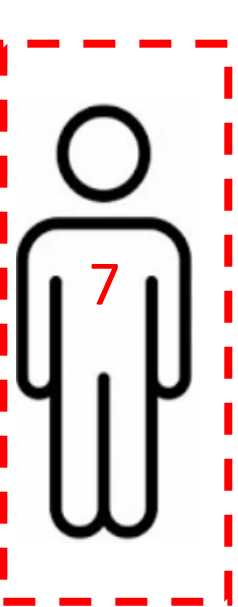
## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовні лівіше упорядковані за зростанням

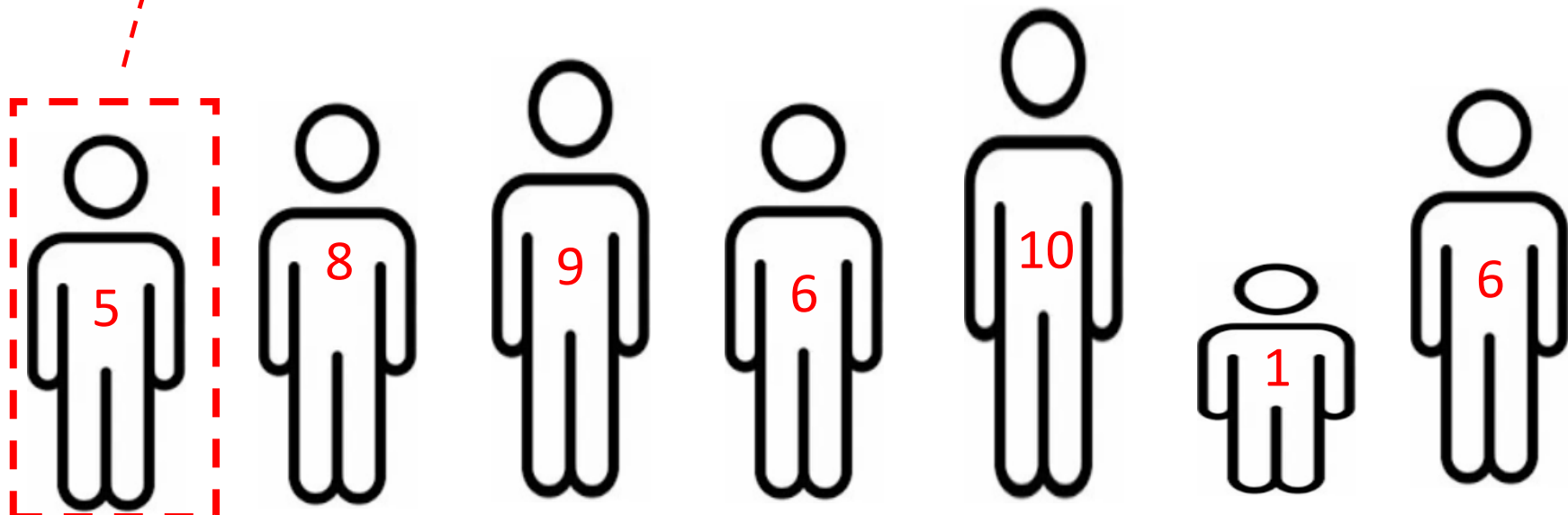


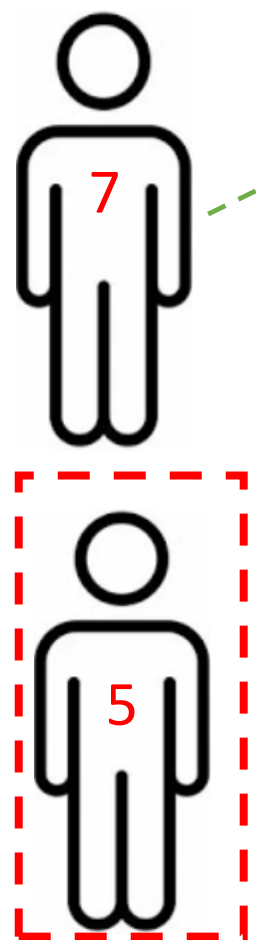




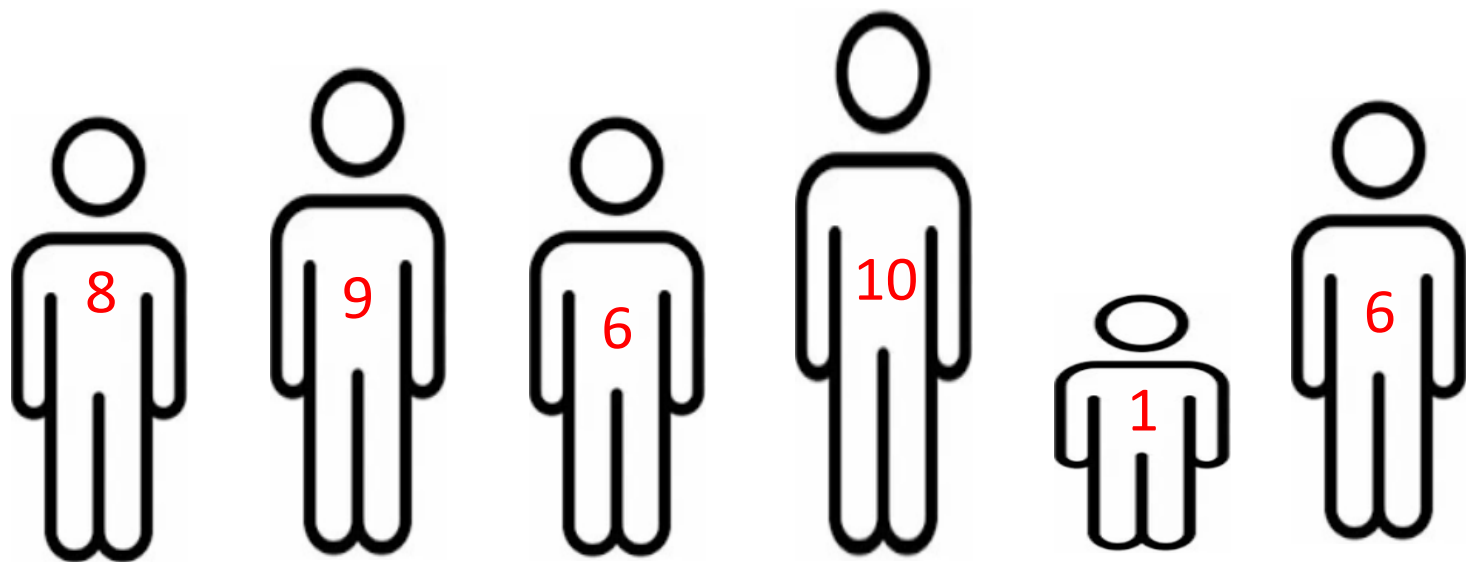


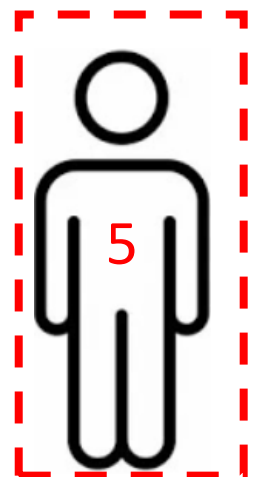
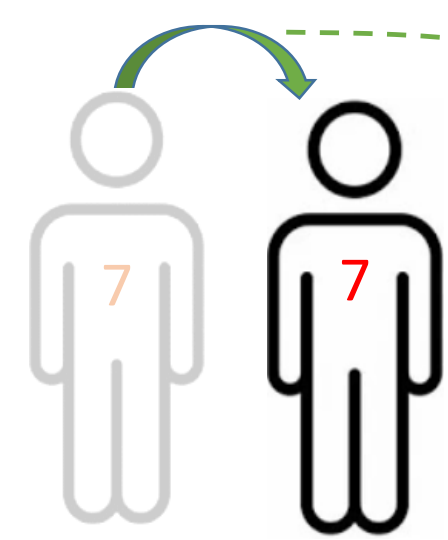
```
//----Сортування включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    .  
    .  
    .  
  }  
}
```



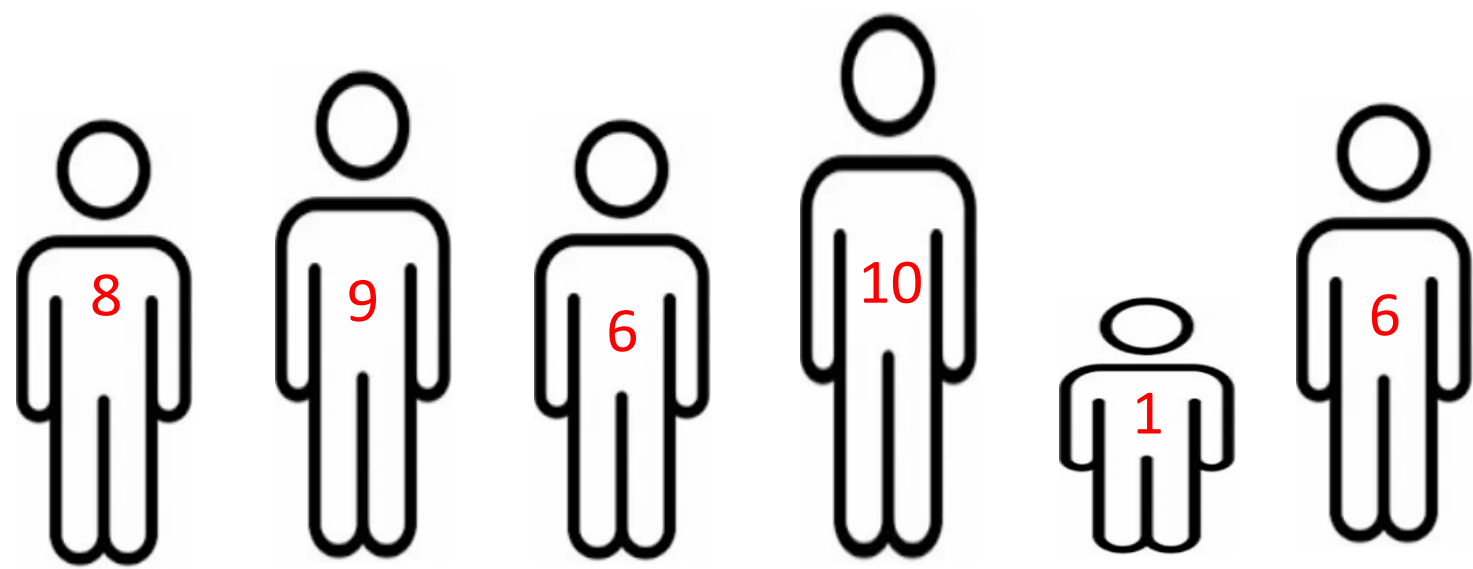


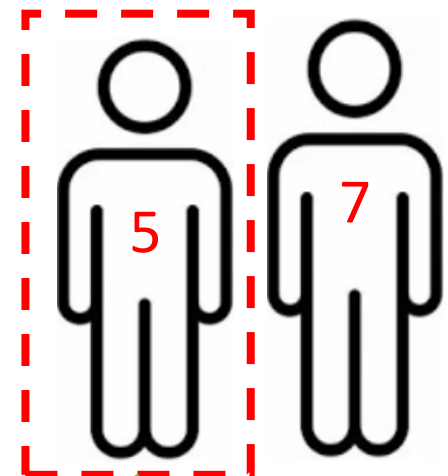
```
// ---- Сортивання включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    let i = k - 1  
    while (i >= 0 && arr[i] > currentElement) {  
      arr[i] = arr[i + 1]  
      i--  
    }  
    arr[i + 1] = currentElement  
  }  
}
```



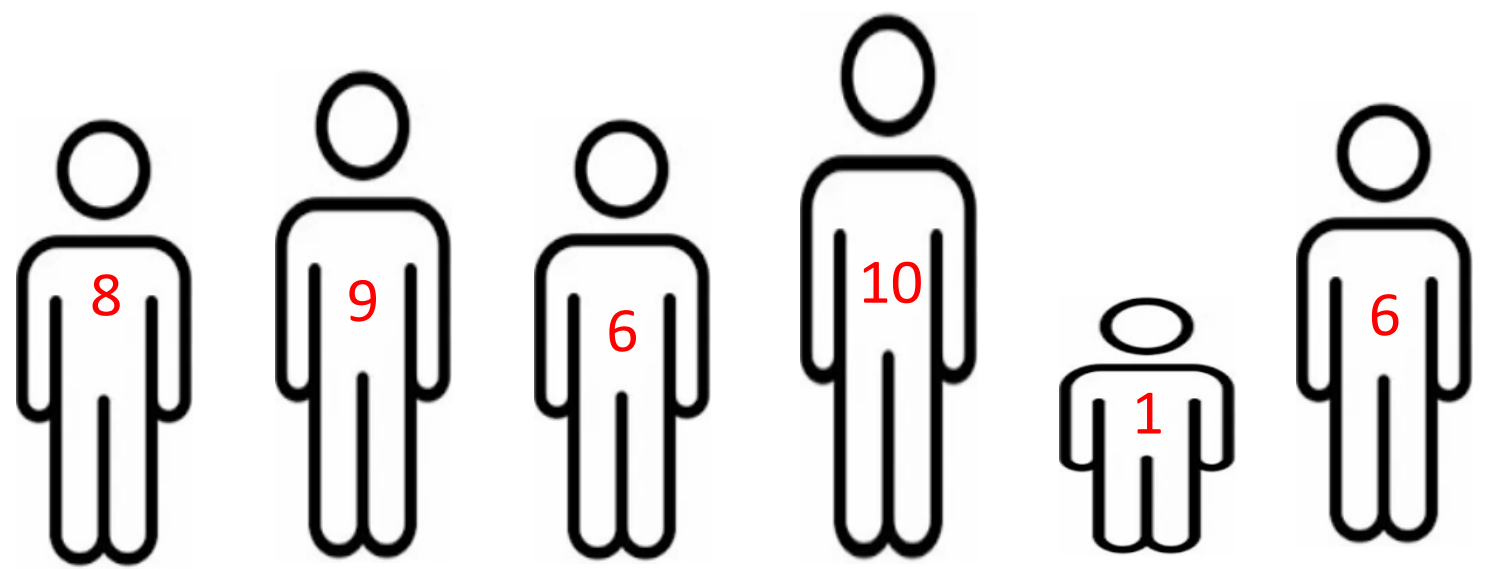


```
// ----- Сортивання включеннями
function insertSort(arr) {
  for (let k = 1; k < arr.length; k++) {
    const currentElement = arr[k]
    let i = k - 1
    while (i >= 0 && arr[i] > currentElement) {
      arr[i + 1] = arr[i]
      i = i - 1
    }
    arr[i + 1] = currentElement
  }
}
```

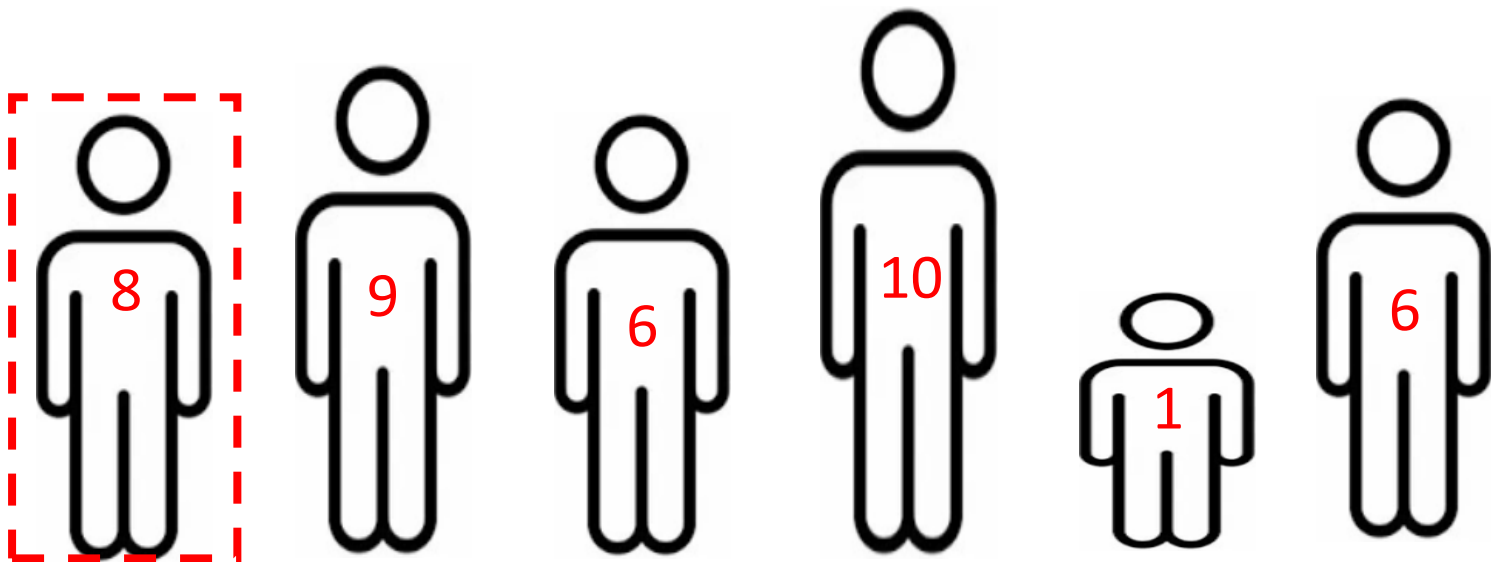
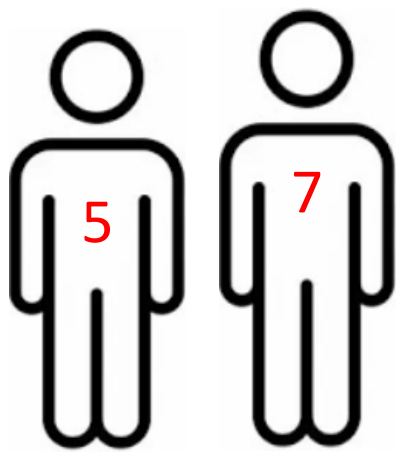


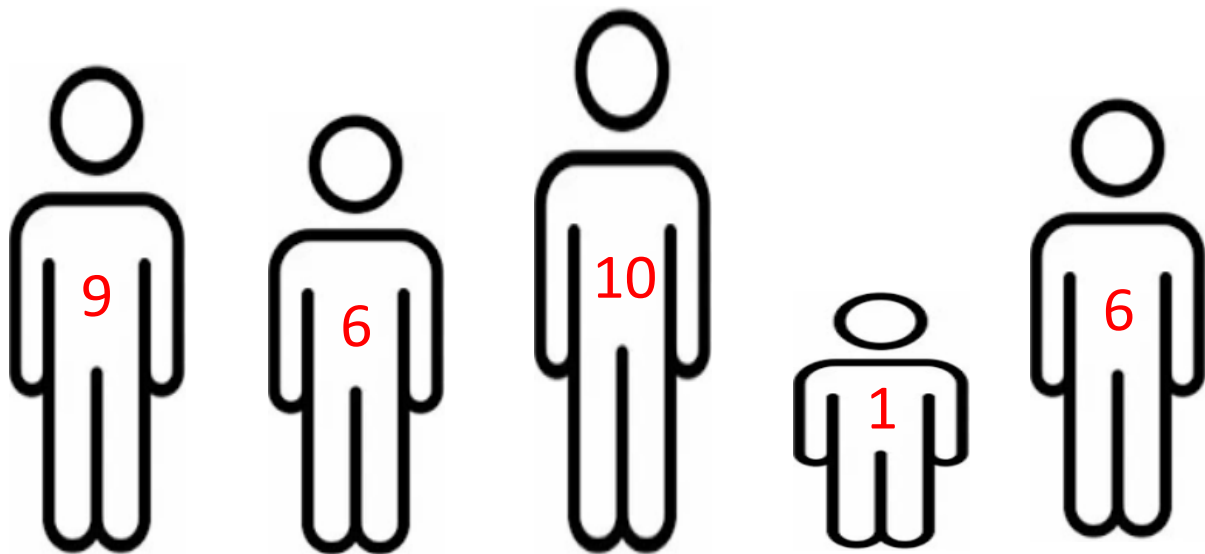
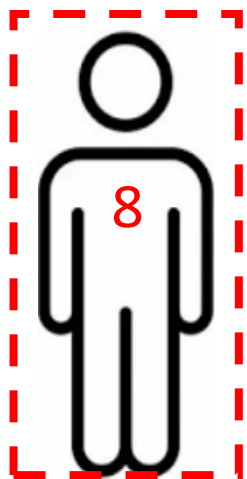
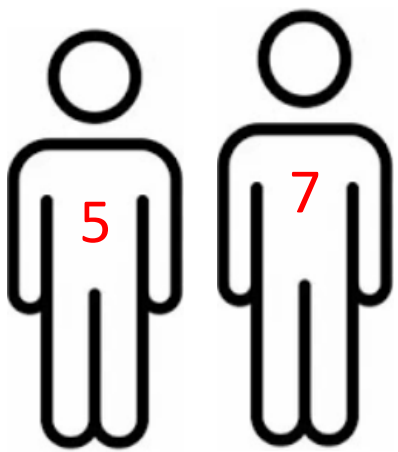


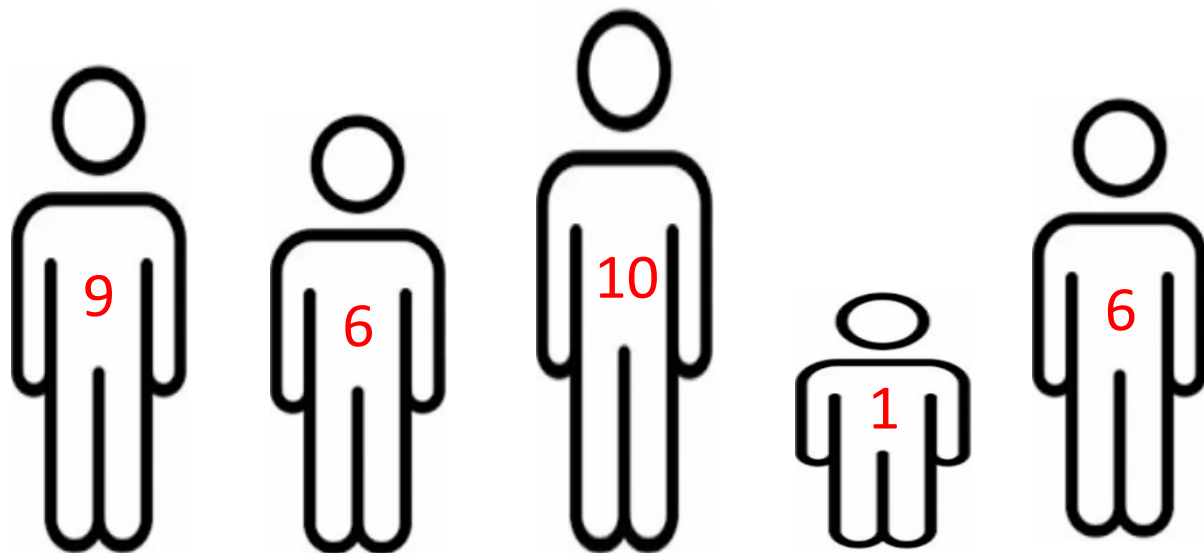
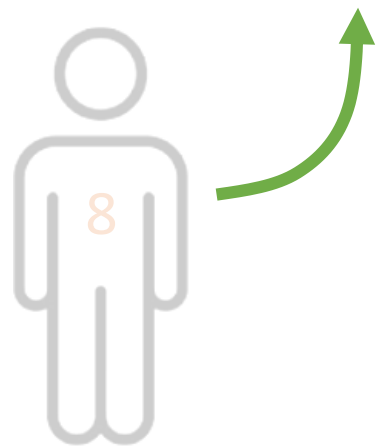
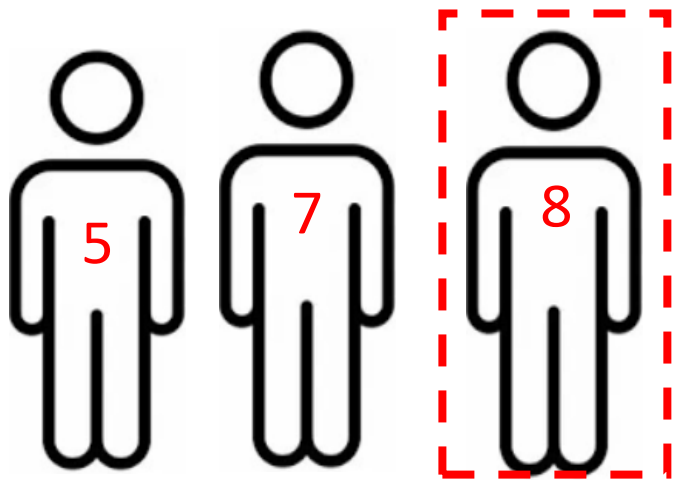
```
// ---- Сортивання включеннями
function insertSort(arr) {
  for (let k = 1; k < arr.length; k++) {
    const currentElement = arr[k]
    let i = k - 1
    while (i >= 0 && arr[i] > currentElement) {
      arr[i + 1] = arr[i]
      i = i - 1
    }
    arr[i + 1] = currentElement
  }
}
```

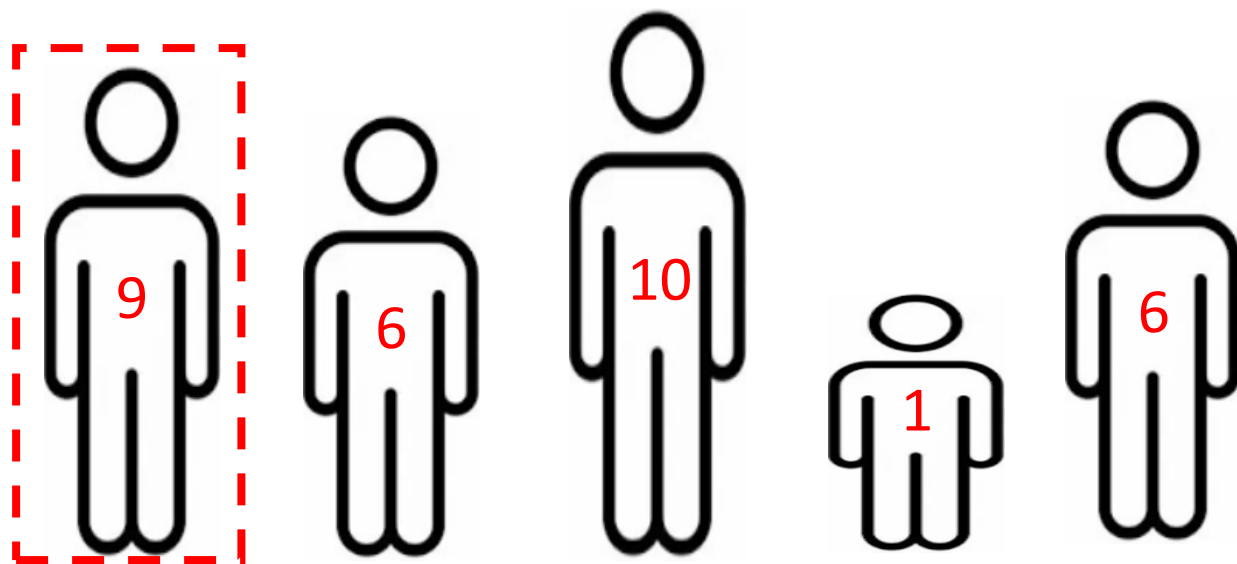
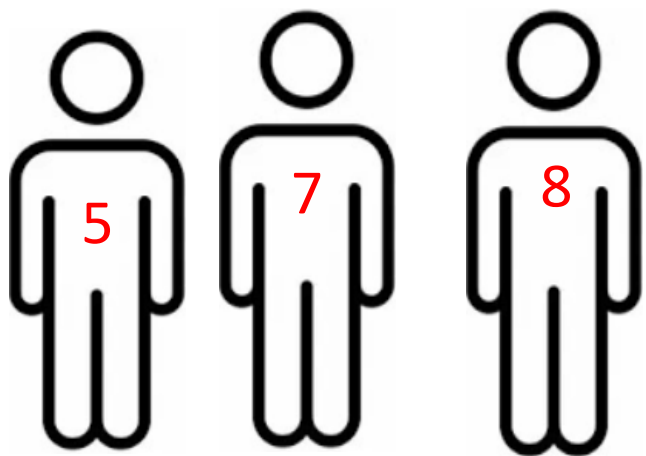


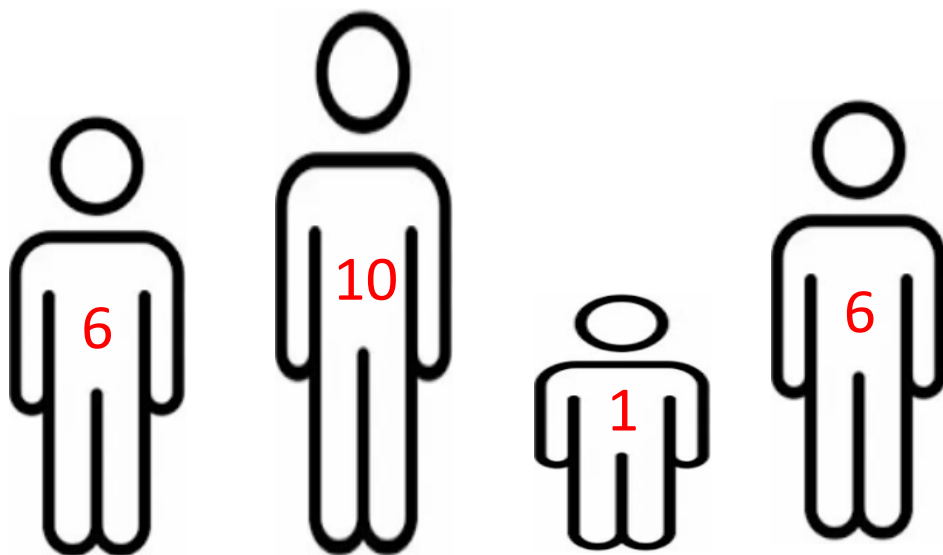
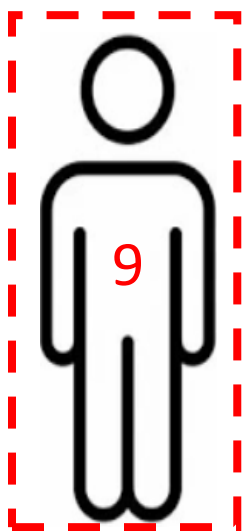
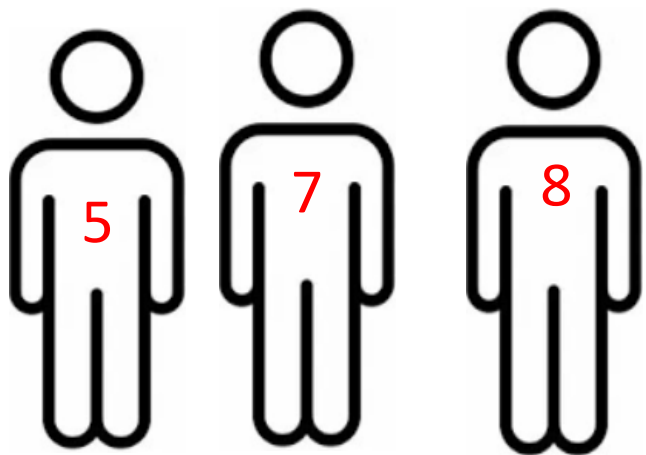


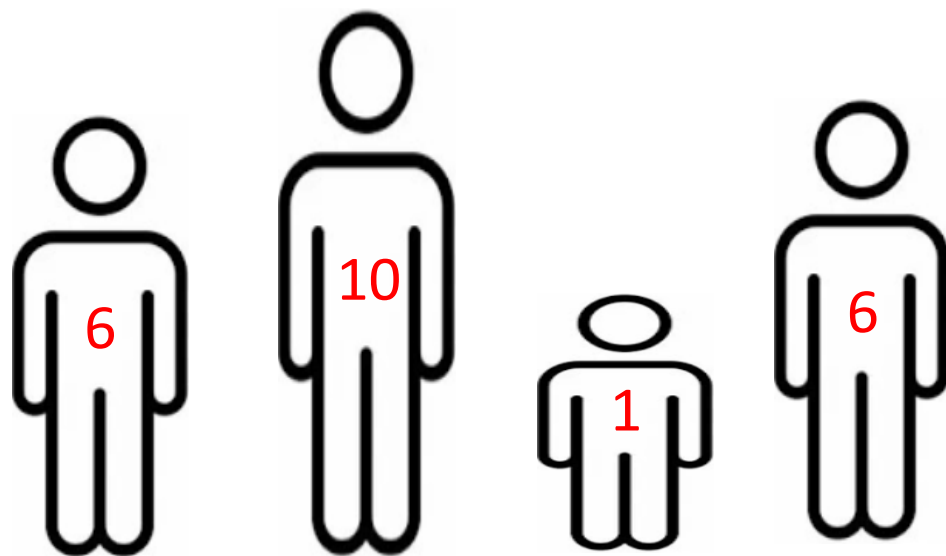
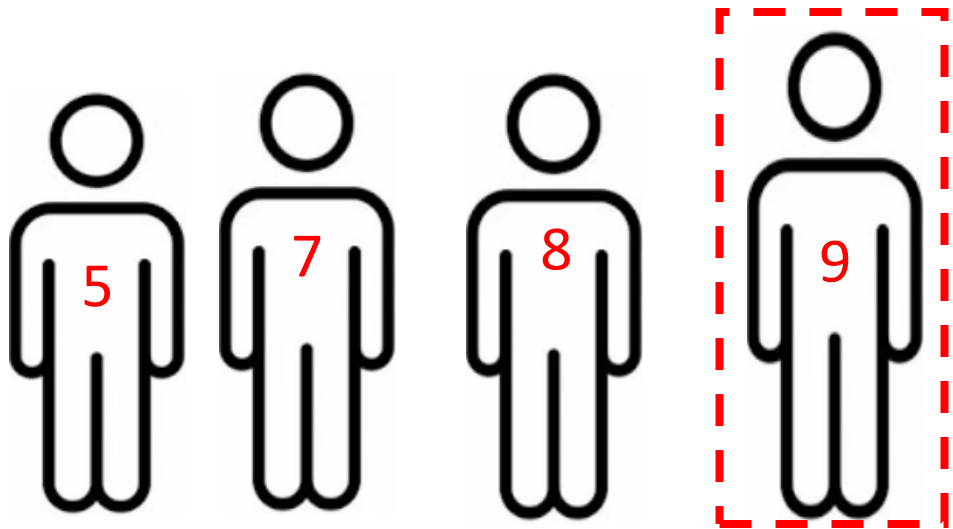


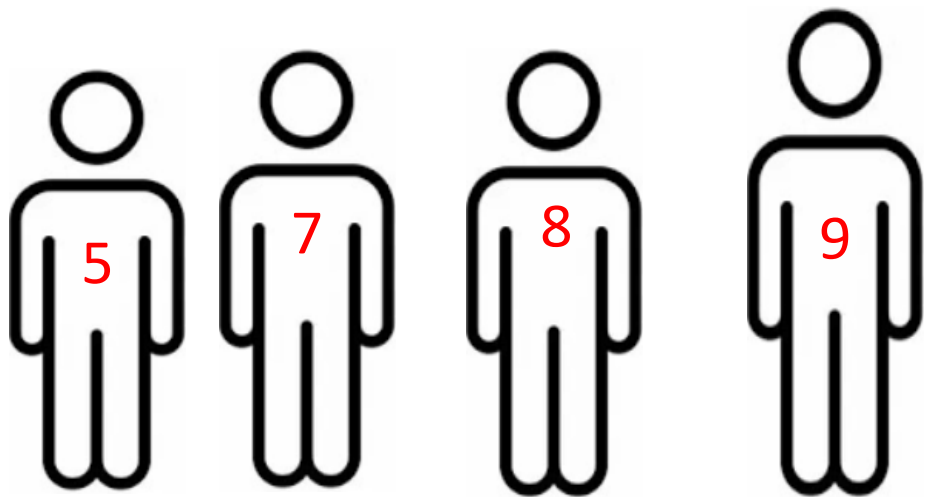




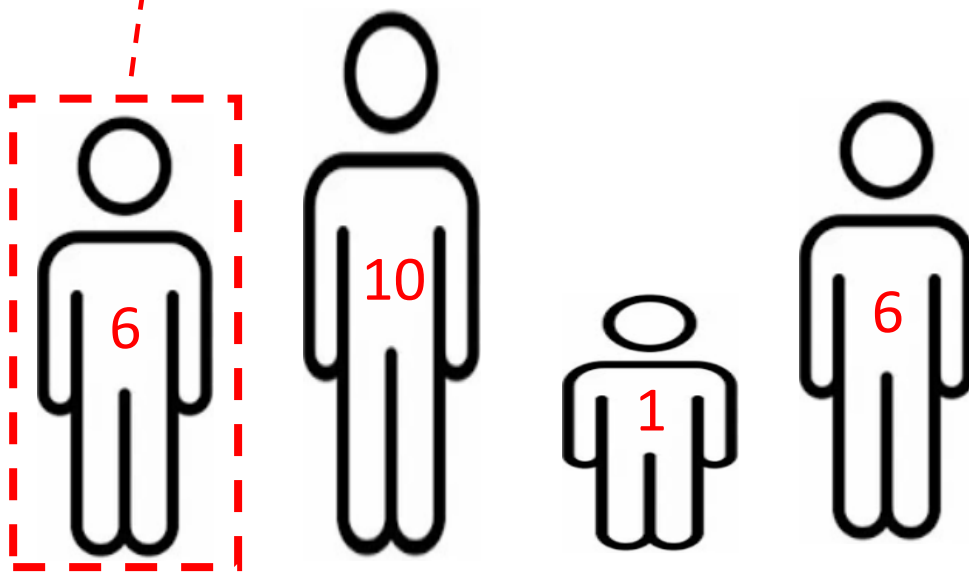


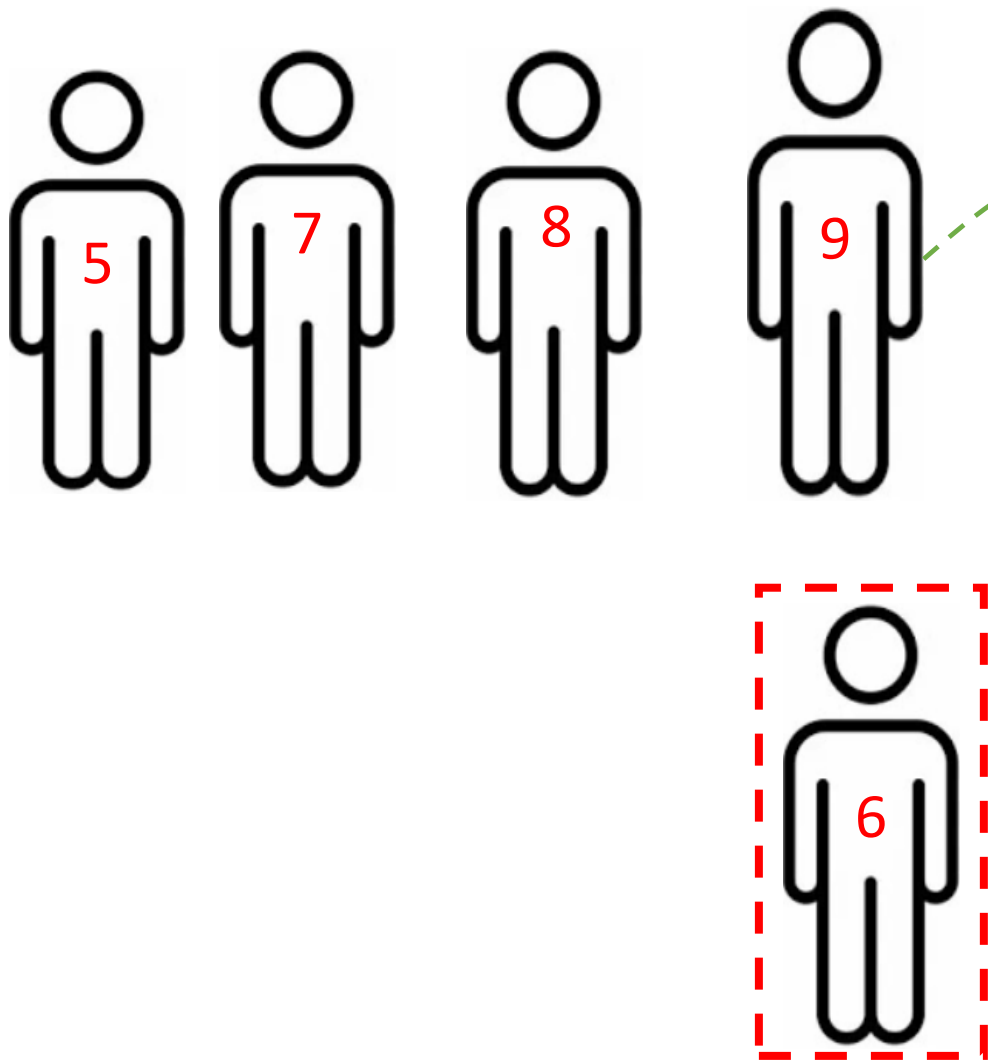




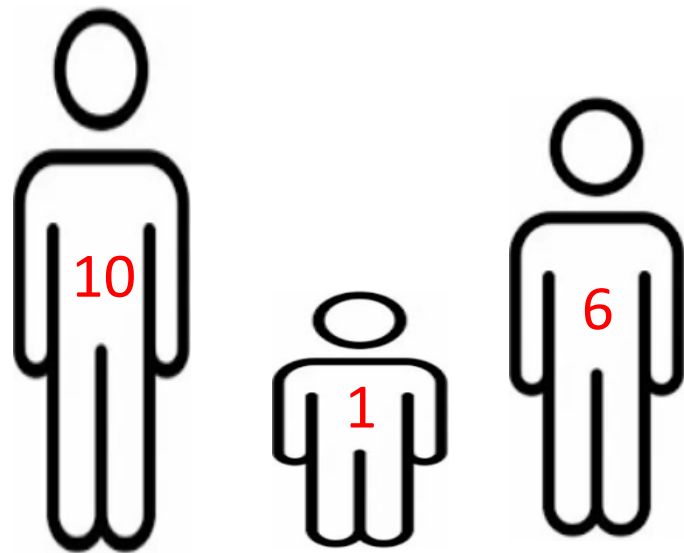


```
//-----Сортування включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    .  
    .  
    .  
  }  
}
```

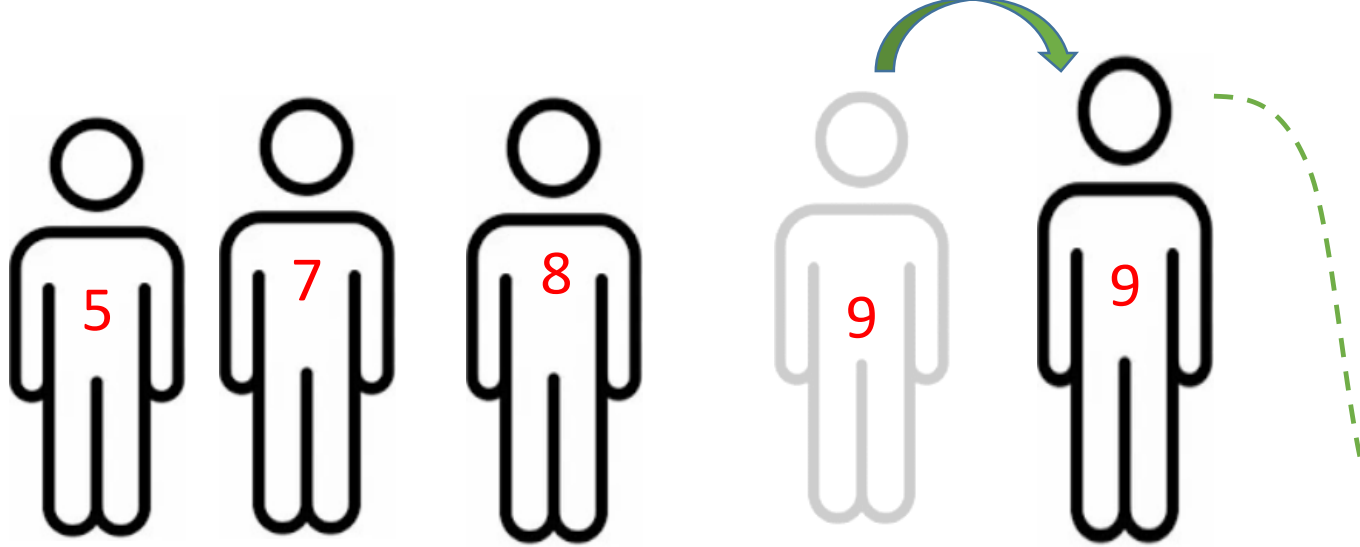




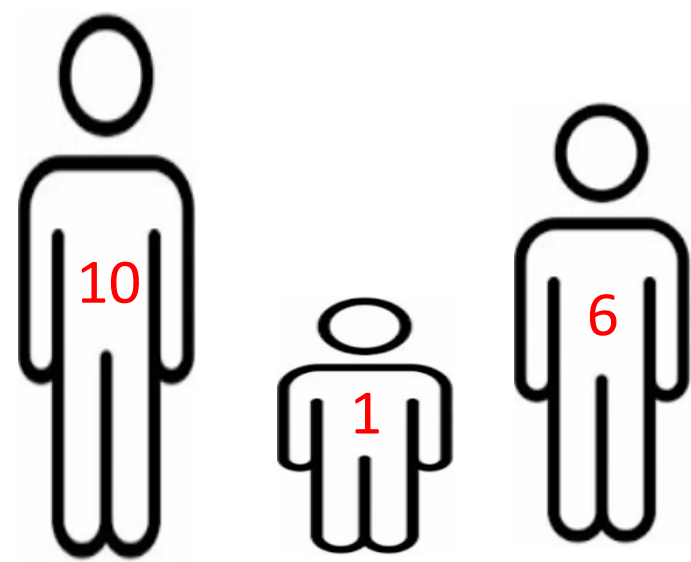
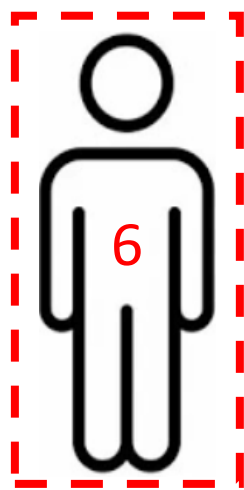
```
//-----Сортування включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    let i = k - 1  
    while (i >= 0 && arr[i] > currentElement) {  
      arr[i + 1] = arr[i]  
      i--  
    }  
    arr[i + 1] = currentElement  
  }  
}
```

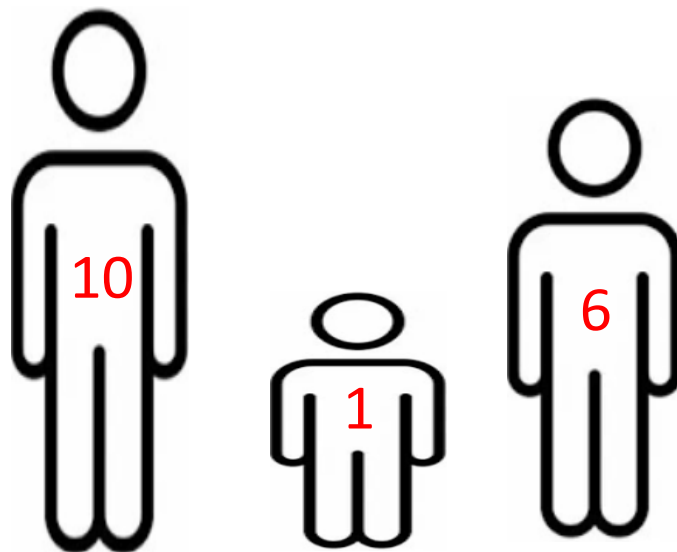
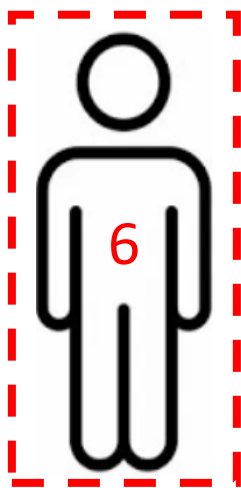
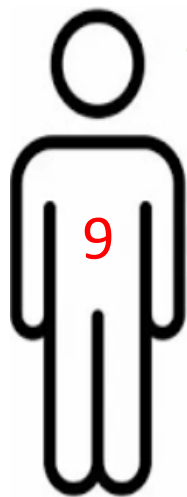
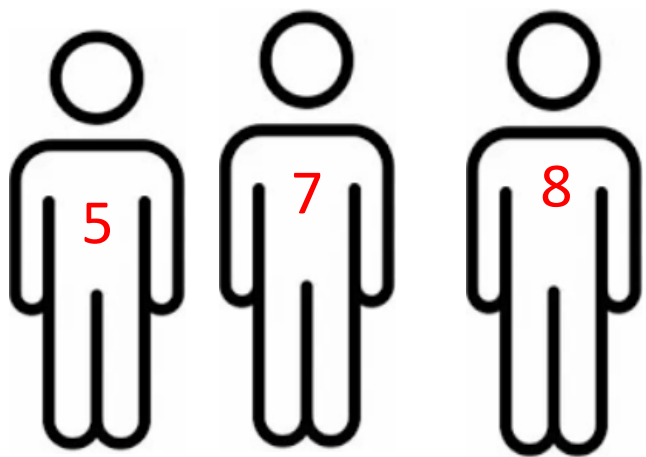




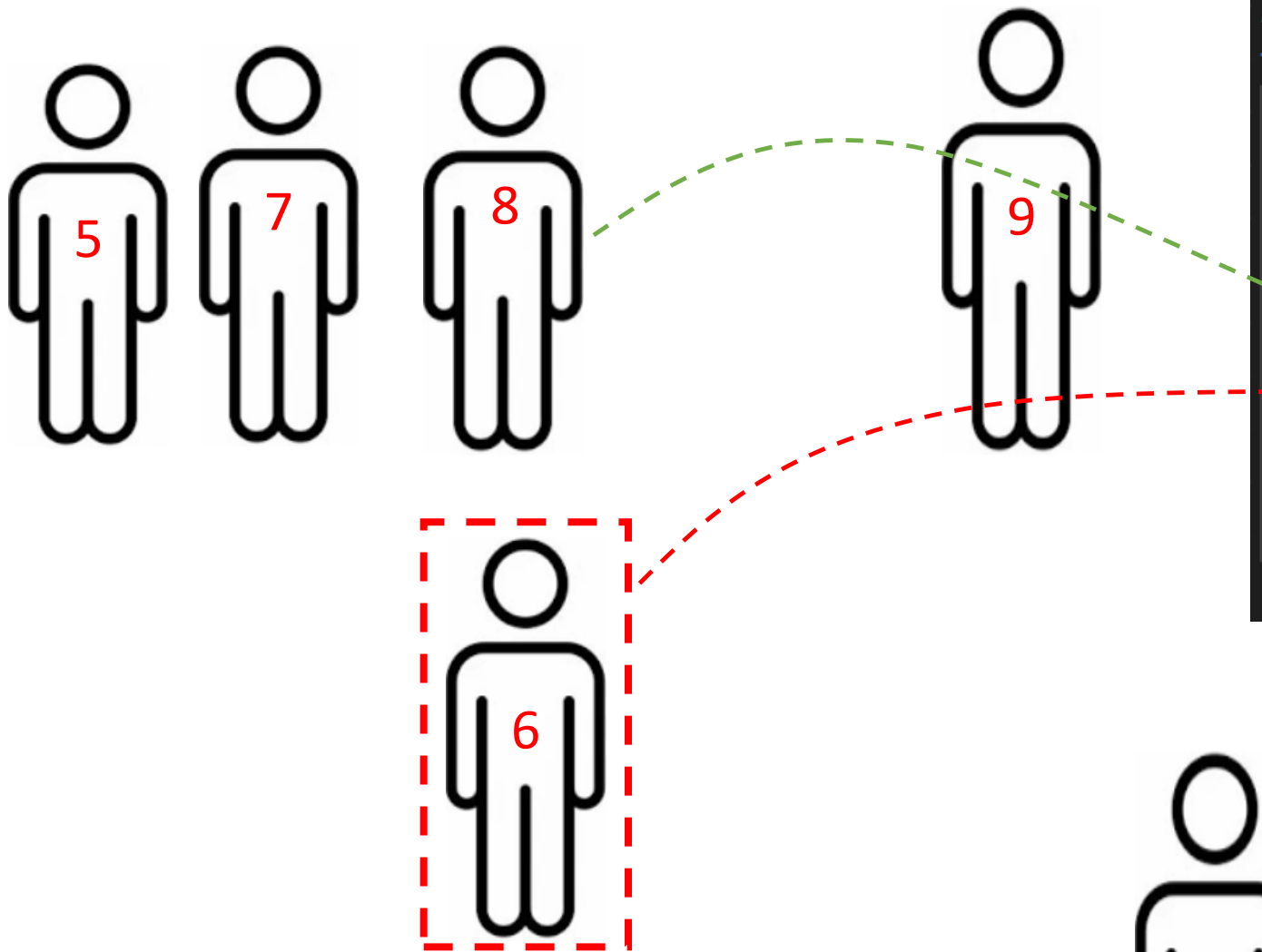


```
// ---- Сортивання включеннями
function insertSort(arr) {
  for (let k = 1; k < arr.length; k++) {
    const currentElement = arr[k]
    let i = k - 1
    while (i >= 0 && arr[i] > currentElement) {
      arr[i + 1] = arr[i]
      i = i - 1
    }
    arr[i + 1] = currentElement
  }
}
```

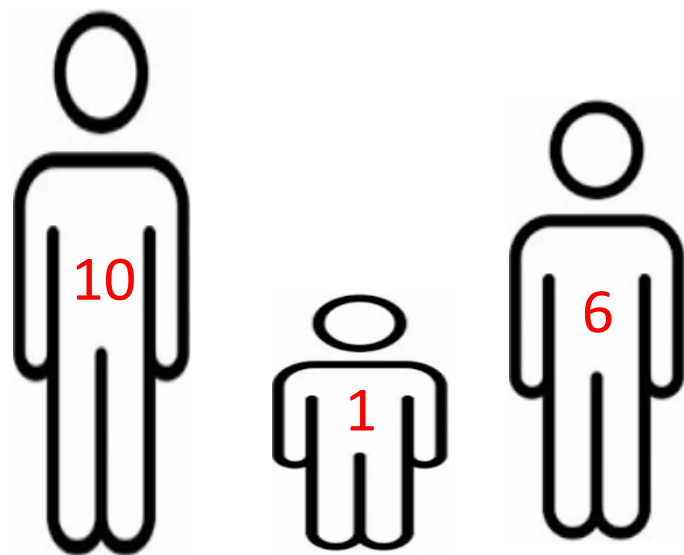


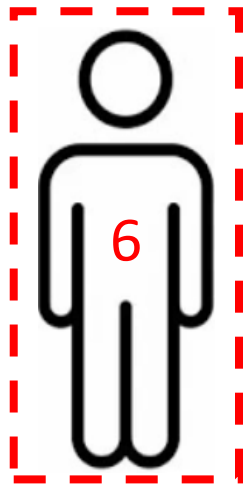
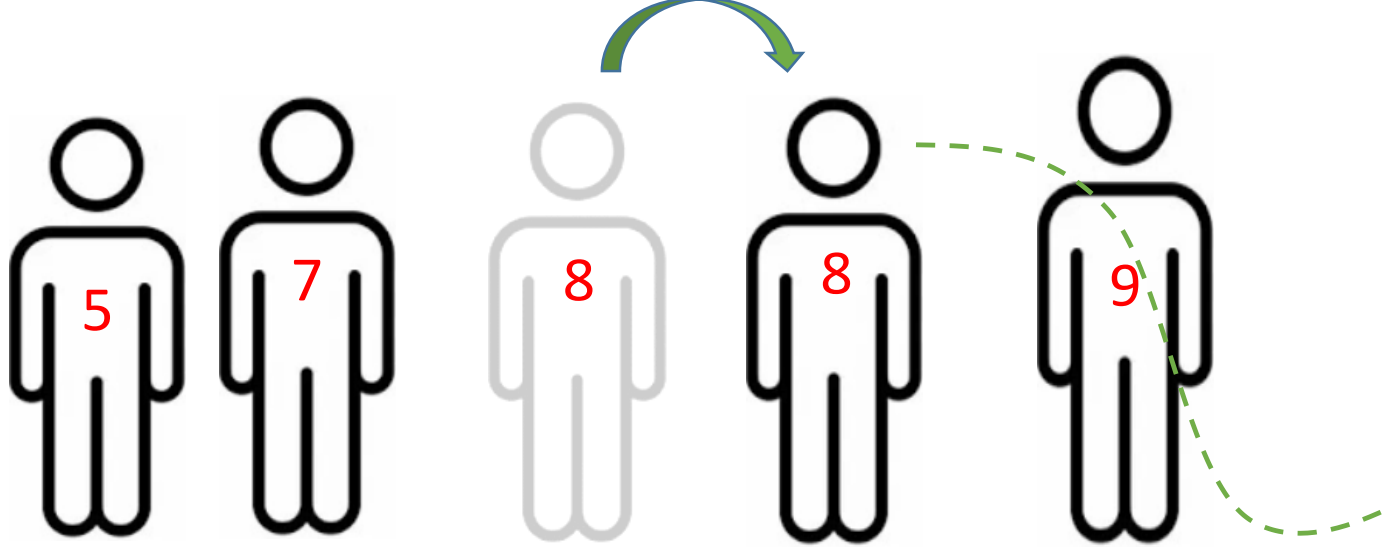


```
// ---- Сортивання включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    let i = k - 1  
    while (i >= 0 && arr[i] > currentElement) {  
      arr[i + 1] = arr[i]  
      i = i - 1  
    }  
    arr[i + 1] = currentElement  
  }  
}
```

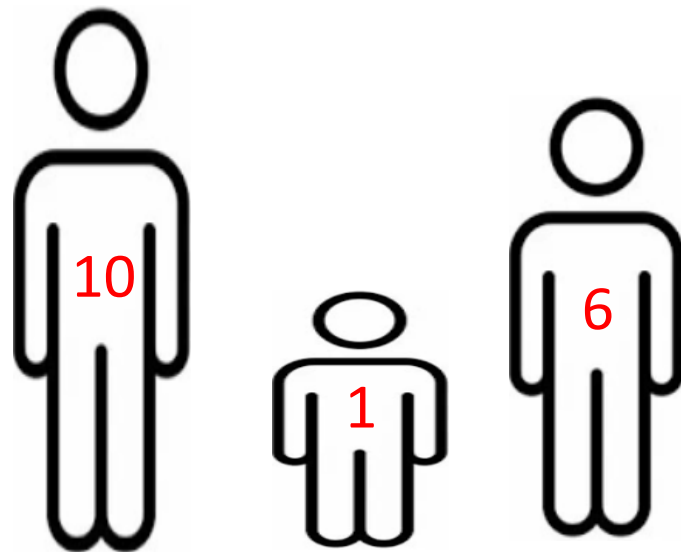


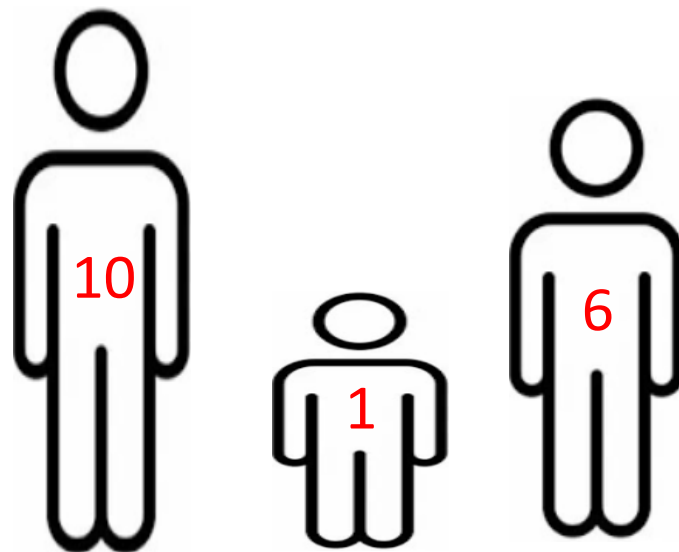
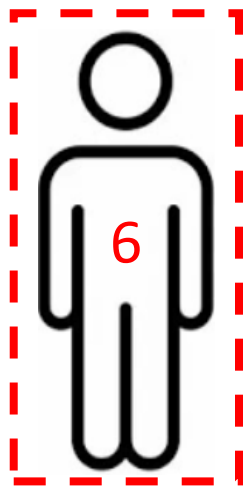
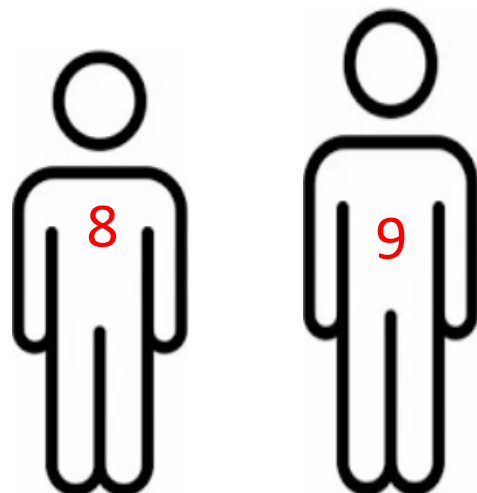
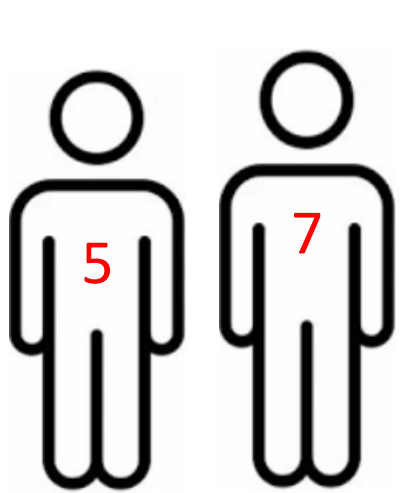
```
//-----Сортування включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    let i = k - 1  
    while (i >= 0 && arr[i] > currentElement) {  
      arr[i + 1] = arr[i]  
      i--  
    }  
    arr[i + 1] = currentElement  
  }  
}
```

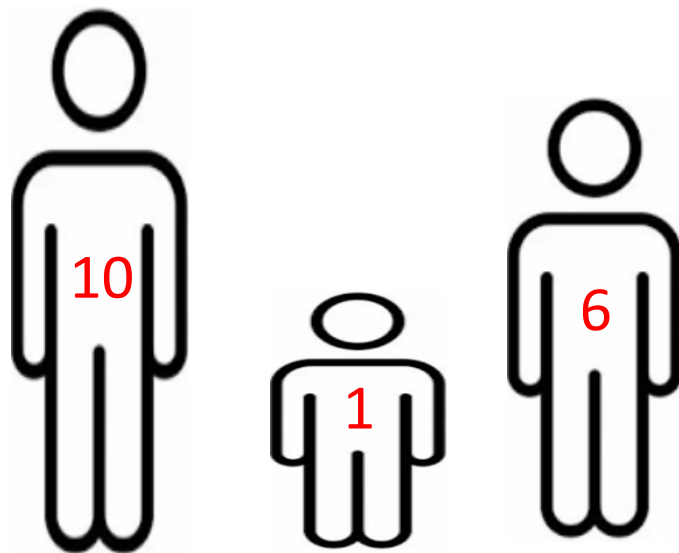
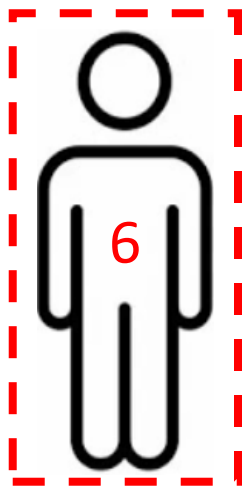
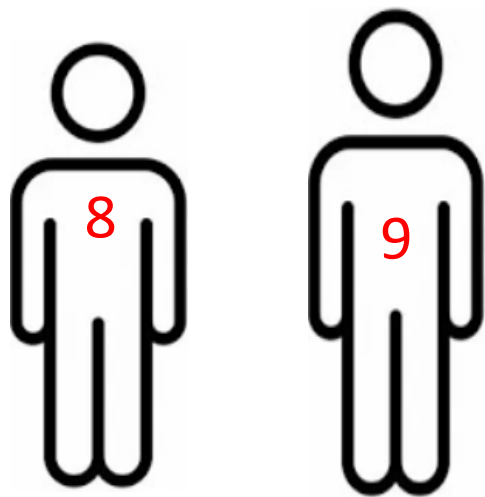
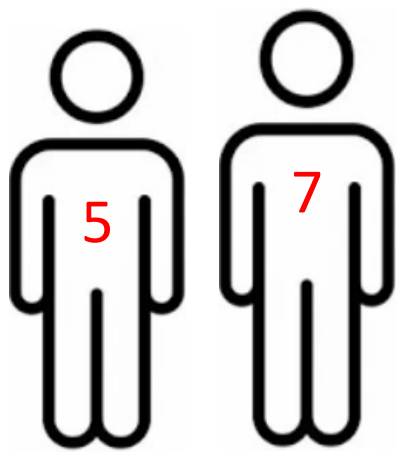


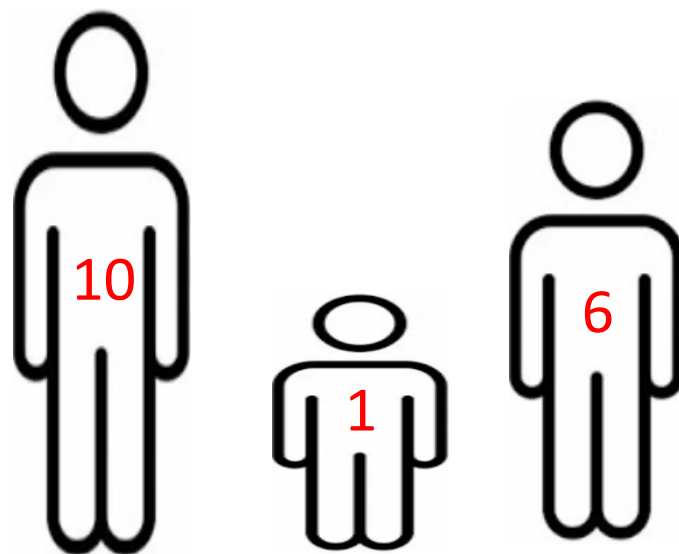
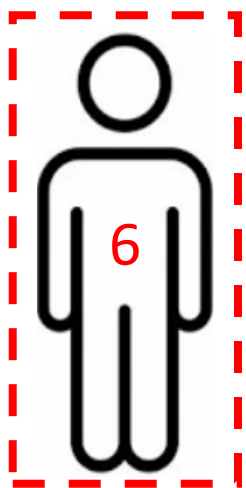
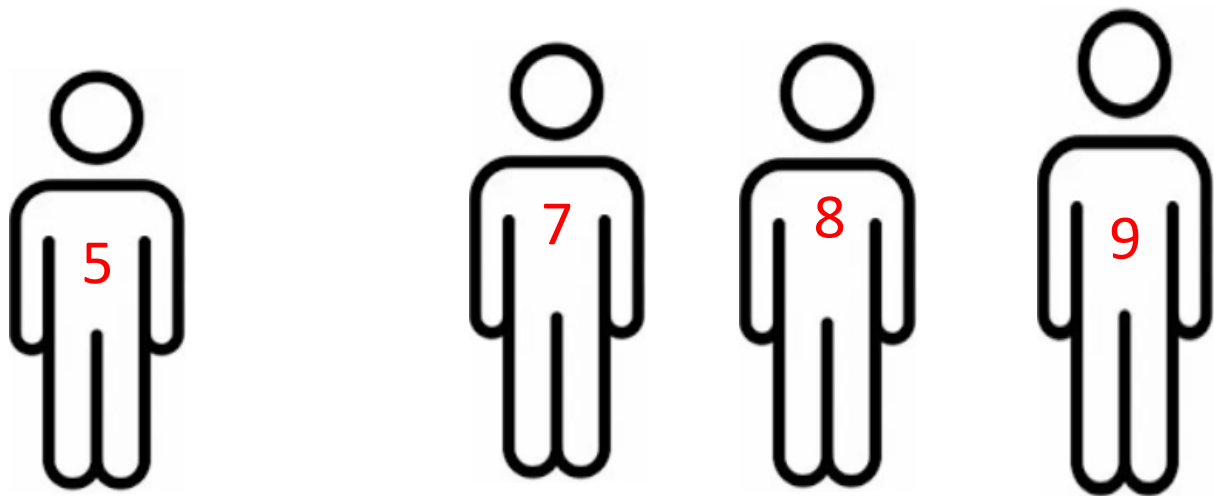


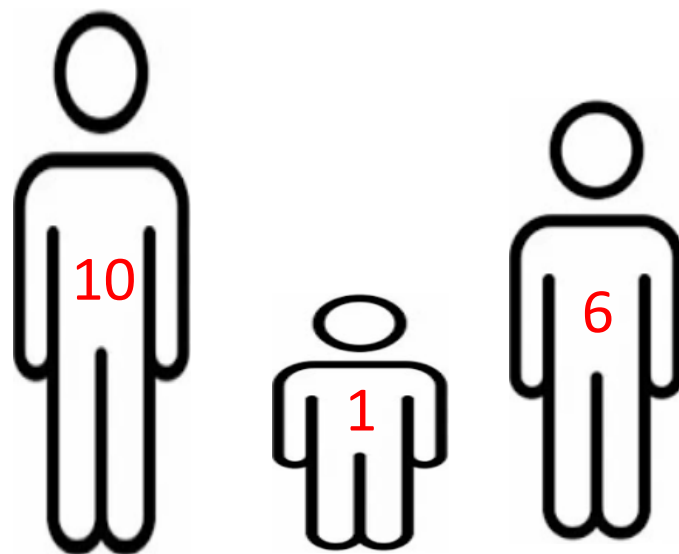
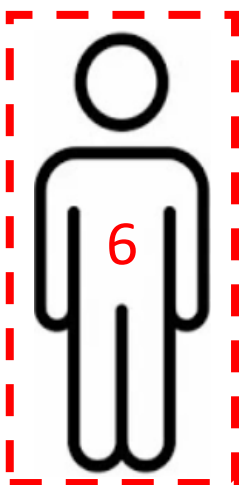
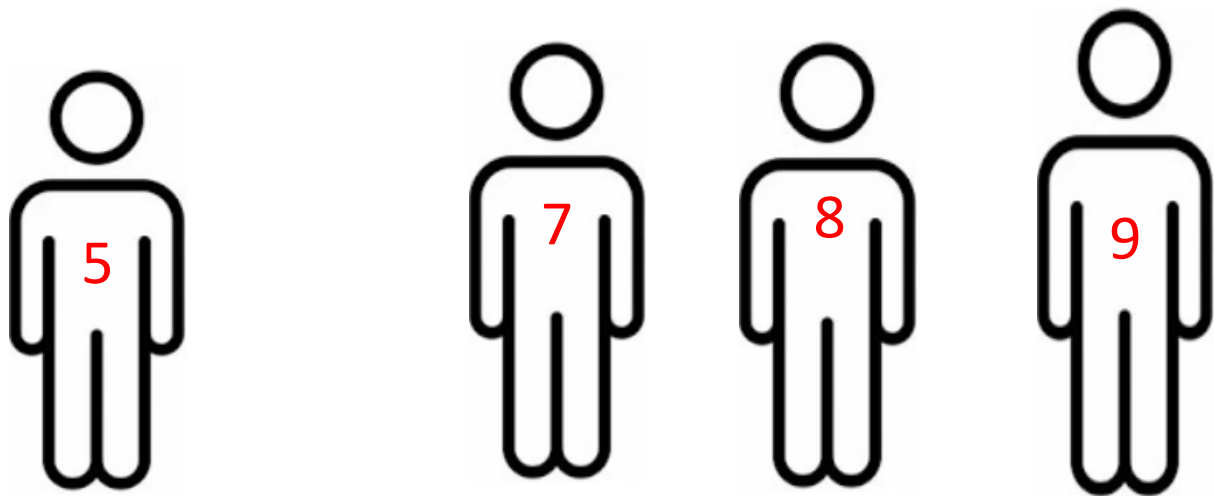
```
// ---- Сортивання включеннями  
function insertSort(arr) {  
  for (let k = 1; k < arr.length; k++) {  
    const currentElement = arr[k]  
    let i = k - 1  
    while (i >= 0 && arr[i] > currentElement) {  
      arr[i + 1] = arr[i]  
      i = i - 1  
    }  
    arr[i + 1] = currentElement  
  }  
}
```



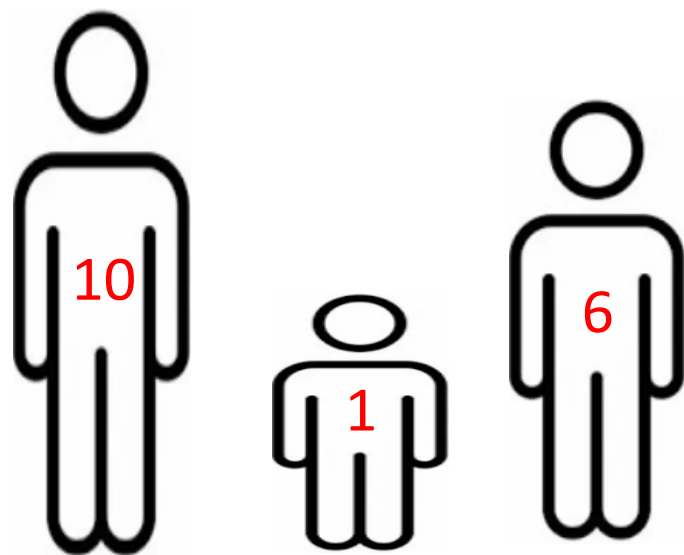
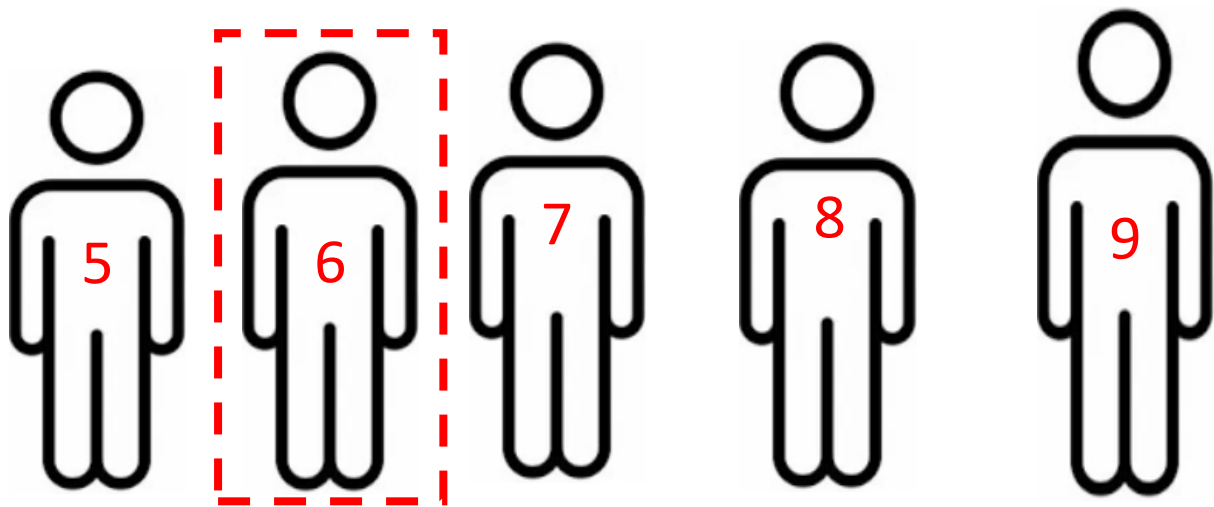


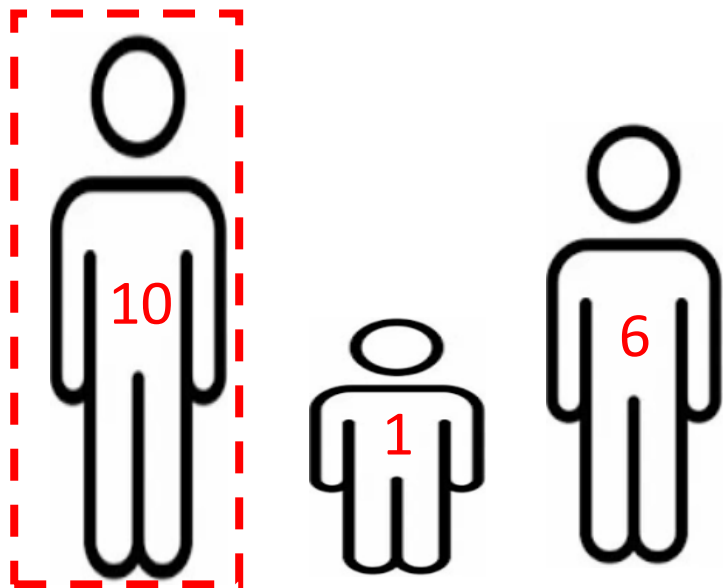
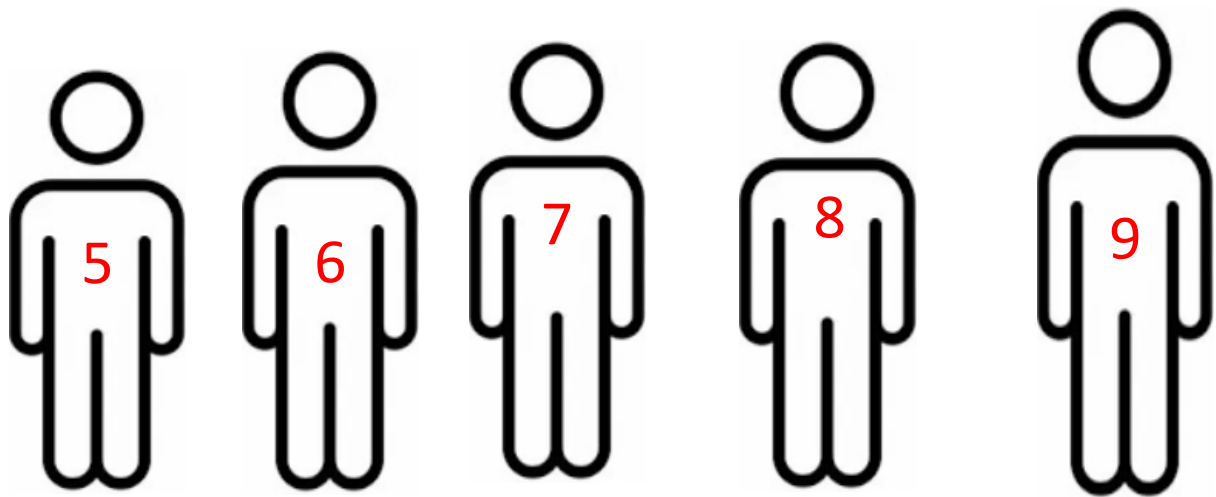


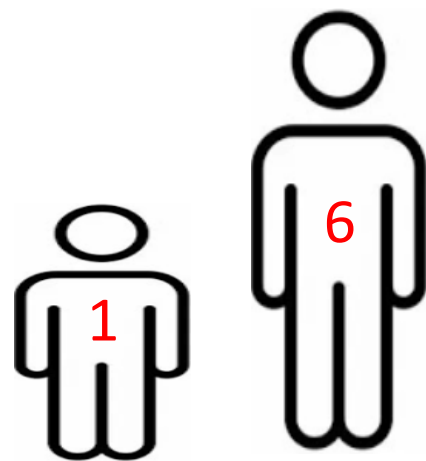
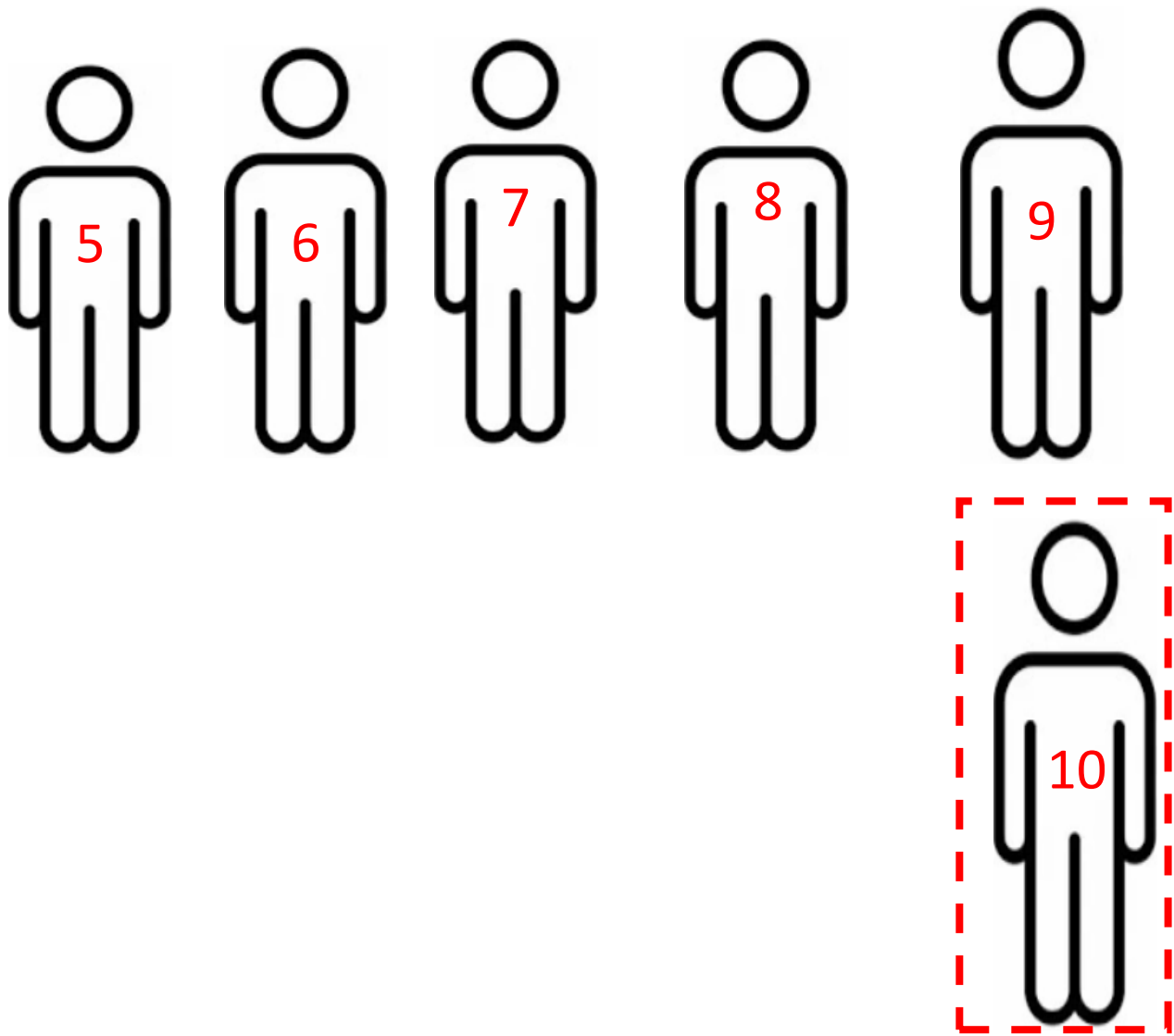


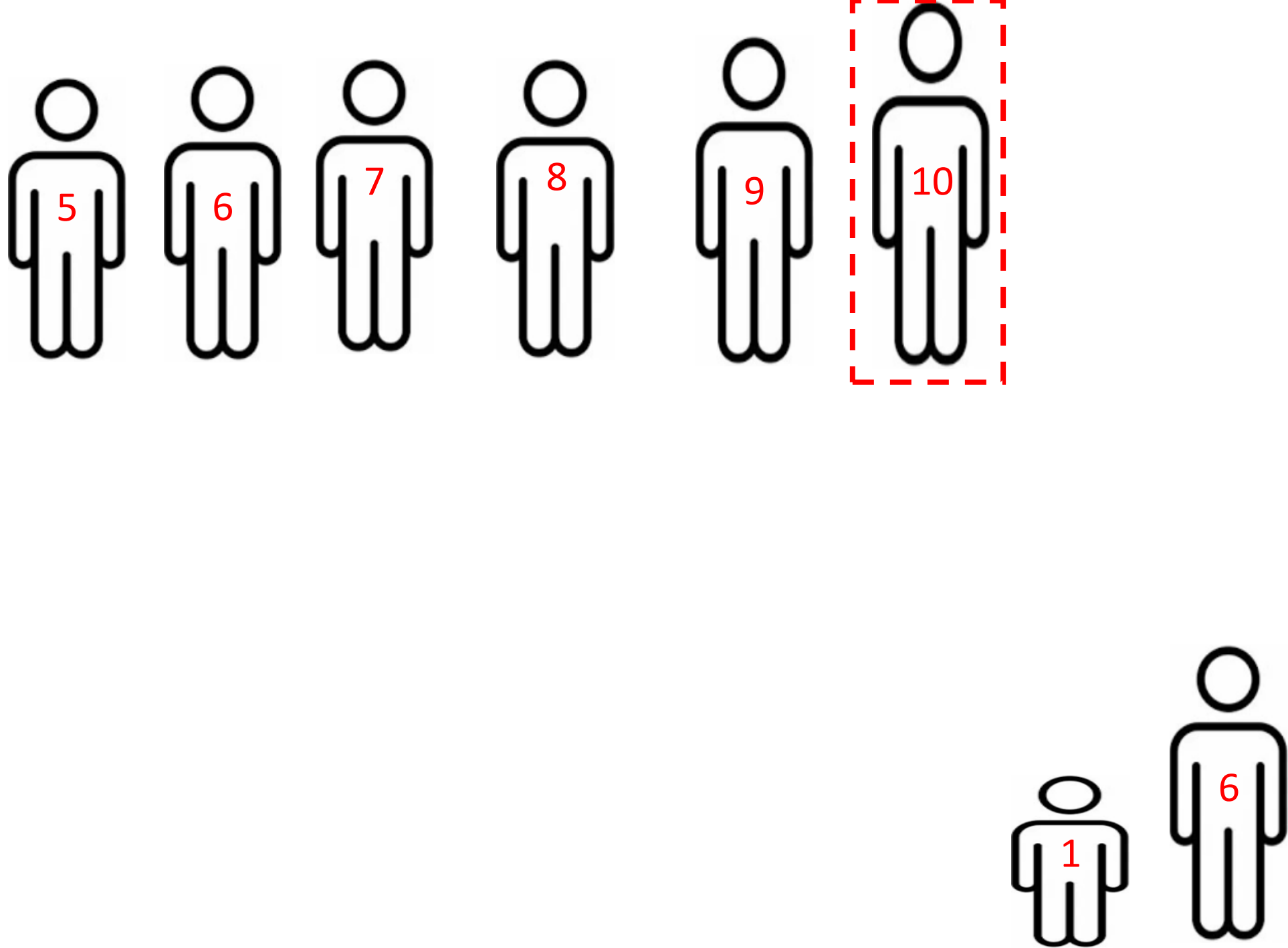


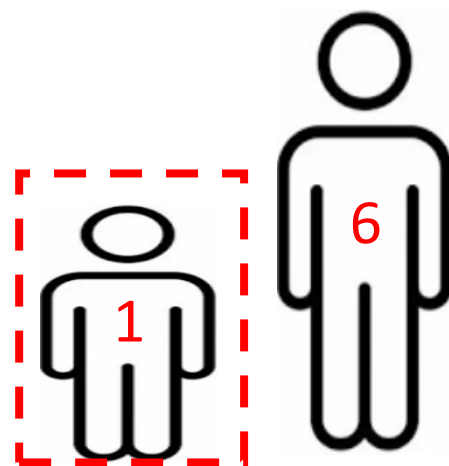
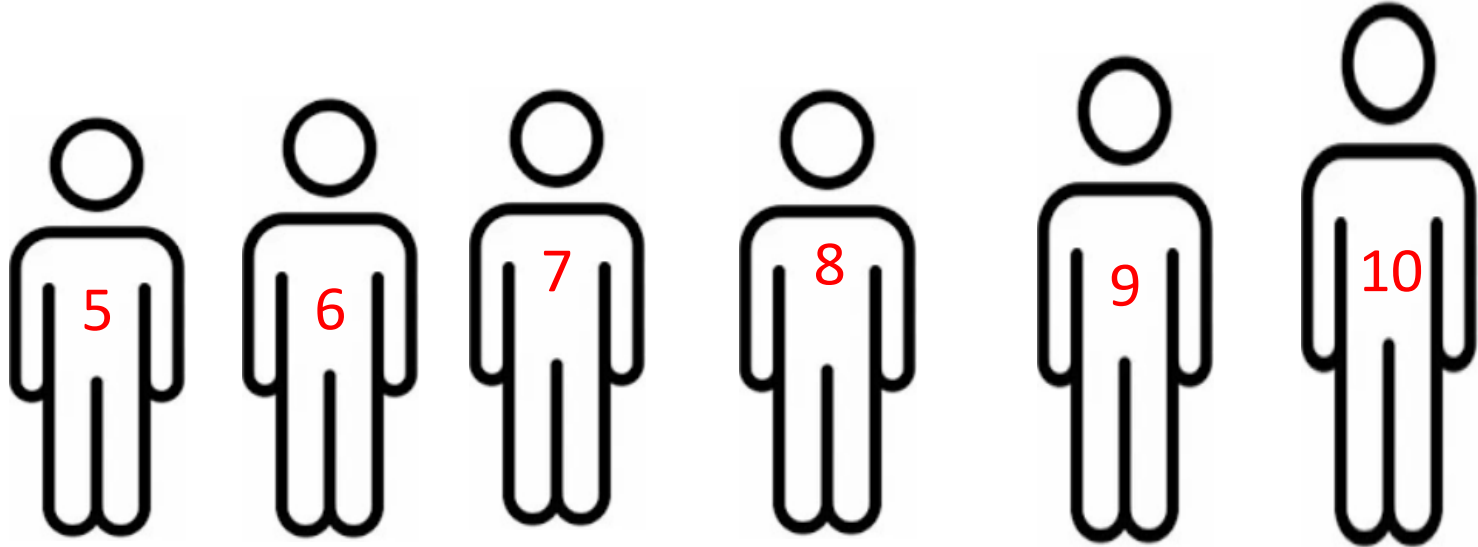


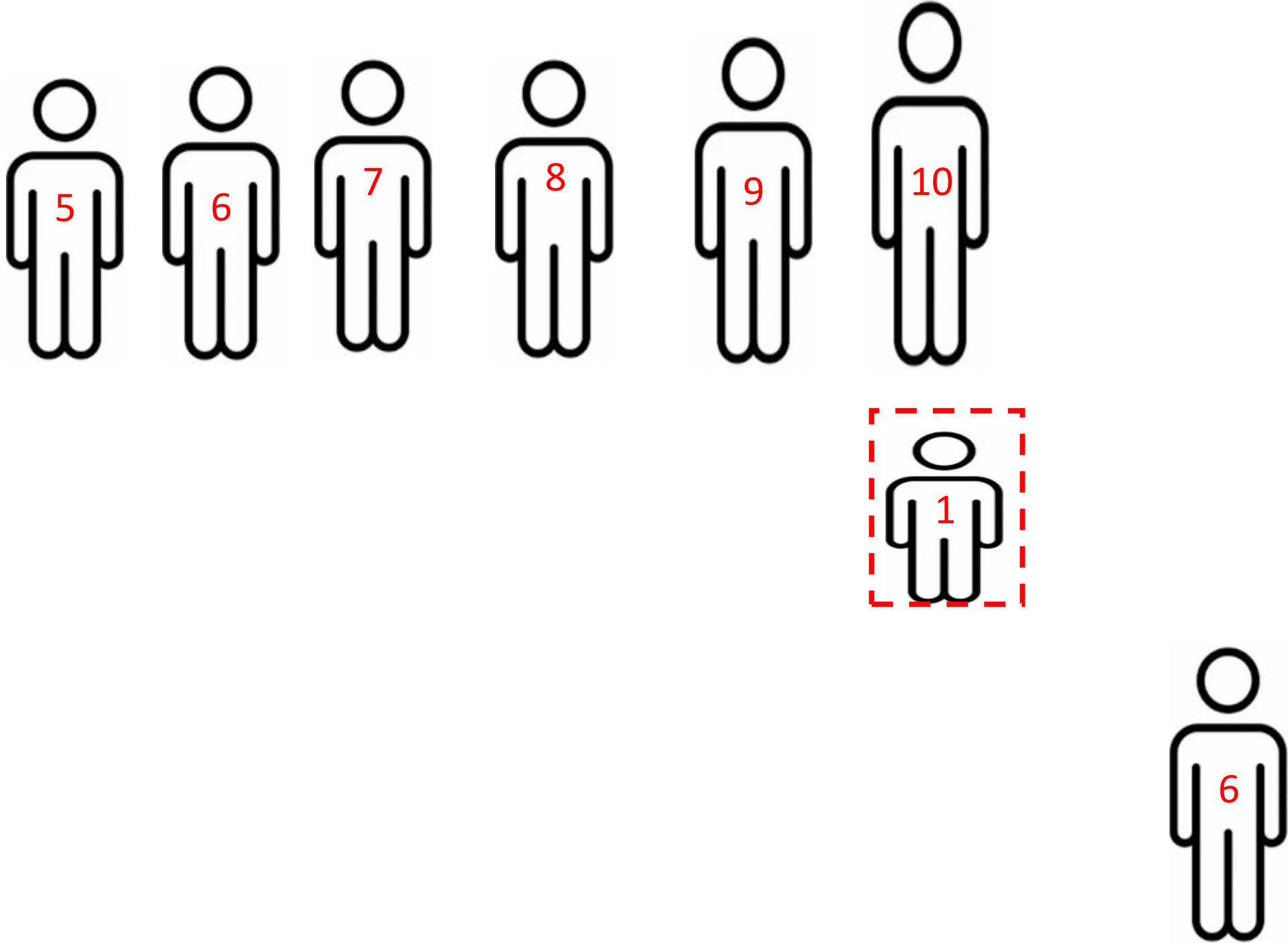


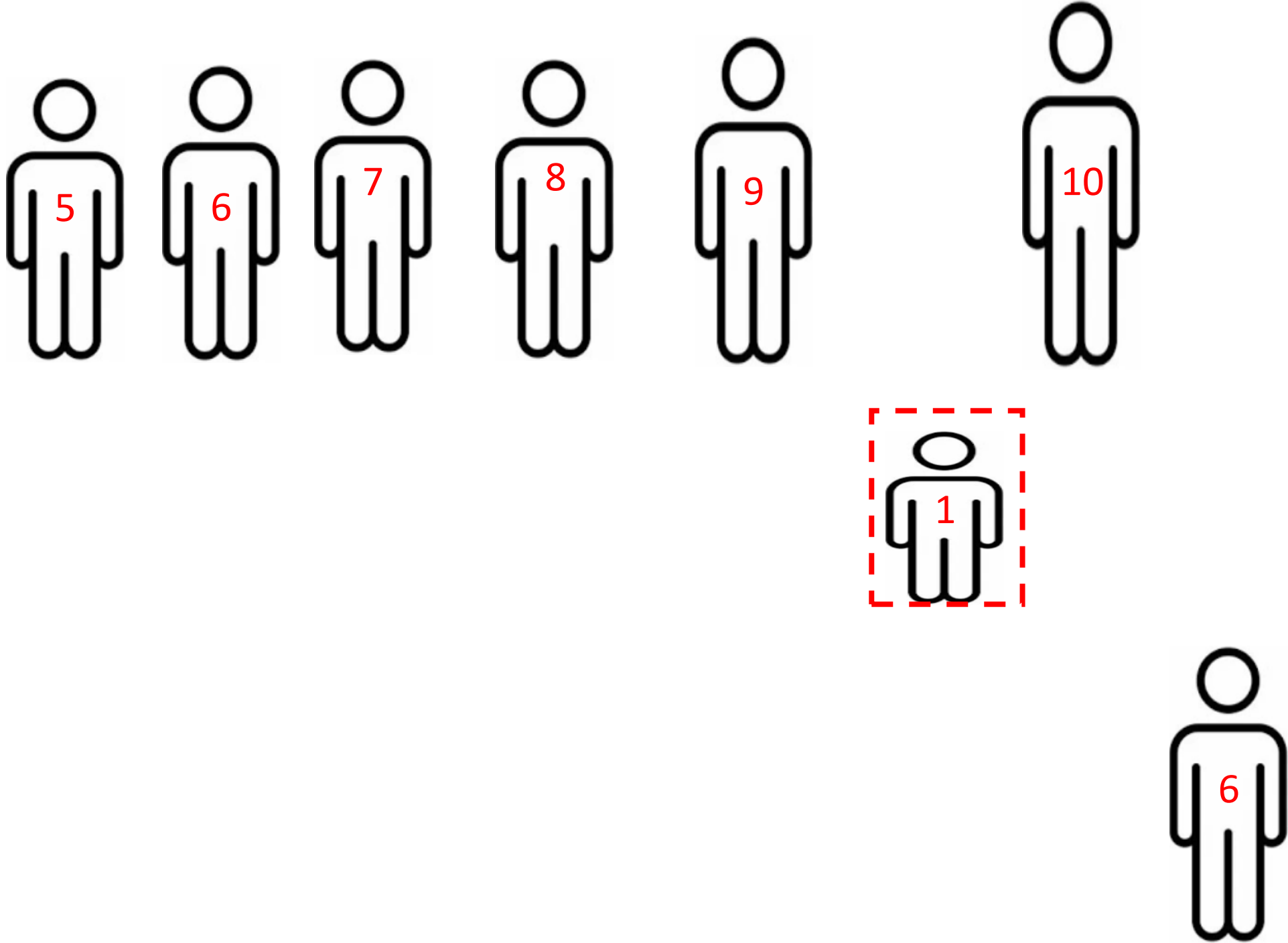


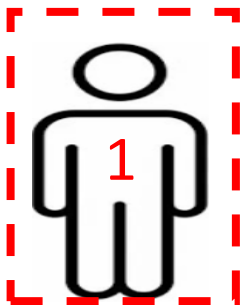
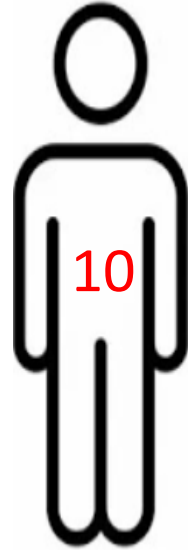
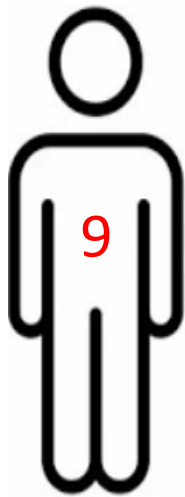
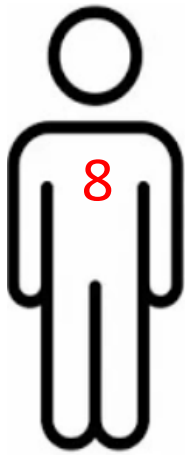
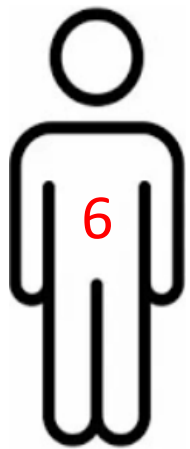
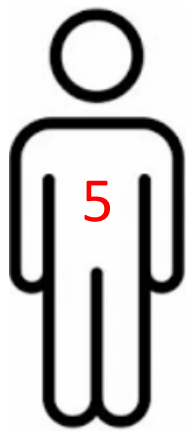




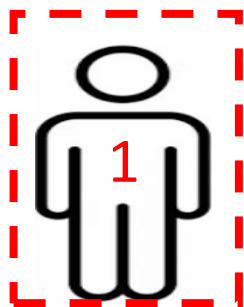
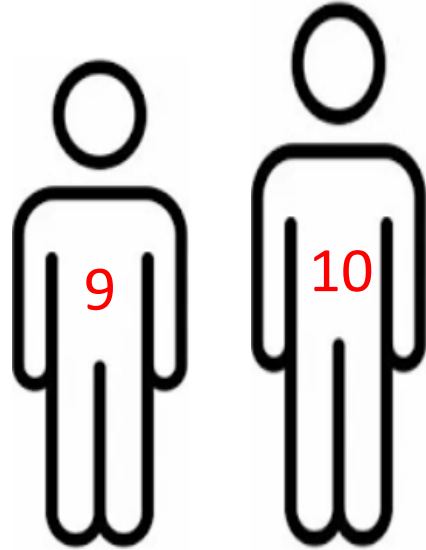
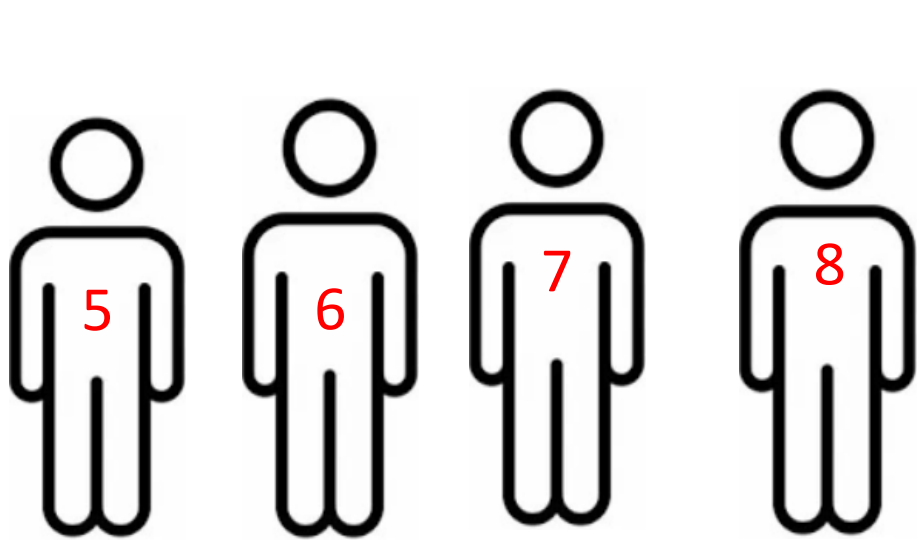


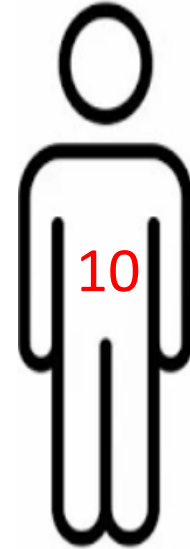
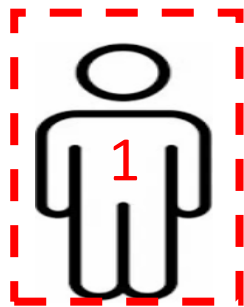
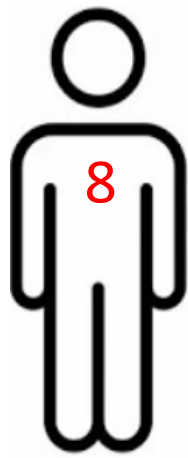
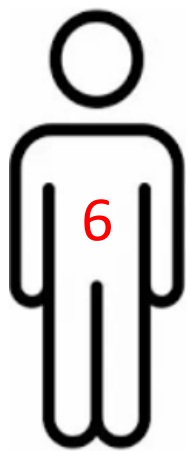
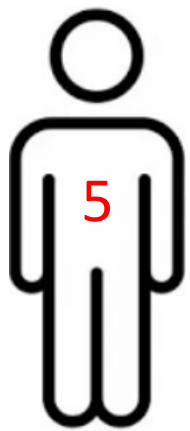


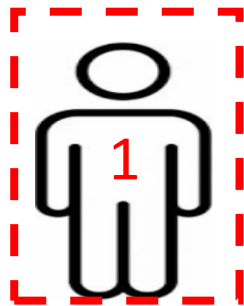
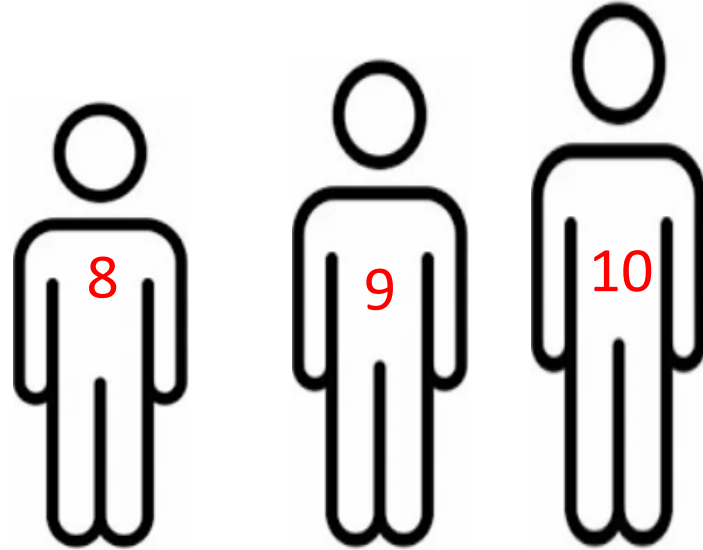
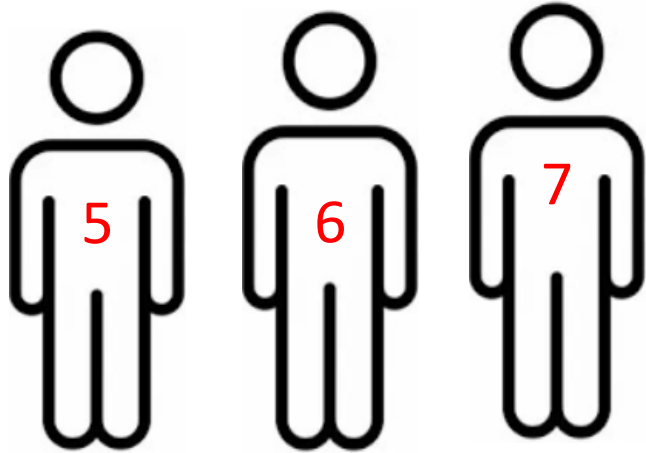


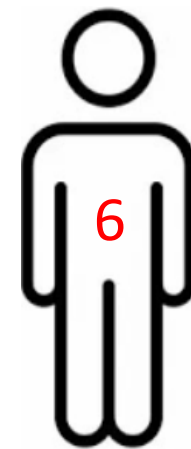
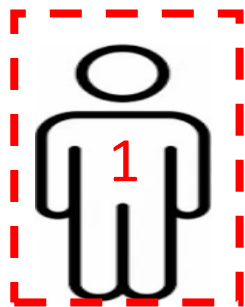
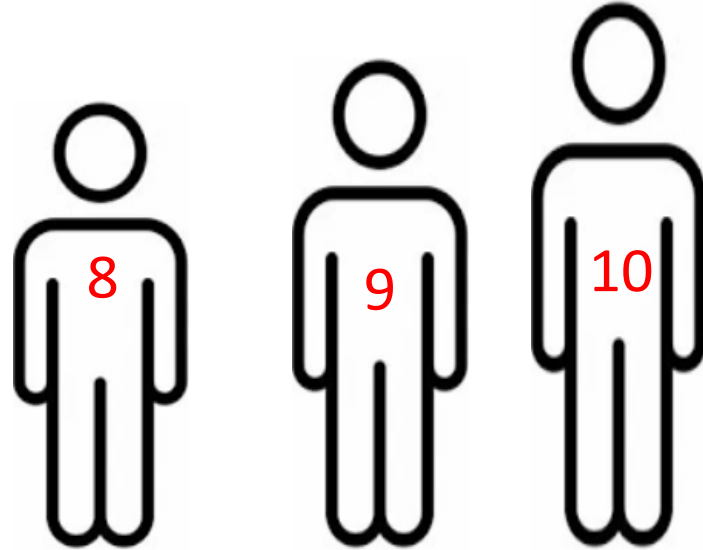
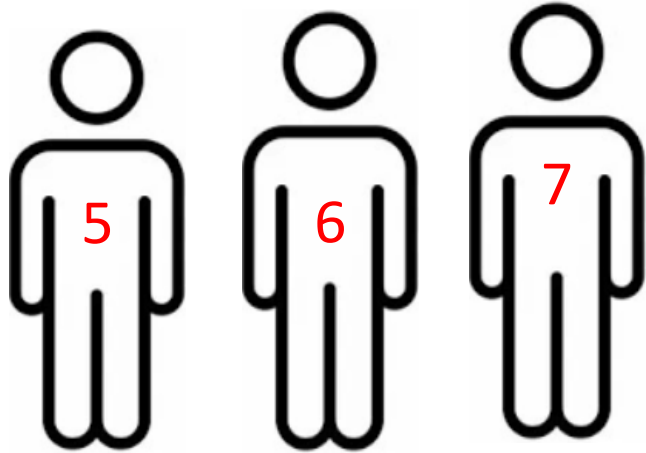


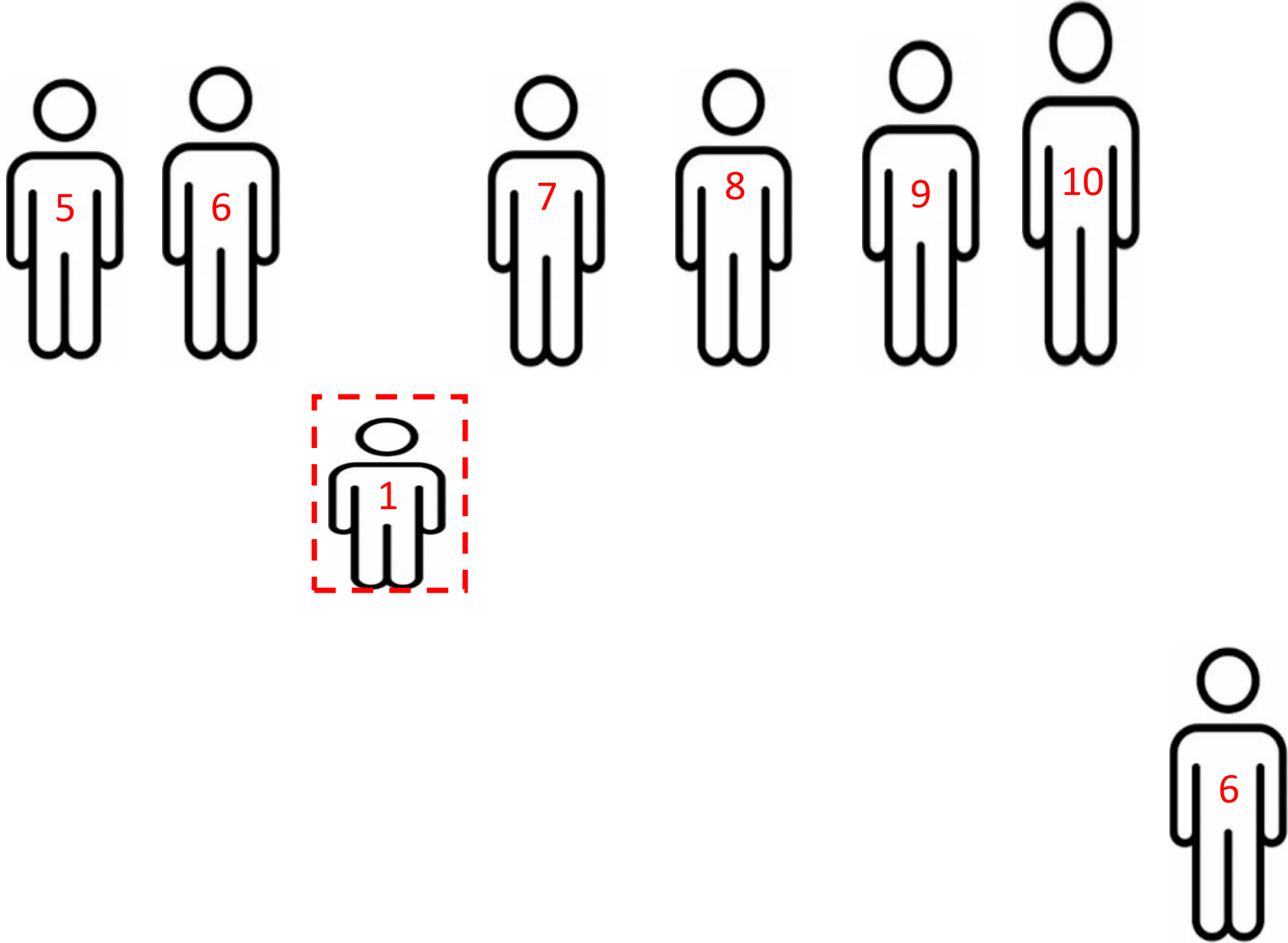


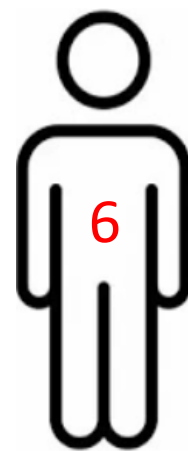
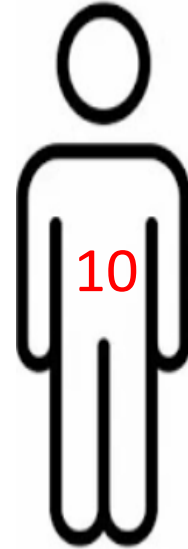
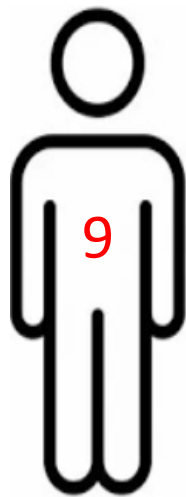
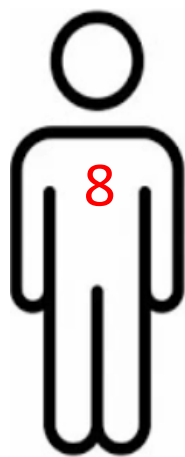
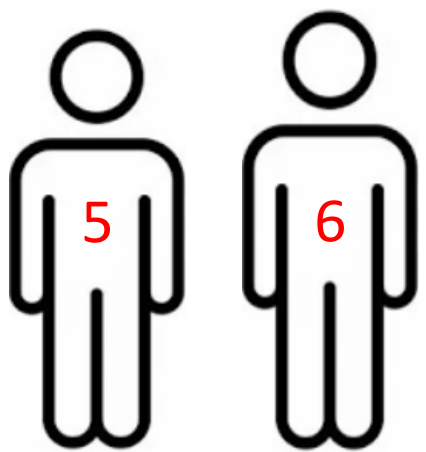


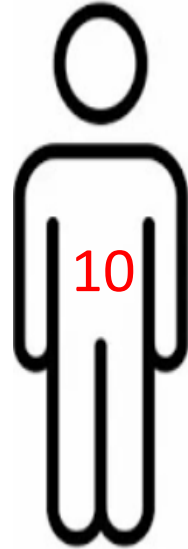
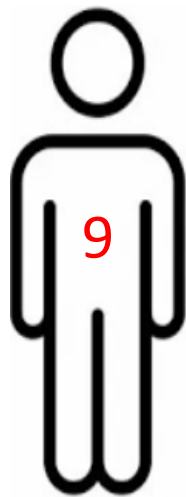
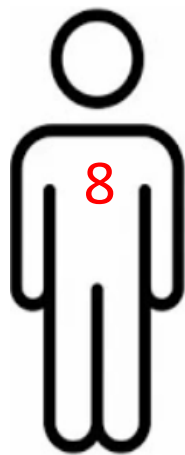
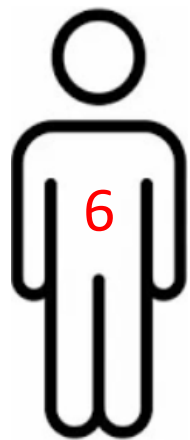
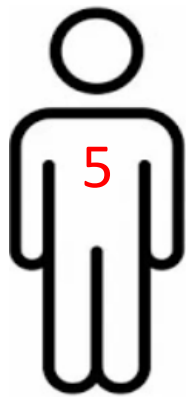


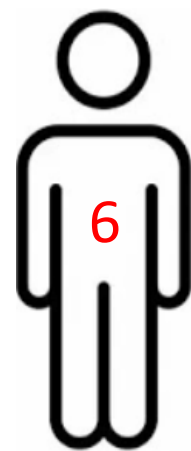
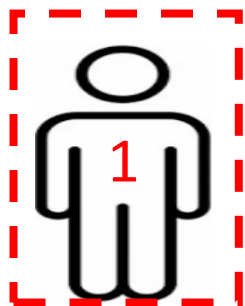
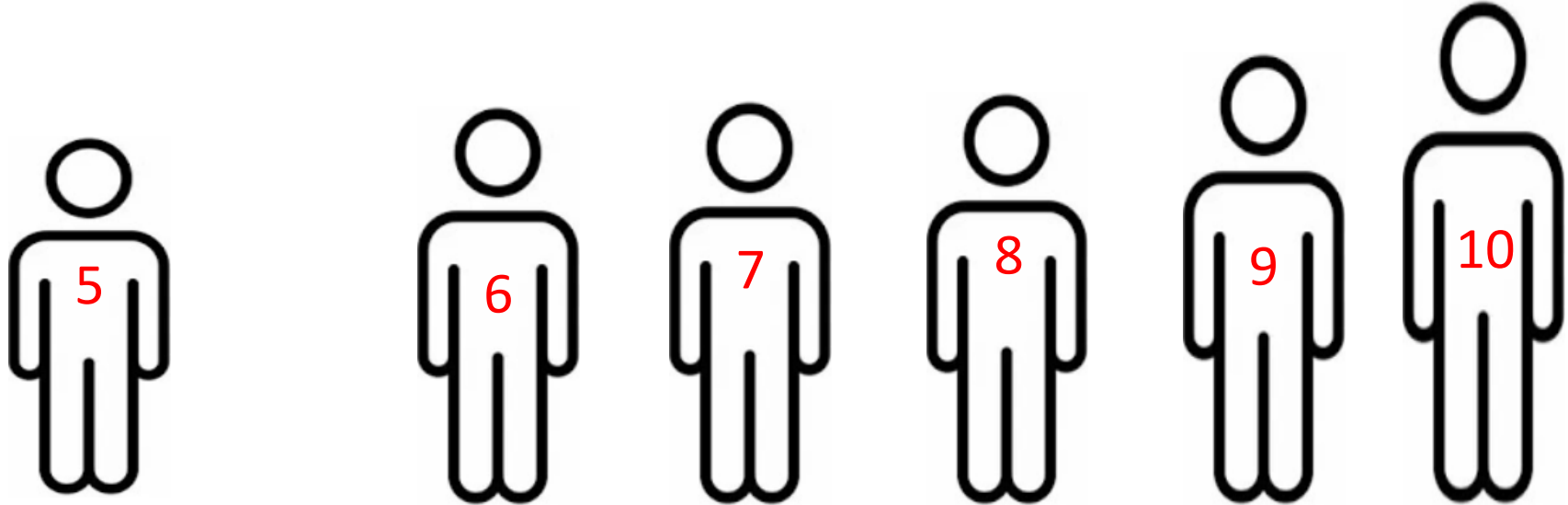




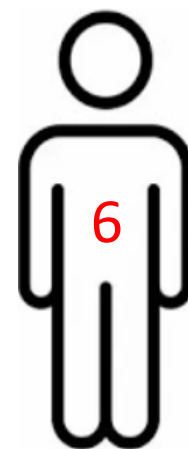
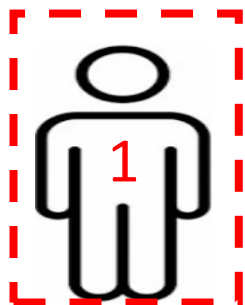
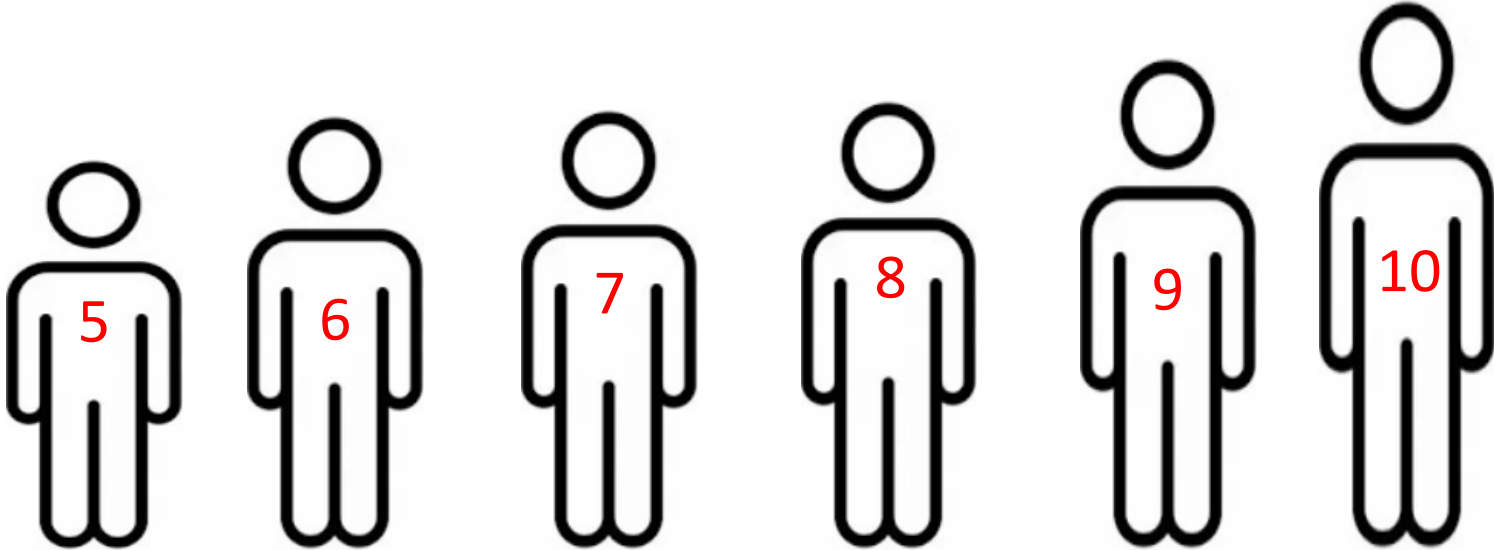


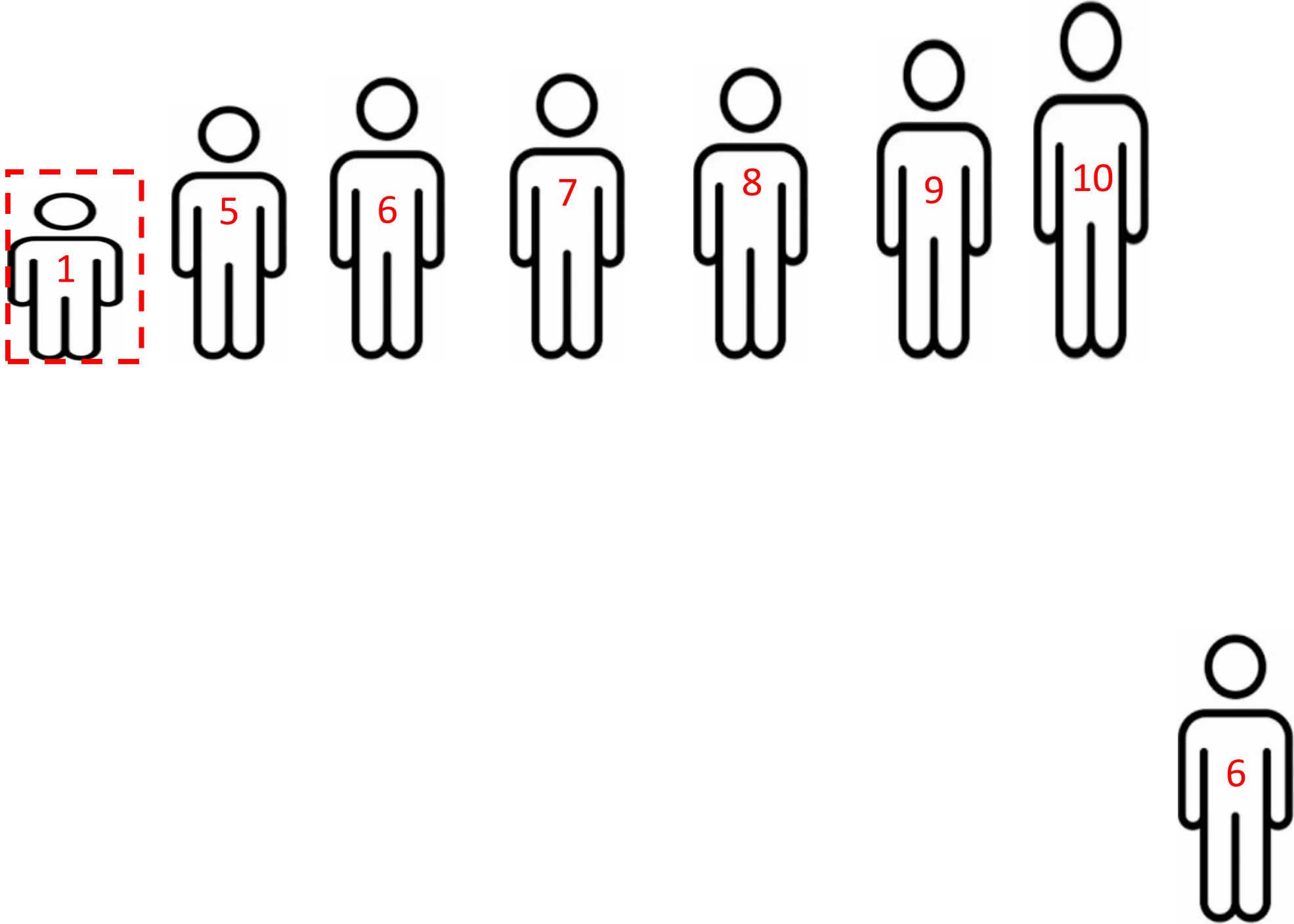


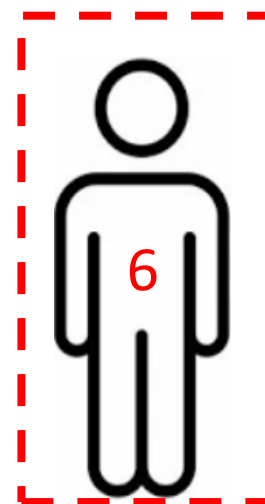
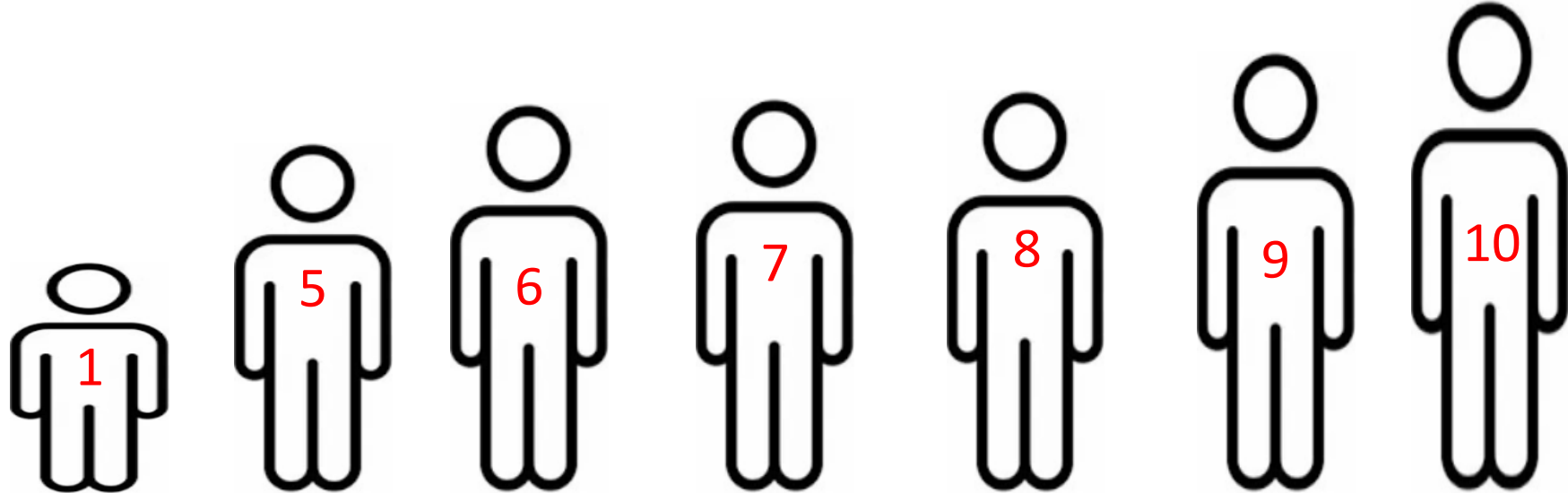


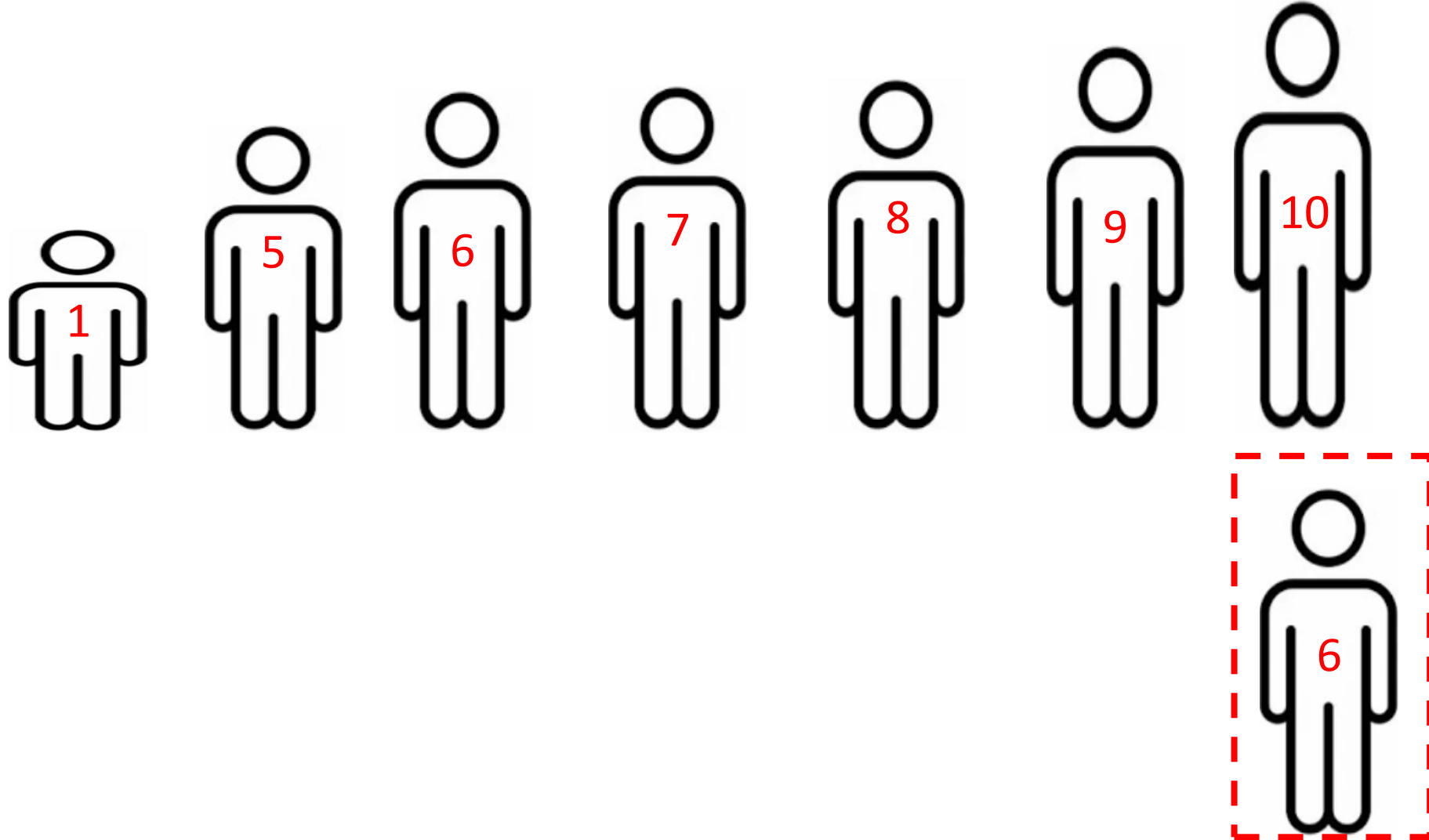


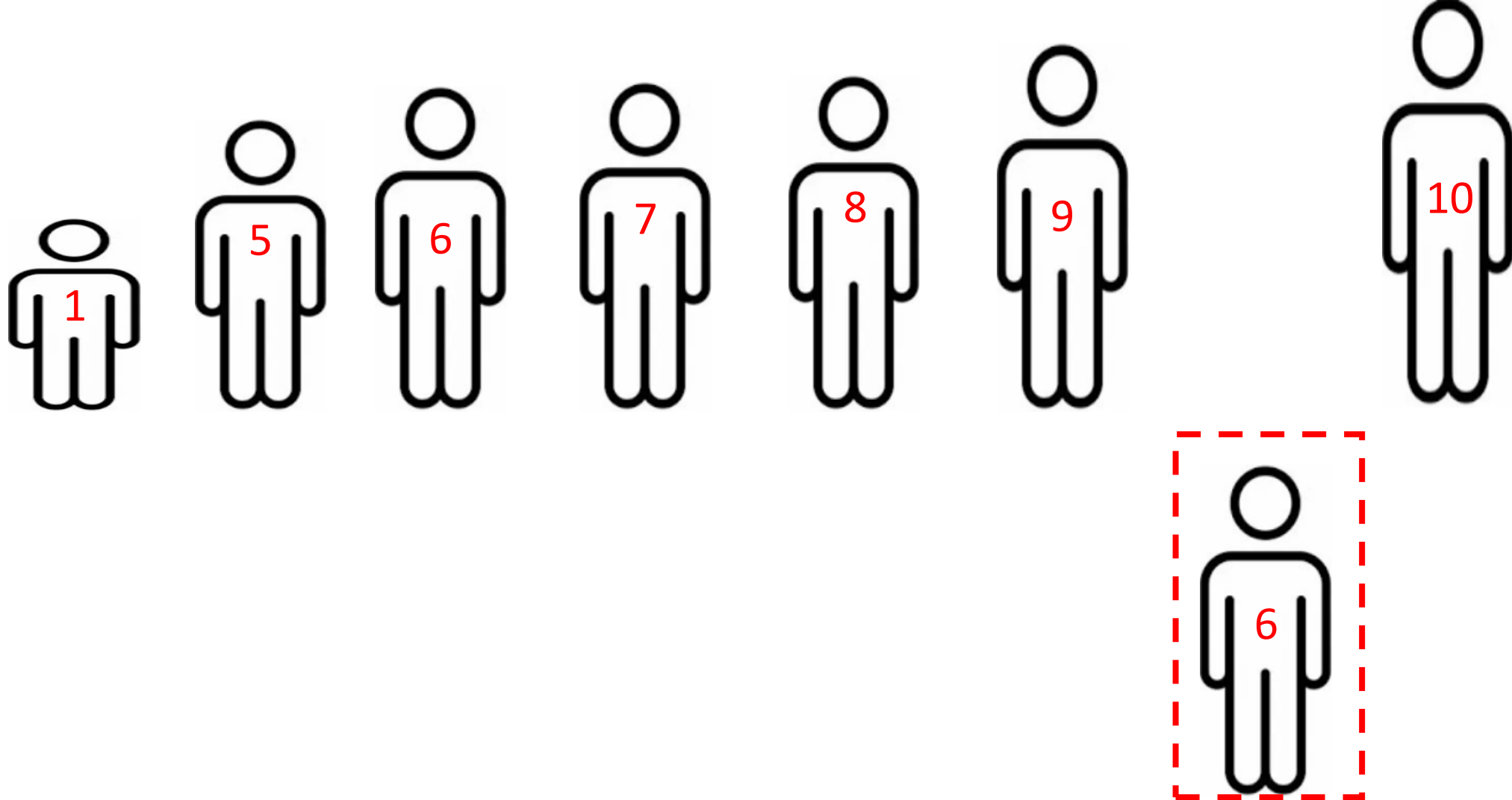


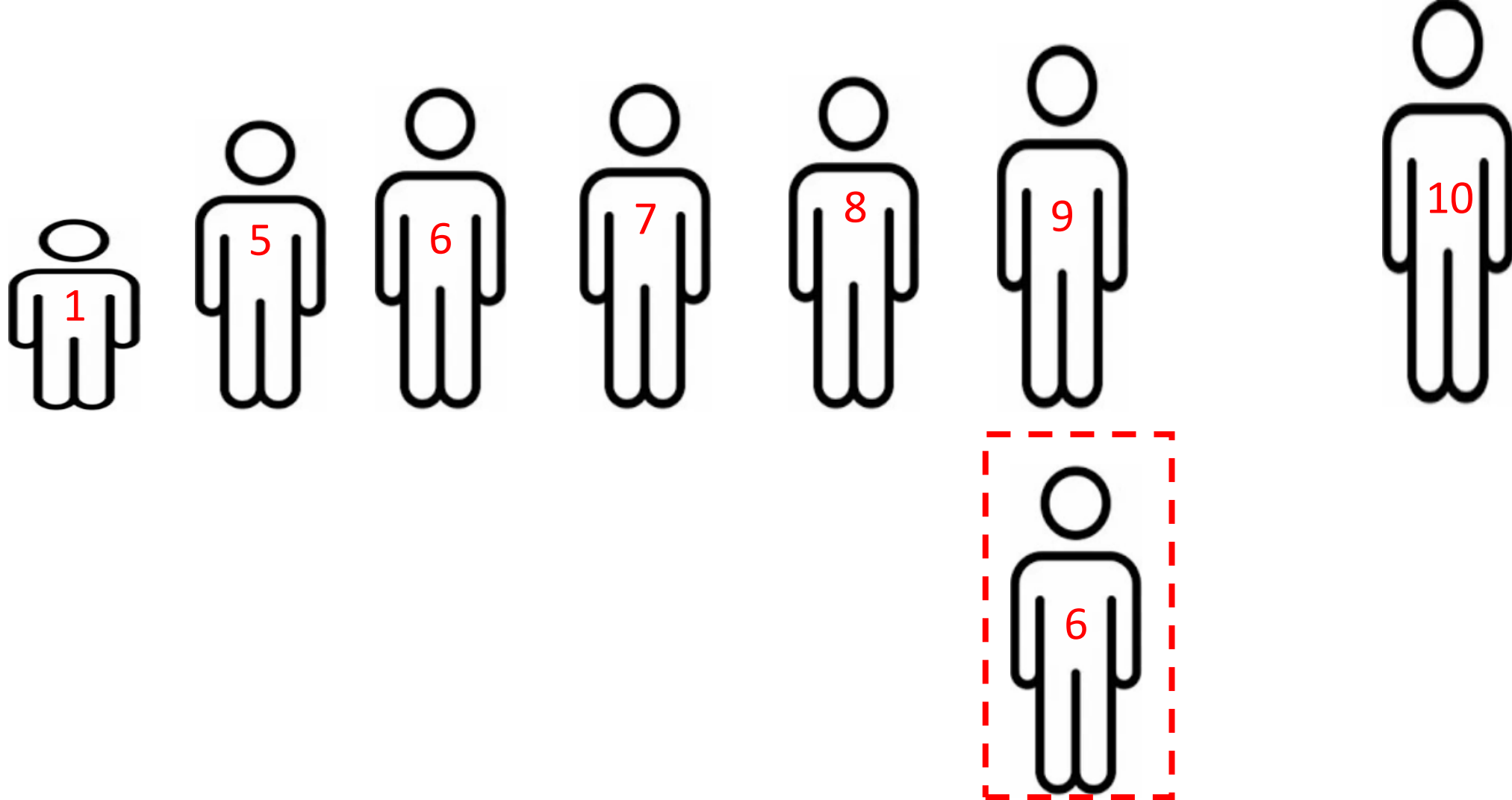


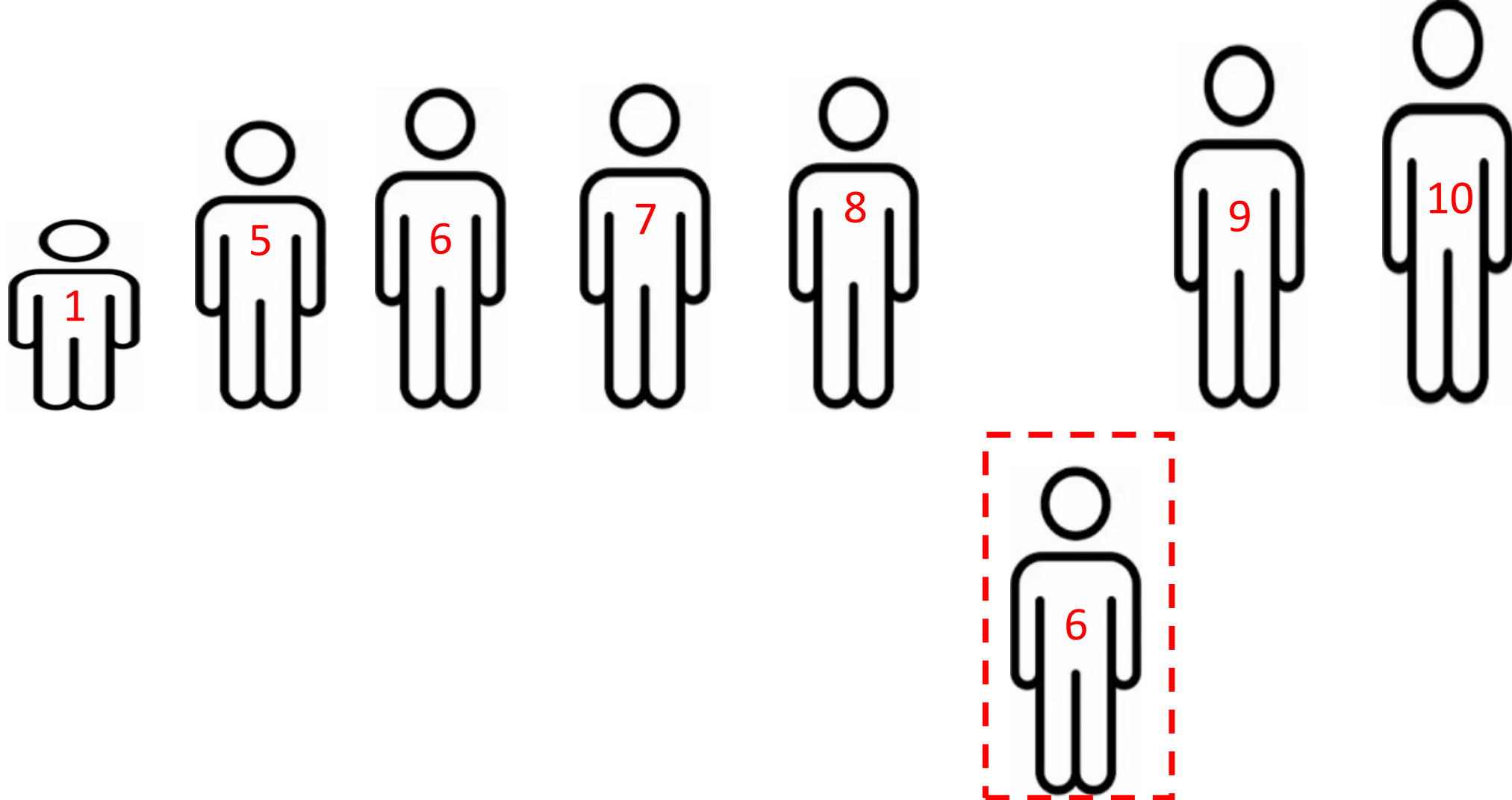


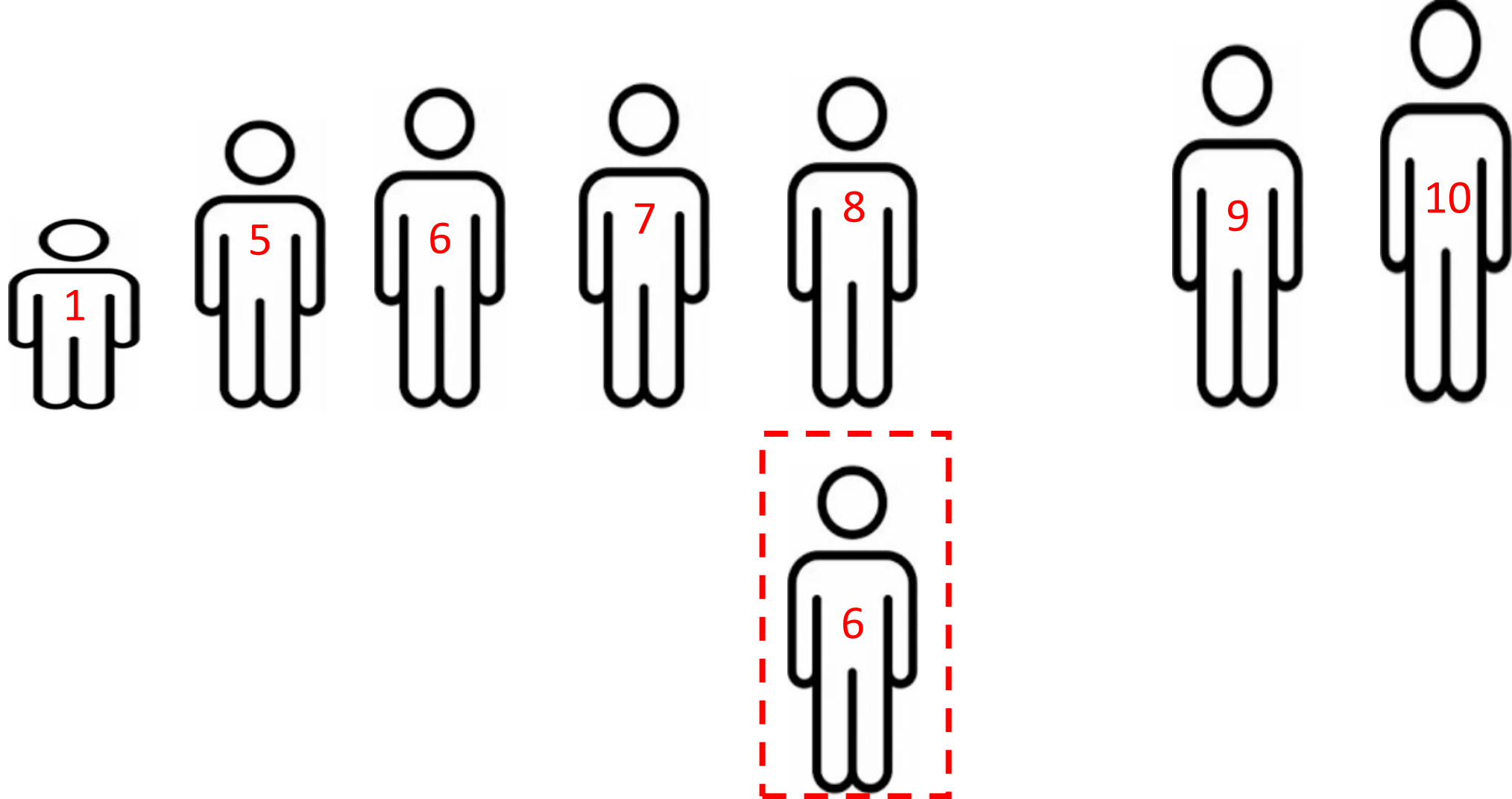




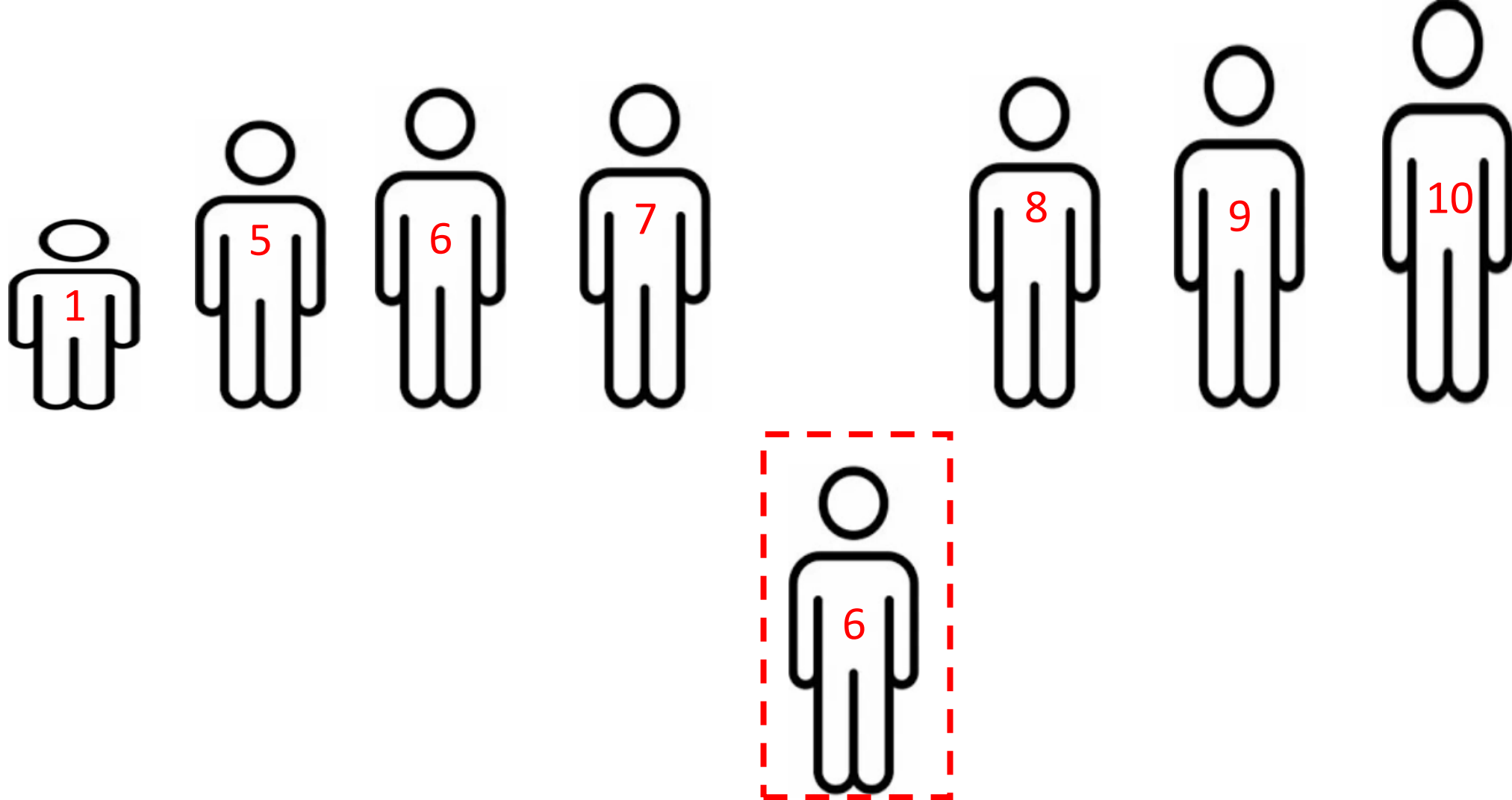


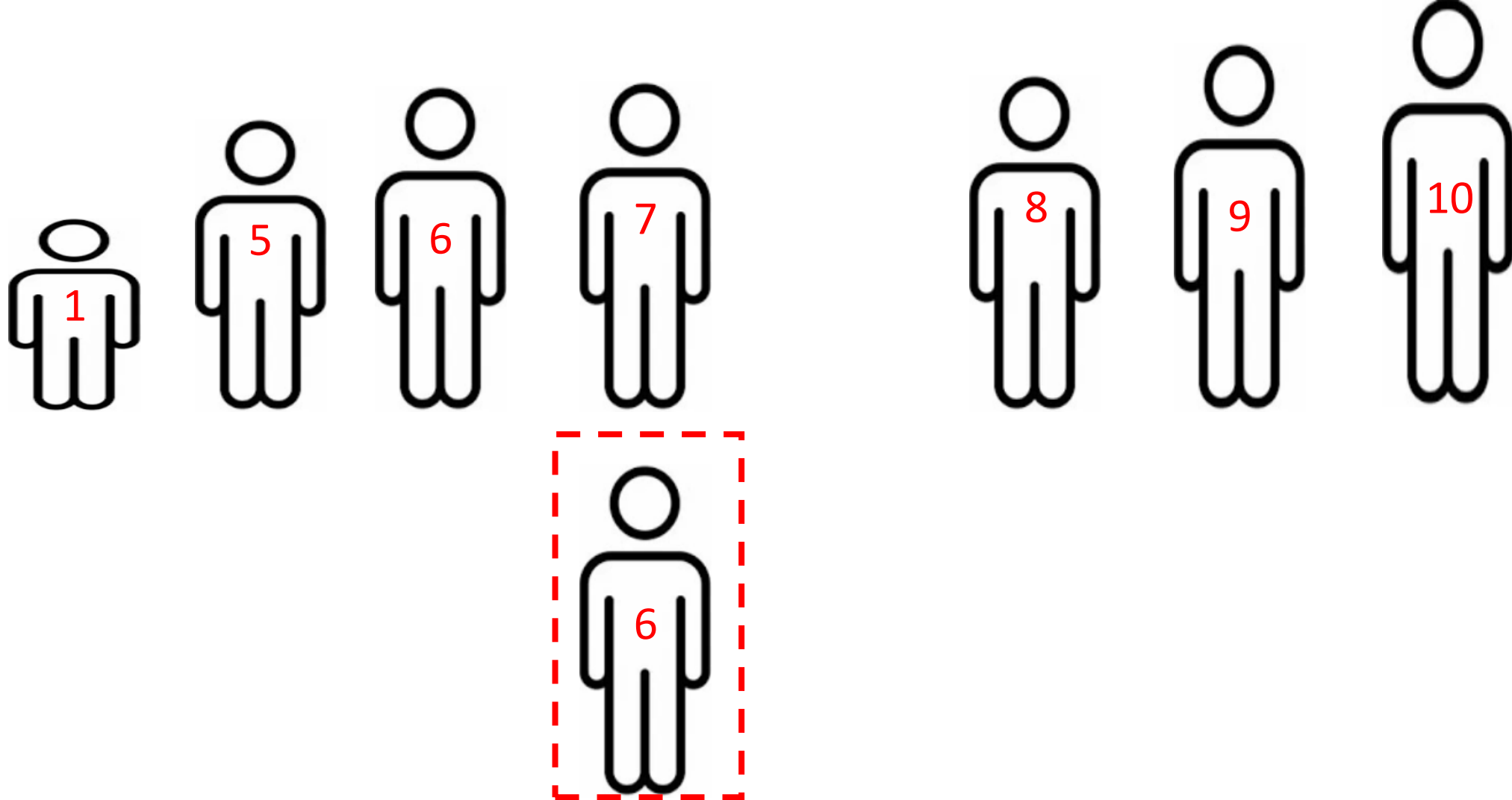


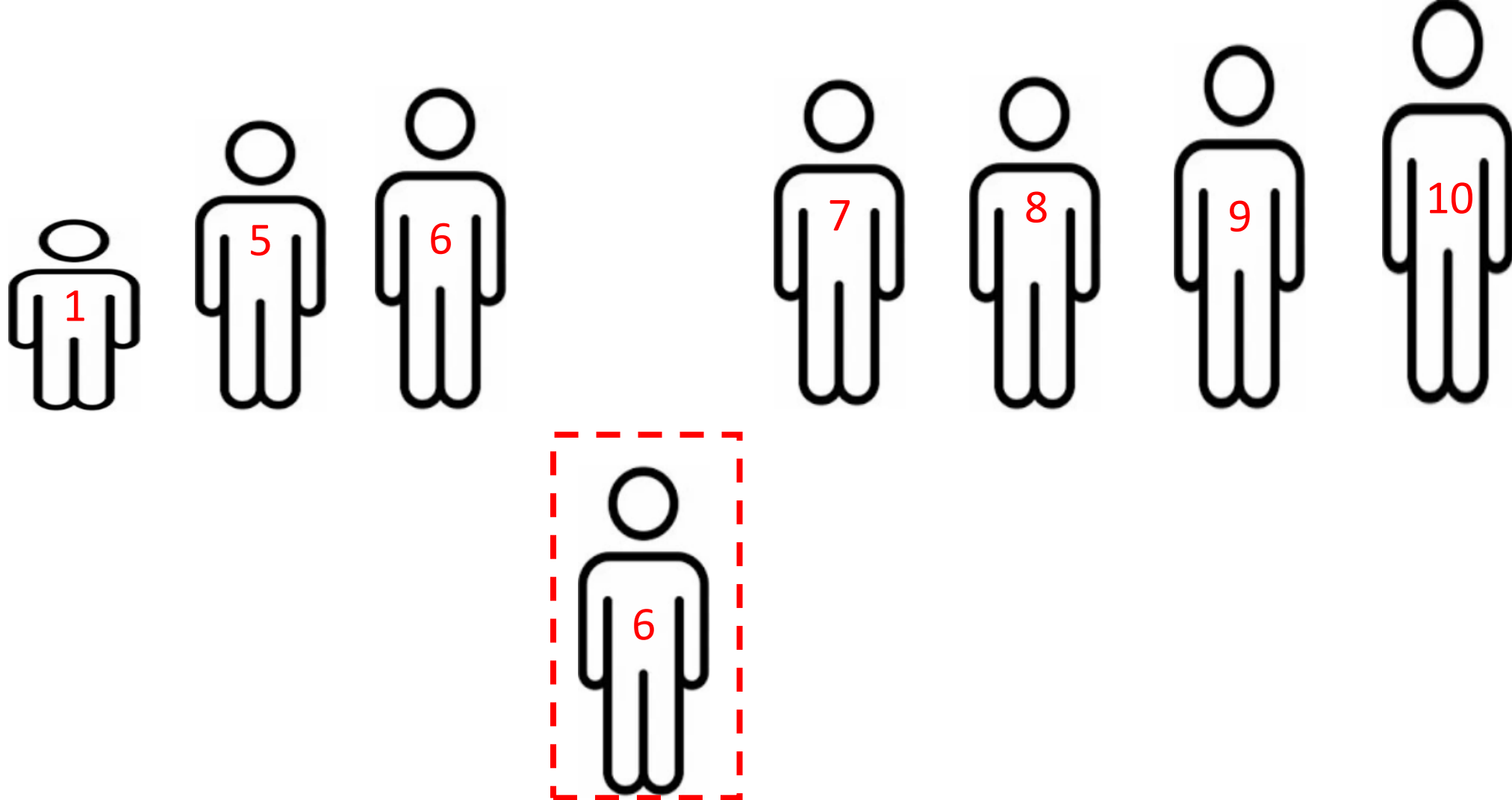


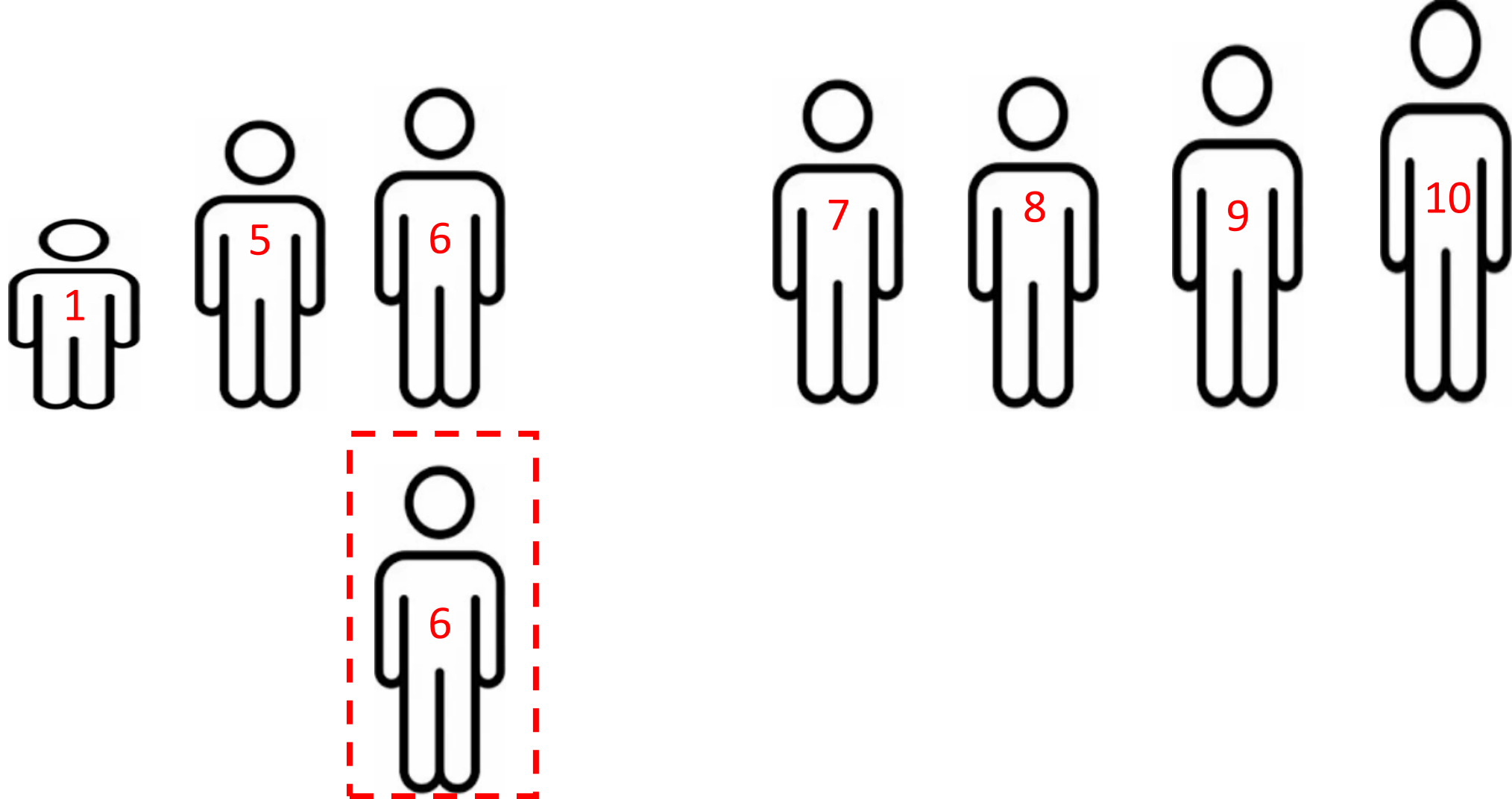


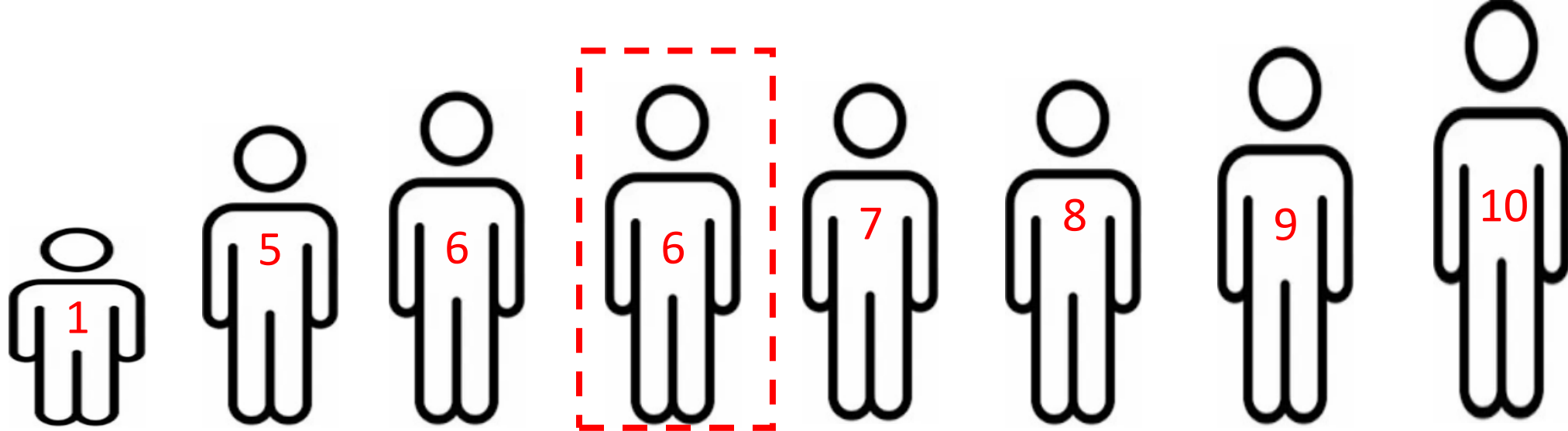












## Сортування включенням

На кожному кроці

- вставляємо  $i$ -вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером K від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//    // перейти до перевірки попереднього елемента (з номером i-1)
//кінєць_поки
//вставляєм поточне значення "key" після елемента,
//який вже не був більшим (або дійшли до початку масиву)
//кінєць_для
```

[https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F\\_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC)

## Сортування включенням

# На кожному кроці

- вставляємо  $i$ -вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням.

ht

%

701

```
// кінець для
```

```
let key, i
```

## Сортування включенням

На кожному кроці

- вставляємо  $i$ -вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером K від 1 до останнього
```

```

// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  // ...
}
```



## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером K від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"

// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  // ...
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"

// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  key = arr[k]
  i = k
  while (i > 0 && arr[i-1] > key) {
    arr[i] = arr[i-1]
    i--
  }
  arr[i] = key
}
```

## Сортування включенням

На кожному кроці

- вставляємо  $i$ -вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером K від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  key = arr[k]
  // ...
}
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
...
...
...
// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  key = arr[k]
  i = k - 1
  ...
  ...
  ...
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//кінєць_поки
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  key = arr[k]
  i = k - 1
  while (i > 0 && arr[i] > key) {
    arr[i + 1] = arr[i]
    i--
  }
  arr[i + 1] = key
}
```

## Сортування включенням

На кожному кроці

- вставляємо  $i$ -вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")

//кінець_поки

// кінець_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
  key = arr[k]
  i = k - 1
  while (i >= 0 && arr[i] > key) {
    // ...
  }
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//
//кінєць_поки
//
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        // shift elements to the right
    }
    arr[i+1] = key
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//
//кінєць_поки
//
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        arr[i + 1] = arr[i]
    }
}
```



## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//    // перейти до перевірки попереднього елемента (з номером i-1)
//кінєць_поки
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        arr[i + 1] = arr[i]
    }
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//    // перейти до перевірки попереднього елемента (з номером i-1)
//кінєць_поки
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        arr[i + 1] = arr[i]
        i = i - 1
    }
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//    // перейти до перевірки попереднього елемента (з номером i-1)
//кінєць_поки
//вставляєм поточне значення "key" після елемента,
//який вже не був більшим (або дійшли до початку масиву)
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        arr[i + 1] = arr[i]
        i = i - 1
    }
}
```

## Сортування включенням

На кожному кроці

- вставляємо і-вий елемент у правильну позицію зліва;
- при цьому вважаємо, що всі елементи, що розташовані лівіше упорядковані за зростанням

```
// для кожного елемента з номером k від 1 до останнього
//----- поставити k-вий елемент у правильну позицію -----
//запам'ятати "k"-товий елемент як поточне значення у "key"
//починаємо перегляд "i"-тових елементів з попередньої до "k"
//поки (номер "i" коректний та i-вий елемент більший за "key")
//    // "i"-вий елемент перемістити вправо (у позицію "i+1")
//    // перейти до перевірки попереднього елемента (з номером i-1)
//кінєць_поки
//вставляєм поточне значення "key" після елемента,
//який вже не був більшим (або дійшли до початку масиву)
//кінєць_для
```

```
let key, i
for (let k = 1; k < arr.length; k++) {
    key = arr[k]
    i = k - 1
    while (i >= 0 && arr[i] > key) {
        arr[i + 1] = arr[i]
        i = i - 1
    }
    arr[i + 1] = key
}
```

```
let a = [23, 0, 0, 4, 11, -2, 9, -5]

// ---- Сортивання включеннями
function insertSort(arr) {
  for (let k = 1; k < arr.length; k++) {
    const currentElement = arr[k]
    let i = k - 1
    while (i >= 0 && arr[i] > currentElement) {
      arr[i + 1] = arr[i]
      i = i - 1
    }
    arr[i + 1] = currentElement
  }
}
```

```
const sortedArray = insertSort(a)
document.write(a)
```

## Сортування вибором 1

```
//=====
// Сортування вибором 1 - поганий
//Цикл_для кожної позиції "i" від 0 до передостанньої
//  ----- поставити мінімальний від "i" до останнього у позицію "i" -----
//  Цикл_для номерів "j" від "i+1" до останнього
//      якщо елемент з номером "j" менше за елемент з номером "i"
//          -- то міняти місцями елементи з номерами "i" та "j"
//  кінець_циклу_для_номера_ j
//кінець_циклу_для_номера_i
//=====
```

```
const prevLastIndex = arr.length - 2
let counter = 0
for (let i = 0; i <= prevLastIndex; i++) {
  for (let j = i + 1; j < arr.length; j++) {
    if (arr[i] > arr[j]) {
      let temp = arr[i]
      arr[i] = arr[j]
      arr[j] = temp
      counter++
    }
  }
}
```

## Сортування вибором 2

```
//=====
// Сортування вибором
//Цикл_для кожної позиції "i" від 0 до передостанньої
//  ----- Знайти "currentMinIndex" - індекс мінімального елемента
//                                           з номером від "i" до останнього
//      спочатку "currentMinIndex" = "i"
//      Цикл_для номерів "j" від "i+1" до останнього
//      якщо елемент з номером "j" менше за елемент з номером "currentMinIndex"
//      -- то "currentMinIndex" = "j"
//      кінець_циклу_для
//      -----
//      Якщо "currentMinIndex" не дорівнює позиції "i"
//      -- то поміняти місцями елементи з номерами "i" та "currentMinIndex"
//кінець_циклу_для
```

```
//=====
// Сортуння вибором
// Цикл_для кожної позиції "i" від 0 до передостанньої
// ----- Знайти "currentMinIndex" - індекс мінімального елемента з номером від "i" до останнього
// ..... спочатку "currentMinIndex" = "i"
// ... Цикл_для номерів "j" від "i+1" до останнього
// ..... якщо елемент з номером "j" менше за елемент з номером "currentMinIndex"
// ..... то "currentMinIndex" = "j"
// ... кінець_циклу_для
// -----
// ... Якщо "currentMinIndex" не дорівнює позиції "i"
// ..... то поміняти місцями елементи з номерами "i" та "currentMinIndex"
// кінець_циклу_для
```

```
const prevLastIndex = arr.length - 2
let counter = 0
for (let i = 0; i <= prevLastIndex; i++) {
  let minIndex = i
  //----- пошук індекса мінімального елемента від i-до останнього
  for (let j = i + 1; j < arr.length; j++) {
    if (arr[j] < arr[minIndex]) {
      minIndex = j
    }
  }
  //----- якщо мінімлаьний не у позиції "i" то міняємо місцями
  if (i !== minIndex) {
    let temp = arr[i]
    arr[i] = arr[minIndex]
    arr[minIndex] = temp
    counter++
  }
}
```



## ***Двійковий (бінарний) пошук***

**Двійко́вий по́шук** — [алгоритм](#) знаходження заданого значення у впорядкованому [масиві](#), який полягає у порівнянні серединного елемента масиву з шуканим значенням, і повторенням алгоритму для тієї або іншої половини (див. [двійкове дерево пошуку](#)), залежно від результату порівняння.

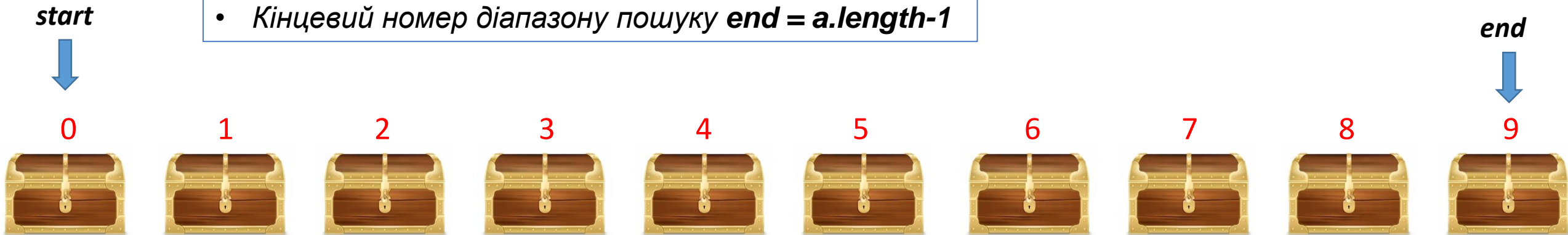


## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число `searchElement = 27`

Спочатку діапазон пошуку – весь масив

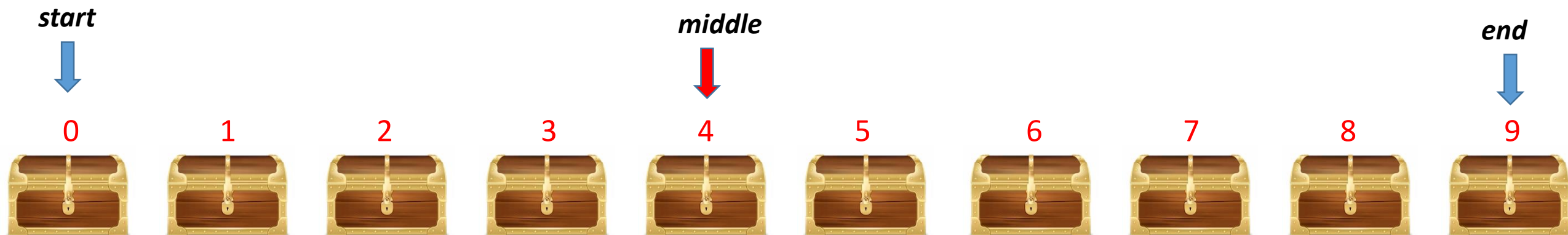
- Початковий номер діапазону пошуку **`start = 0`**
- Кінцевий номер діапазону пошуку **`end = a.length - 1`**



```
function includes_binarySearch(a, searchElement) {  
  let start = 0  
  let end = a.length - 1  
  
  return false  
}
```

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число 27



$middle = \text{Math.floor}((start+end)/2)$

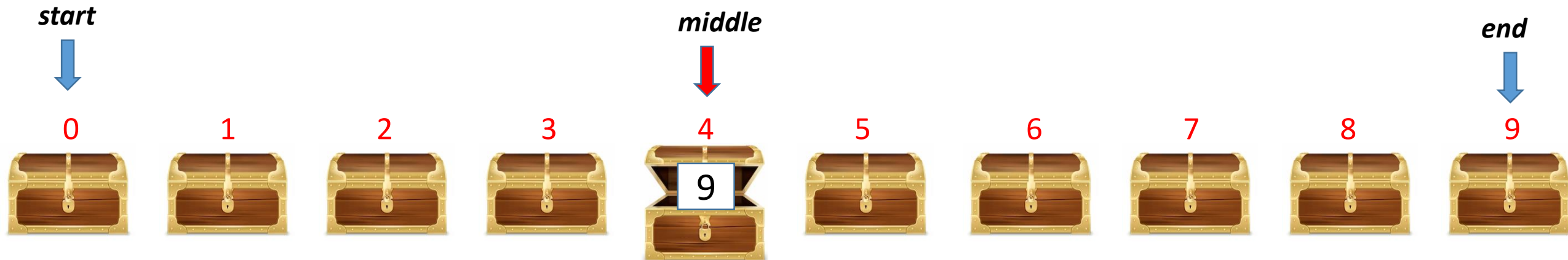
Знаходимо елемент **middle**,  
що знаходиться між **start** та **end**

```
function includes_binarySearch(a, searchElement) {  
  let start = 0  
  let end = a.length - 1  
  while (start <= end) {  
    const middle = Math.floor((start + end) / 2)  
    // ...  
  }  
  return false  
}
```

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

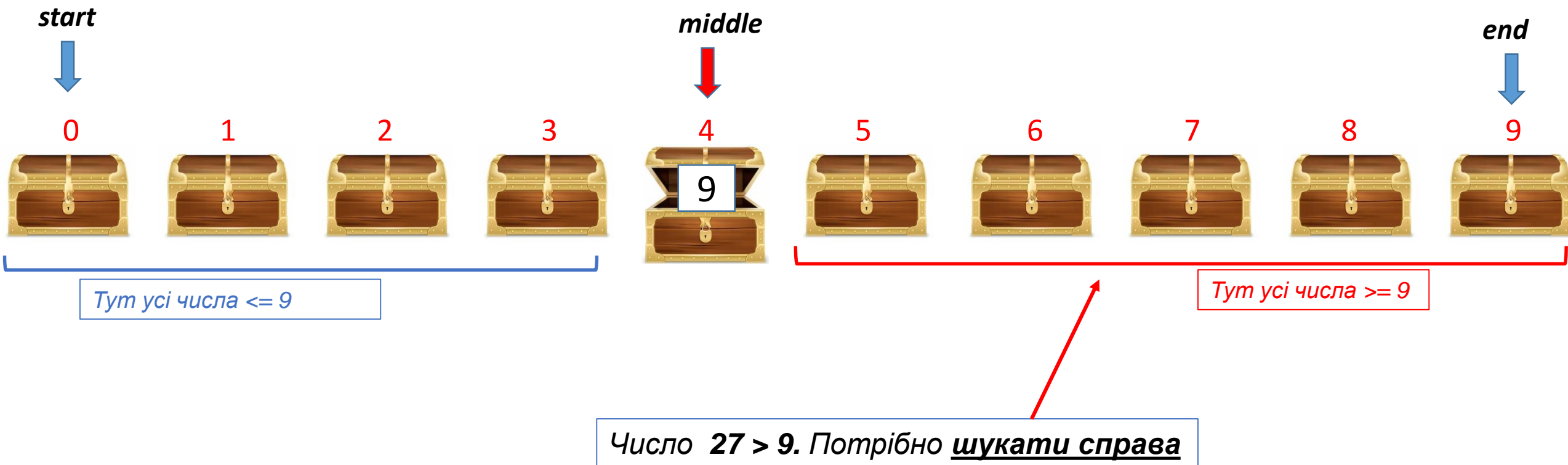


Порівнюємо елемент, що знаходиться у позиції **middle** з шуканим елементом 27

## Двійковий (бінарний) пошук

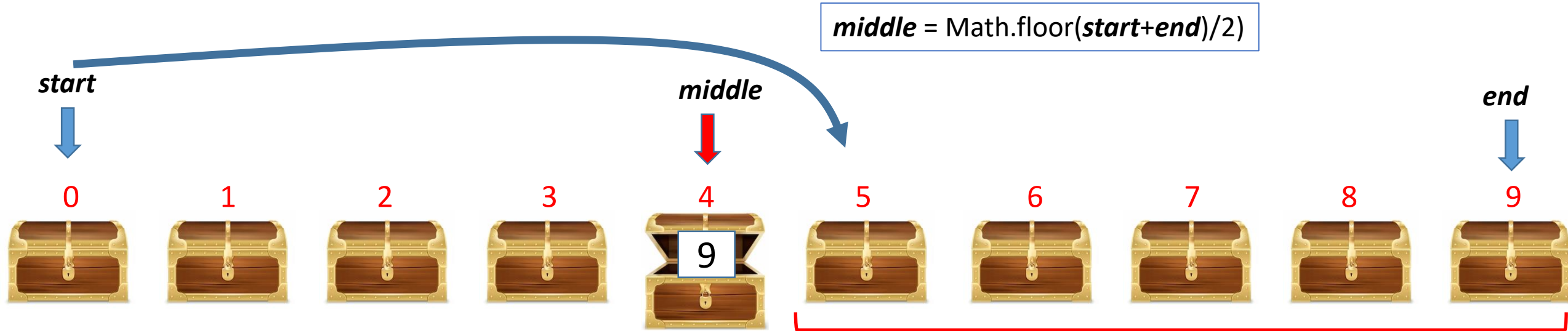
Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}(start+end)/2$



## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число `searchElement = 27`



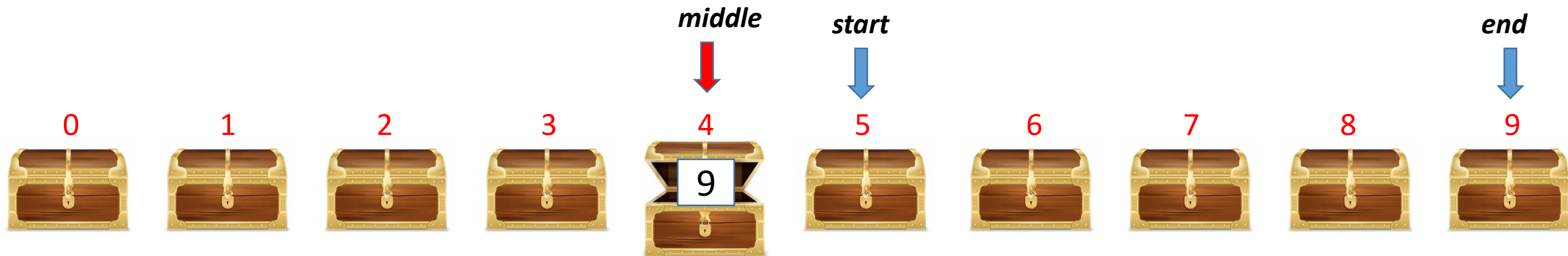
Число  $27 > 9$ . Потрібно **шукати справа**  
`start` – зміщуємо вправо від `middle`  
(`start = middle + 1`)

Тут усі числа  $\geq 9$

```
function includes_binarySearch(a, searchElement) {
  let start = 0
  let end = a.length - 1
  while (start <= end) {
    const middle = Math.floor((start + end) / 2)
    if (a[middle] === searchElement) return true
    if (a[middle] < searchElement) start = middle + 1
    if (a[middle] > searchElement) end = middle - 1
  }
  return false
}
```

## Двійковий (бінарний) пошук

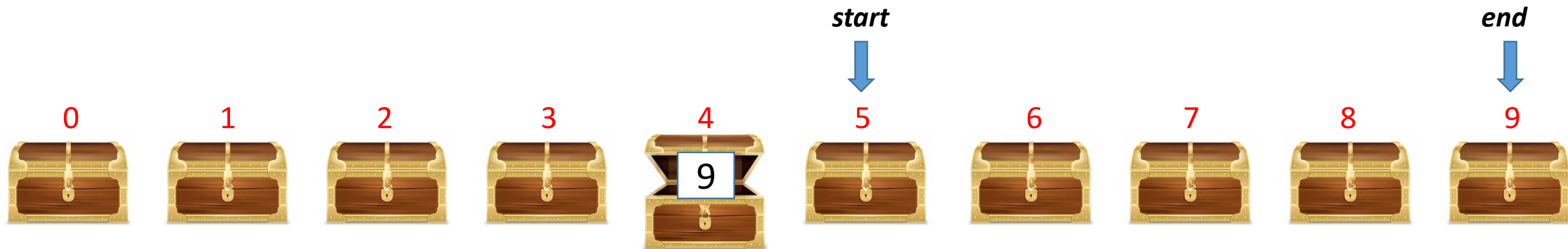
Числа у скринях упорядковані за зростанням. Шукаємо число **27**



Число  $27 > 9$ . Потрібно шукати справа  
start – зміщуємо вправо від middle

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

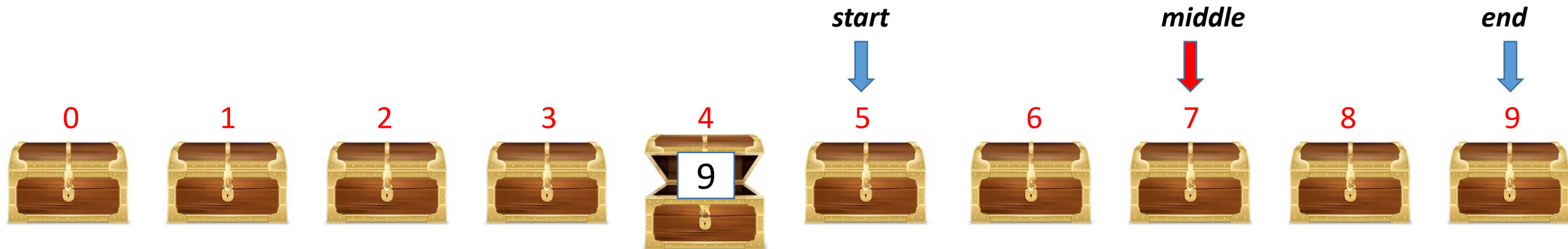




## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

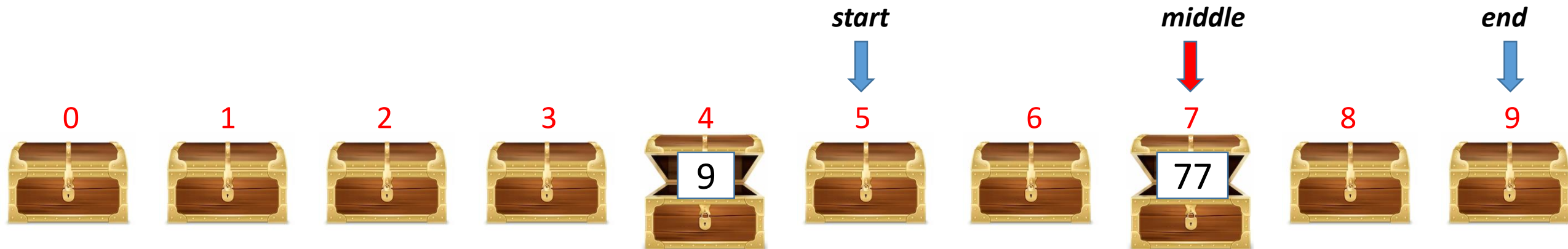


Знаходимо елемент **middle**, що знаходиться між **start** та **end**

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

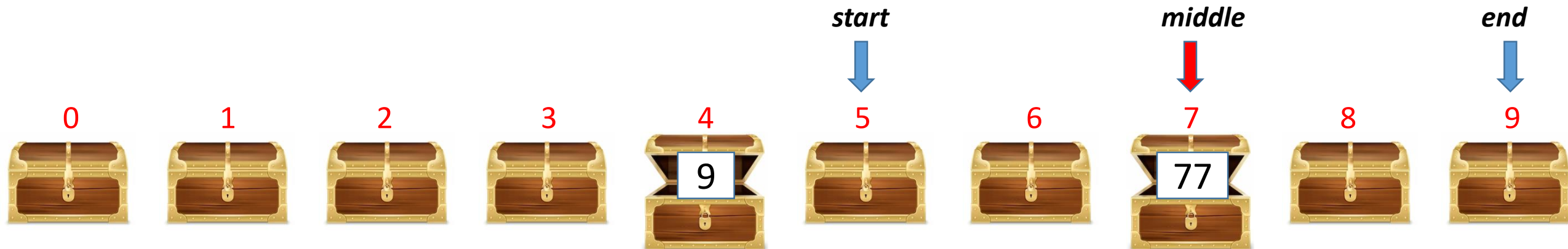


Знаходимо елемент **middle**, що знаходиться між **start** та **end**

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

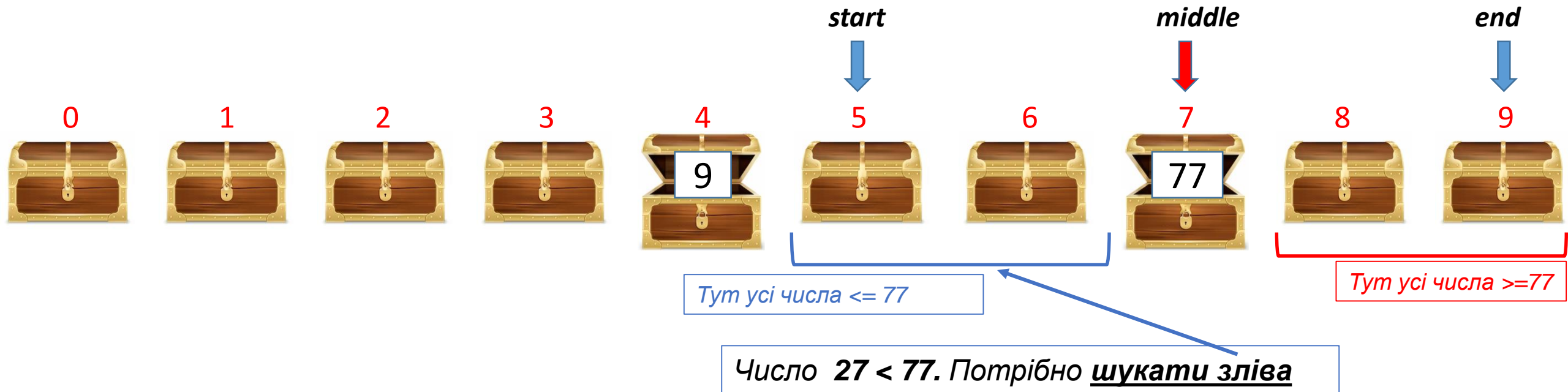


Порівнюємо елемент, що знаходиться у позиції **middle** з шуканим елементом 27

## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число **27**

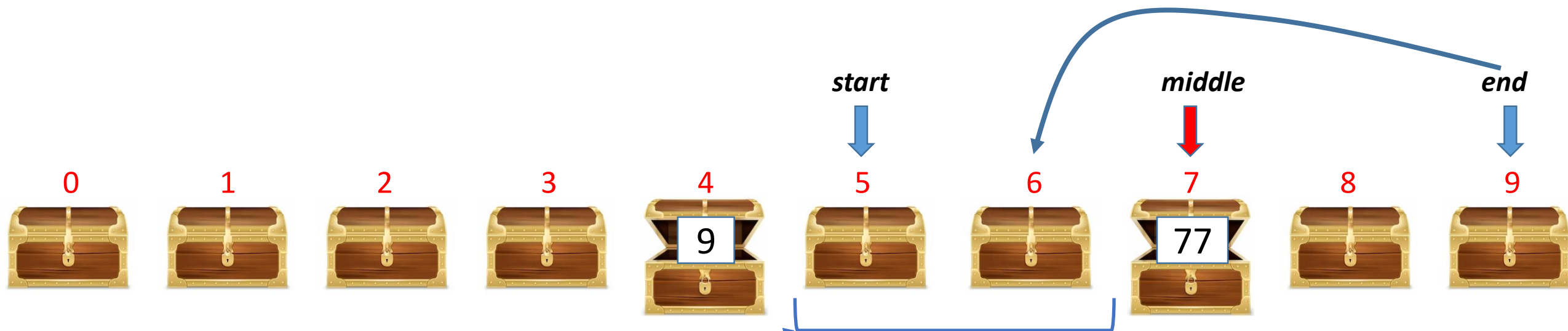
$middle = \text{Math.floor}((start+end)/2)$



## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start + end) / 2)$



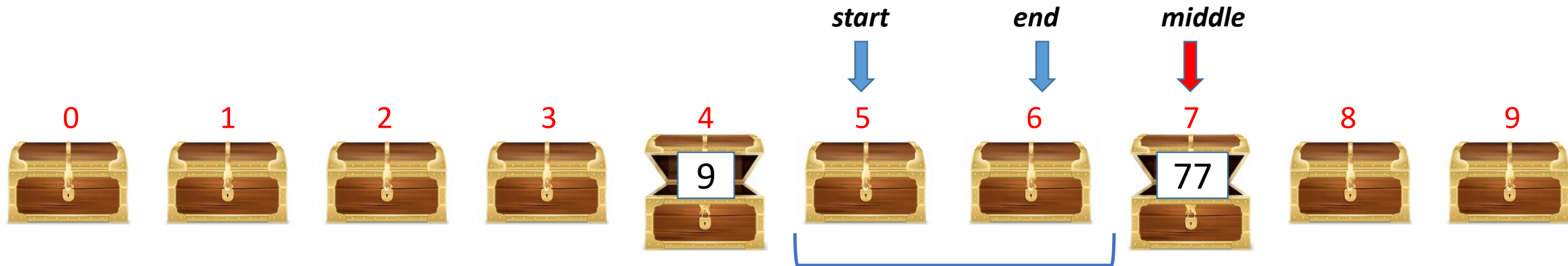
Число  $27 < 77$ . Потрібно шукати зліва  
end – зміщуємо вліво від middle  
( $end = middle - 1$ )

```
function includes_binarySearch(a, searchElement) {  
  let start = 0  
  let end = a.length - 1  
  while (start <= end) {  
    const middle = Math.floor((start + end) / 2)  
    if (a[middle] === searchElement) return true  
    if (a[middle] < searchElement) start = middle + 1  
    if (a[middle] > searchElement) end = middle - 1  
  }  
  return false  
}
```

## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

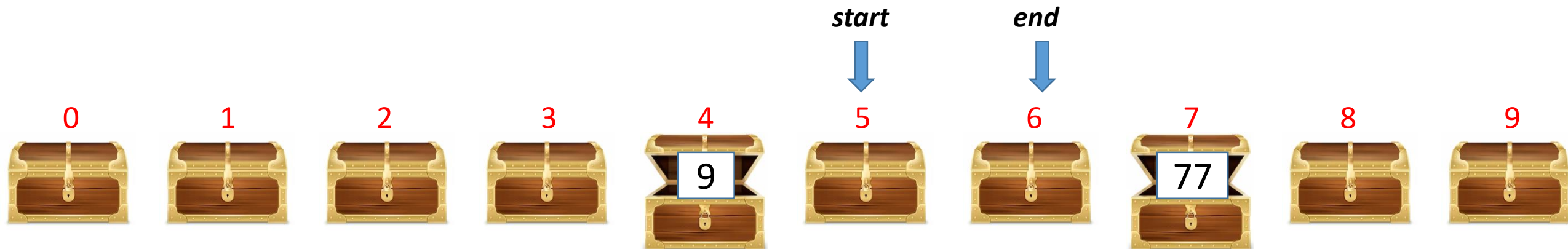


Число  $27 < 77$ . Потрібно шукати зліва  
end – зміщуємо вліво від middle

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

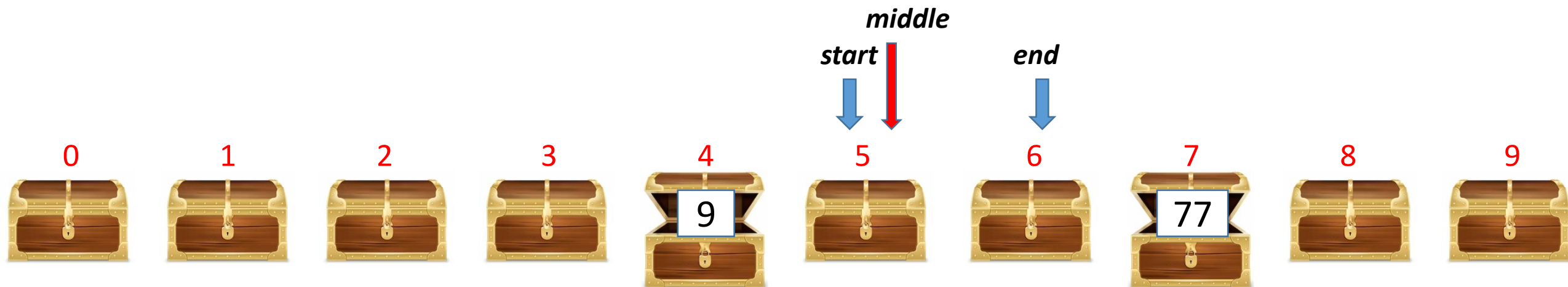
$middle = \text{Math.floor}((start+end)/2)$



## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



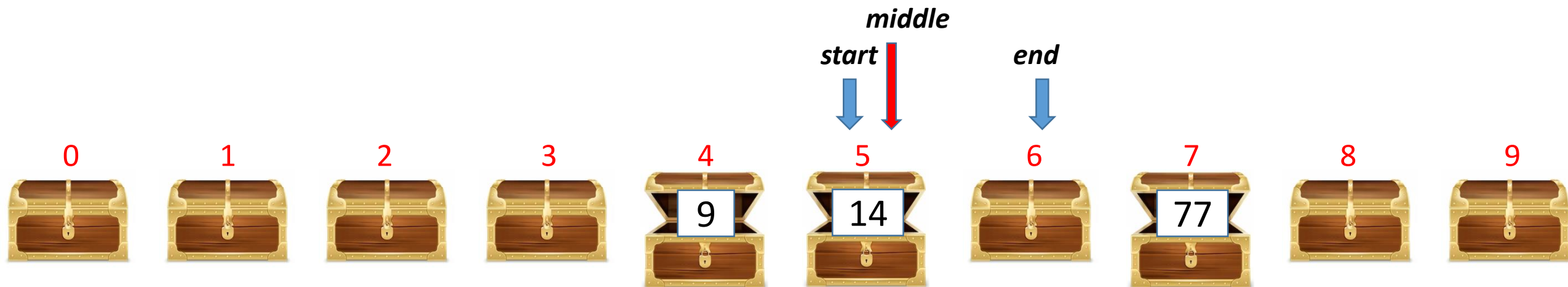
Знаходимо елемент **middle**, що знаходиться між **start** та **end**



## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

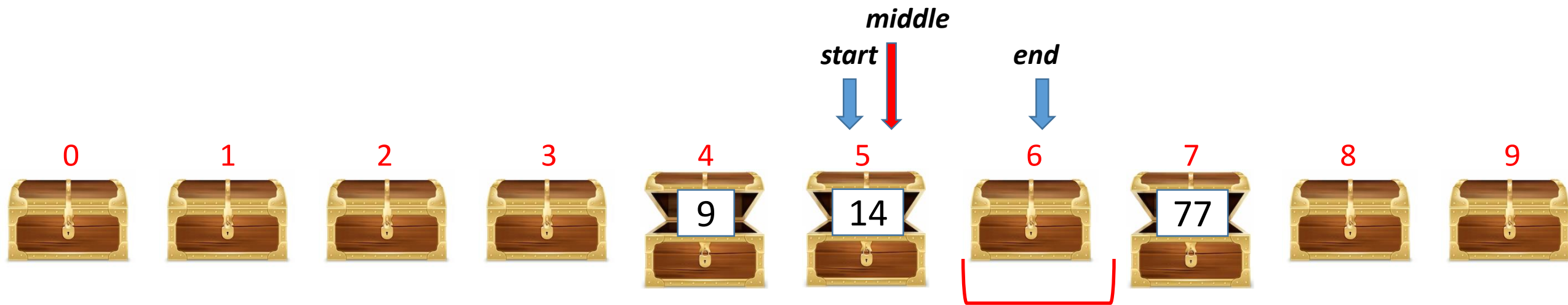


Порівнюємо елемент, що знаходиться у позиції **middle** з шуканим елементом 27

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



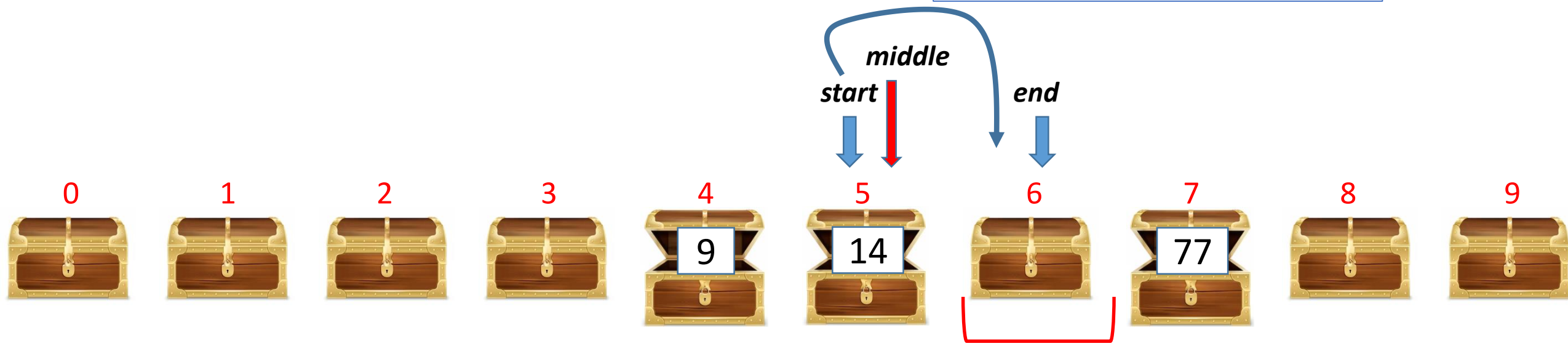
Число  $27 > 14$ . Потрібно шукати справа

Порівнюємо елемент, що знаходиться у позиції **middle** з шуканим елементом 27

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$

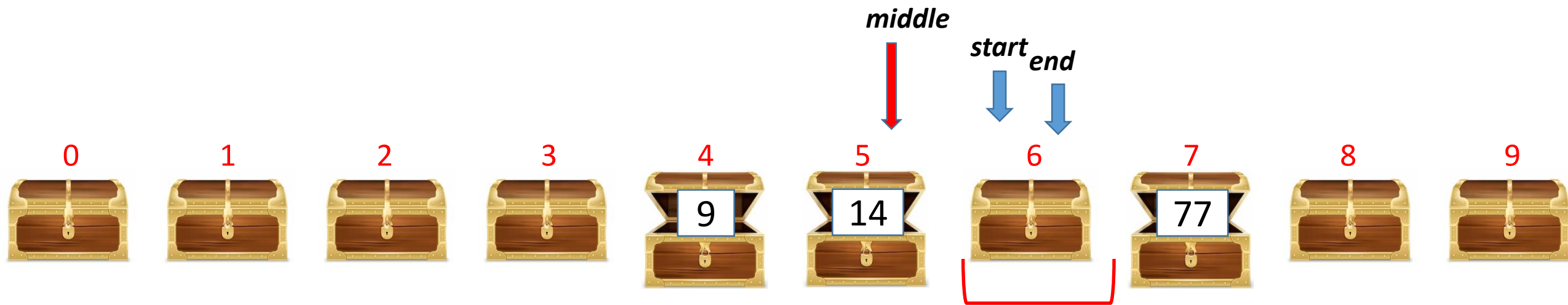


Число  $27 > 14$ . Потрібно шукати справа  
start – зміщуємо вправо від middle

## Двійковий (бінарний) пошук

Числа у скринях **упорядковані за зростанням**. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



Число  $27 > 14$ . Потрібно шукати справа  
start – зміщуємо вправо від middle

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



Знаходимо елемент **middle**, що знаходиться між **start** та **end**

## Двійковий (бінарний) пошук

Числа у скринях упорядковані за зростанням. Шукаємо число **27**

$middle = \text{Math.floor}((start+end)/2)$



Порівнюємо елемент, що знаходиться у позиції **middle** з шуканим елементом 27

**Ура!!!! Ми знайшли!!!!**

Якщо б тут було не 27, то його не було б у даній послідовності.

```
//----·Перевірка·належності·елемента·упорядкованого·масиву·  
//·····(належить·-·true,·не·належить·-·false)  
function·includes_binarySearch(a·,·searchElement){  
·let·start·=·0  
·let·end·=·a·.length·-·1  
·while·(start·<=·end){  
··const·middle·=·Math.floor((start·+·end)·/·2)  
··if·(a[middle]·===·searchElement)·return·true  
··if·(a[middle]·<·searchElement)·start·=·middle·+·1  
··if·(a[middle]·>·searchElement)·end·=·middle·-·1  
·}  
·return·false  
}
```

```
const·sortedArray·=·insertSort(a)  
document.write(a)  
if·(includes_binarySearch(a,·4))·alert('4·входить·у·масив')  
else·alert('4·не·входить·у·масив')
```



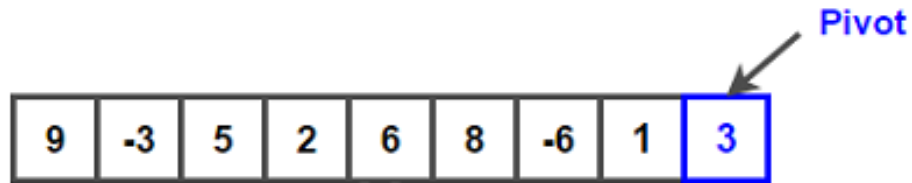
```
//---- Знаходження індексу елемента у масиві (якщо немає, то результат = -1)
function findIndex_binarySearch(a, searchElement) {
    let start = 0
    let end = a.length - 1
    while (start <= end) {
        const middle = Math.floor((start + end) / 2)
        if (a[middle] === searchElement) return middle
        if (a[middle] < searchElement) start = middle + 1
        if (a[middle] > searchElement) end = middle - 1
    }
    return -1
}
```

```
const sortedArray = insertSort(a)
```

```
const searchIndex = findIndex_binarySearch(a, 4)
if (searchIndex !== -1)
    alert(`4 входить у масив під індексом ${searchIndex}`)
else alert('4 не входить у масив')
```

## Швидке сортування

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої.



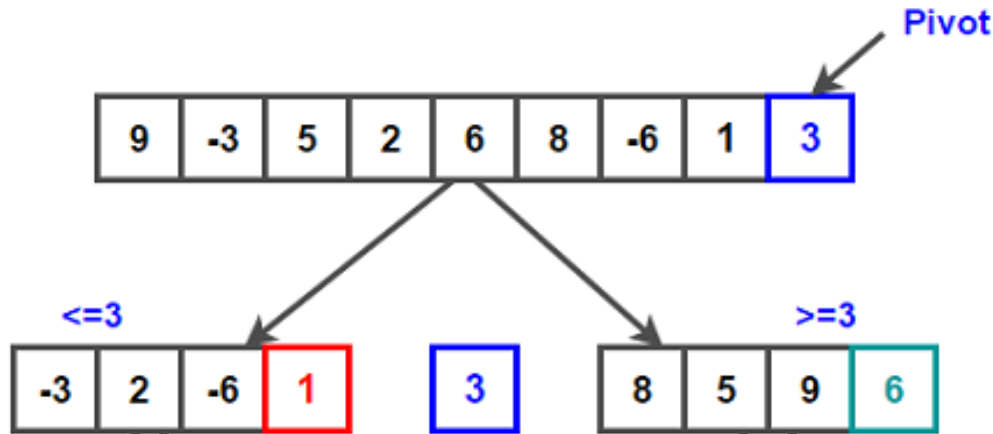
```
function Partition(A, p, q) {  
    let pivot = A[q]  
    let i = p - 1  
    for (let j = p; j < q; j++) {  
        if (A[j] <= pivot) {  
            i = i + 1  
            swap(A, i, j)  
        }  
    }  
    swap(A, i + 1, q)  
    return i + 1  
}
```

[https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5\\_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

<https://www.techiedelight.com/ru/quicksort/>

## Швидке сортування

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої.



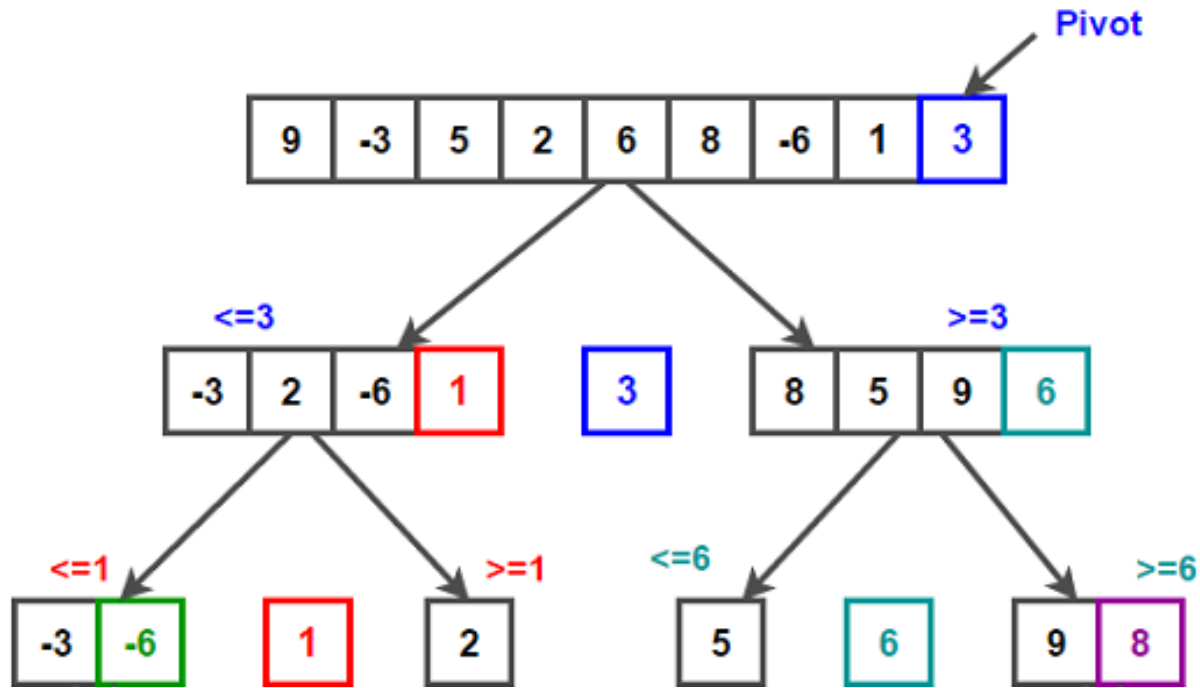
```
function Partition(A, p, q) {  
    let pivot = A[q]  
    let i = p - 1  
    for (let j = p; j < q; j++) {  
        if (A[j] <= pivot) {  
            i = i + 1  
            swap(A, i, j)  
        }  
    }  
    swap(A, i + 1, q)  
    return i + 1  
}
```

[https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5\\_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

<https://www.techiedelight.com/ru/quicksort/>

## Швидке сортування

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої.



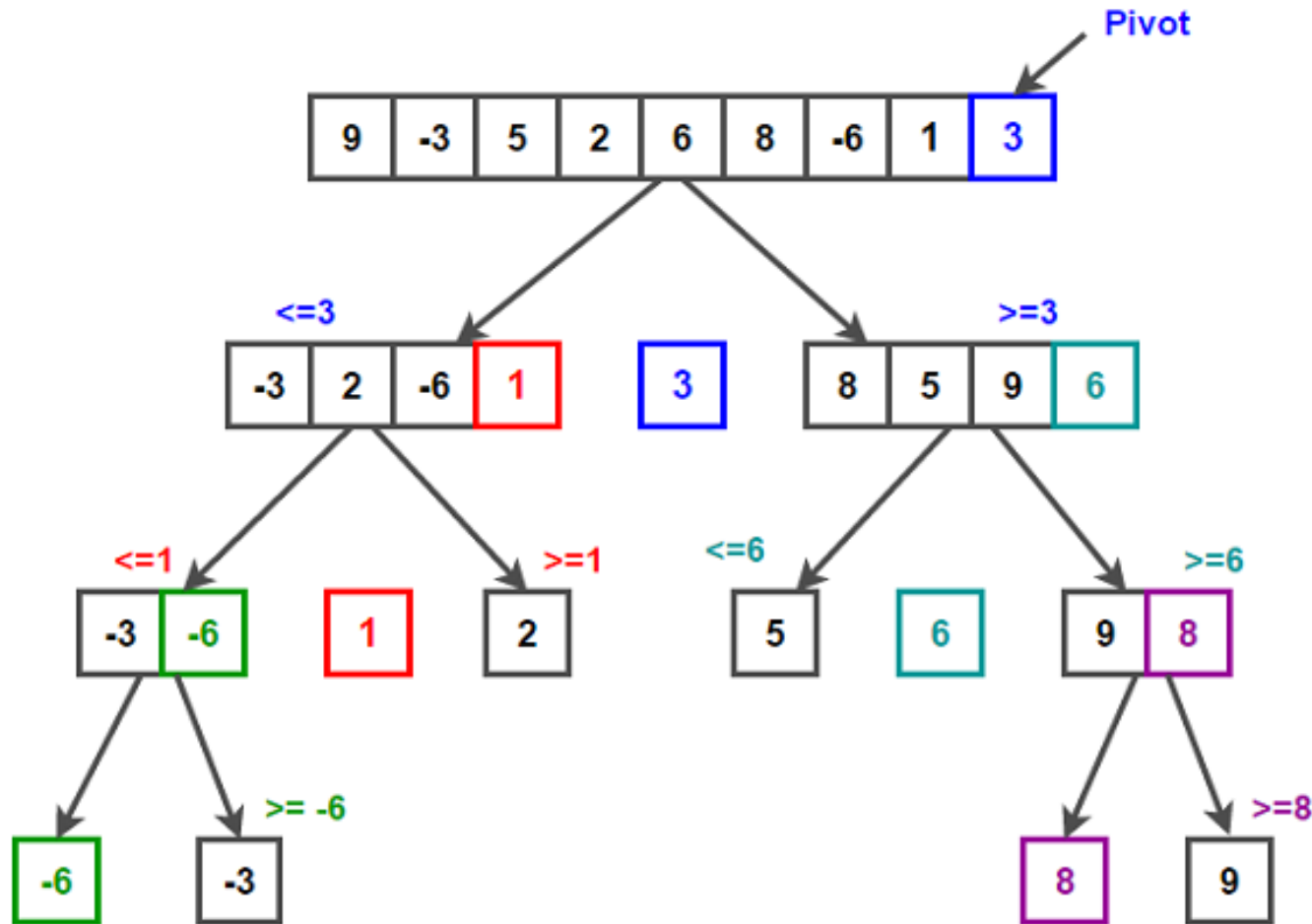
```
function Partition(A, p, q) {  
    let pivot = A[q]  
    let i = p - 1  
    for (let j = p; j < q; j++) {  
        if (A[j] <= pivot) {  
            i = i + 1  
            swap(A, i, j)  
        }  
    }  
    swap(A, i + 1, q)  
    return i + 1  
}
```

[https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5\\_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

<https://www.techiedelight.com/ru/quicksort/>

## Швидке сортування

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої.



```
function Partition(A, p, q) {
    let pivot = A[q]
    let i = p - 1
    for (let j = p; j < q; j++) {
        if (A[j] <= pivot) {
            i = i + 1
            swap(A, i, j)
        }
    }
    swap(A, i + 1, q)
    return i + 1
}
```

[https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5\\_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

<https://www.techiedelight.com/ru/quicksort/>

1) `//Функція обміну елементів місцями`  
`function swap(A, i, j) {`  
    `let temp = A[i]`  
    `A[i] = A[j]`  
    `A[j] = temp`  
`}`

2) `//Поділ фрагменту від p до q`  
`//на дві частини (<=pivot та >pivot)`  
`function Partition(A, p, q) {`  
    `let pivot = A[q]`  
    `let i = p - 1`  
    `for (let j = p; j < q; j++) {`  
        `if (A[j] <= pivot) {`  
            `i = i + 1`  
            `swap(A, i, j)`  
        `}`  
    `}`  
    `swap(A, i + 1, q)`  
    `//повертаємо нову позицію pivot`  
    `return i + 1`  
`}`

3) `//Основна функція`  
`function Quicksort(A, p, q) {`  
    `if (p >= q) return`  
    `//ділимо на дві частини`  
    `let i = Partition(A, p, q)`  
    `//окремо сортуємо першу частину`  
    `Quicksort(A, p, i - 1)`  
    `//окремо сортуємо другу частину`  
    `Quicksort(A, i + 1, q)`  
`}`

4) `// Тестовий масив`  
`let arr = [`  
    `56, -78, 252, 23, -122, 356, 122, 436,`  
    `-111, 344, 21, 3, -356, 342,`  
`]`  
`Quicksort(arr, 0, arr.length - 1)`  
`document.write(arr)`