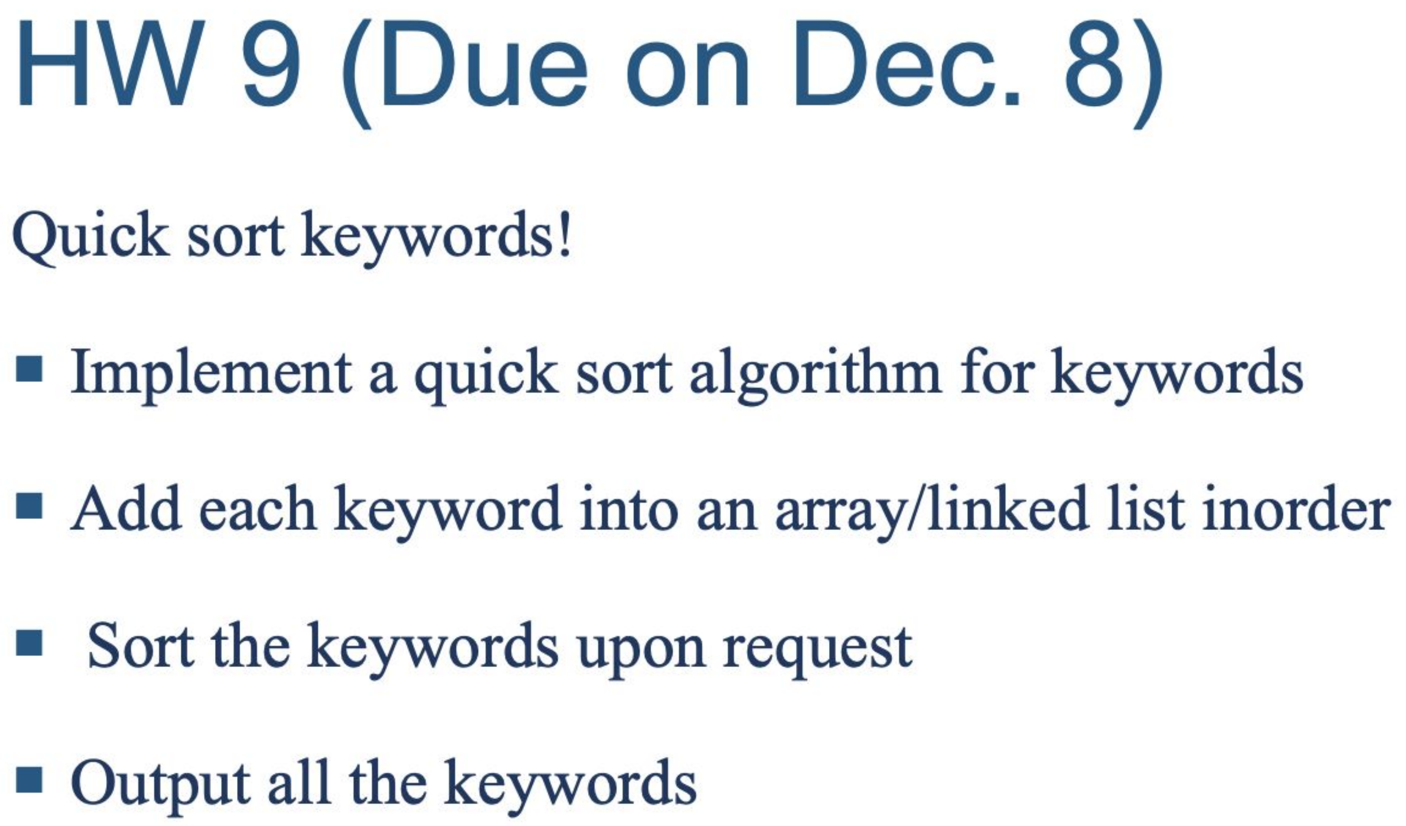
HW9

林苡晴 110356019@nccu.edu.tw
蔣其叡 111356024@nccu.edu.tw

HW9

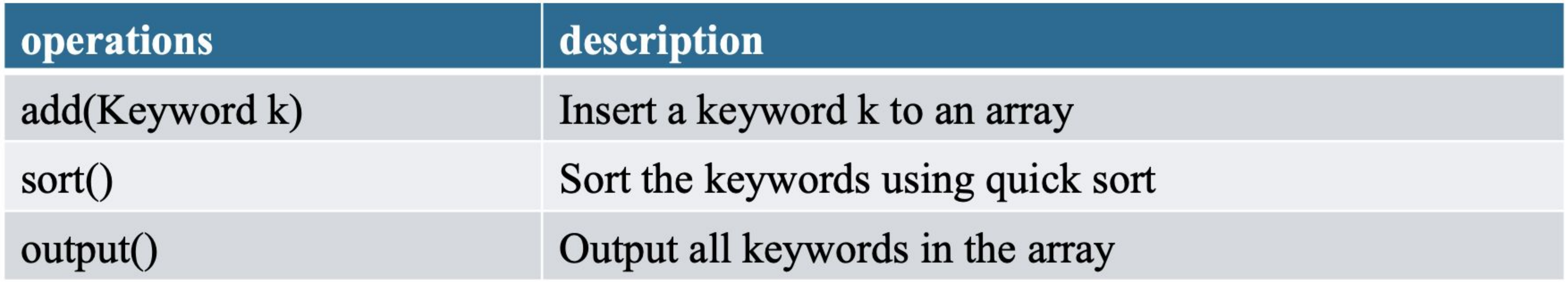


HW 9 (Due on Dec. 8)

Quick sort keywords!

- Implement a quick sort algorithm for keywords
- Add each keyword into an array/linked list inorder
- Sort the keywords upon request
- Output all the keywords

Operations



operations	description
add(Keyword k)	Insert a keyword k to an array
sort()	Sort the keywords using quick sort
output()	Output all keywords in the array

Keyword

- A keyword is a tuple of [String *name*, Integer *count*]
  - For example:

```
{
    name: “Fang”,
    count: 3
}
```

- A keyword should output in format **[name,count]** :
  - [Fang,3]

Requirements

- Maintain a keyword list, and implement the Quick Sort algorithm
- List order

  keyword.count

- For the list structure, you can
  - Use java.util.ArrayList
  - Or develop it by yourself

I/O Example: add

- To do: Insert a keyword [k,c] to the list
- Input:
    - Token1 : a constant “add”
    - Token2 : keyword name **k**
    - Token3 : keyword count **c**
    - EX: add Fang 3

I/O Example: sort

- To do: Sort the list using Quick Sort.
- Input:
  - Token1 : a constant “sort”
  - EX: sort
- Output:
  - If list is empty, then output “InvalidOperation”:

    InvalidOperation

I/O Example: output

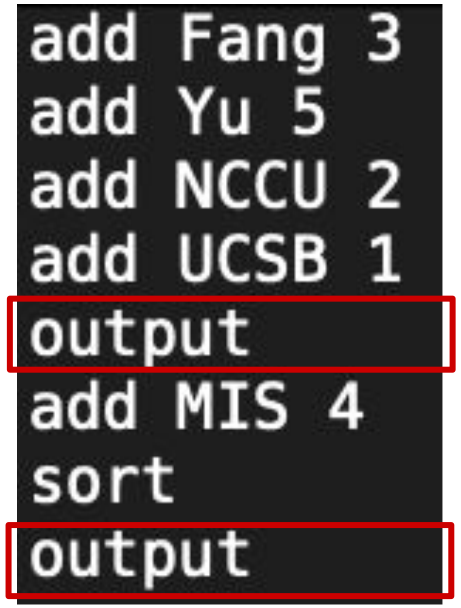
- To do: Output all the keywords in order (ascending)
- Input:
  - Token1 : a constant “output”
  - EX: output
- Output:
  - If list is empty, then output “InvalidOperation”:

    InvalidOperation
  - If list is not empty:

    [NCCU,4] [MIS,5] [DS,6]

Input file

- You need to read the sequence of operations from a txt file
- The format is firm
- Raise an exception if the input does not match the format



```
add Fang 3
add Yu 5
add NCCU 2
add UCSB 1
output
add MIS 4
sort
output
```

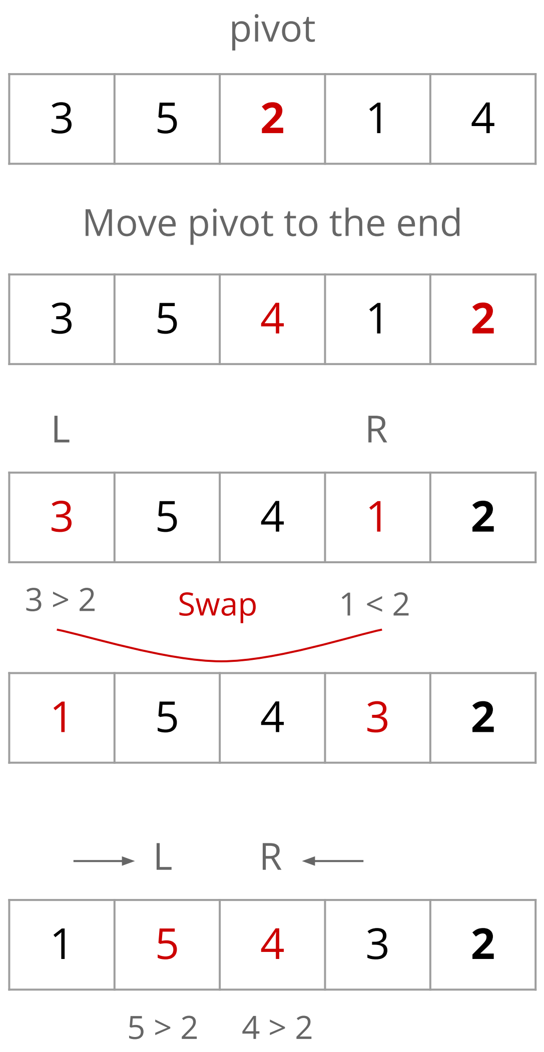
Way 1

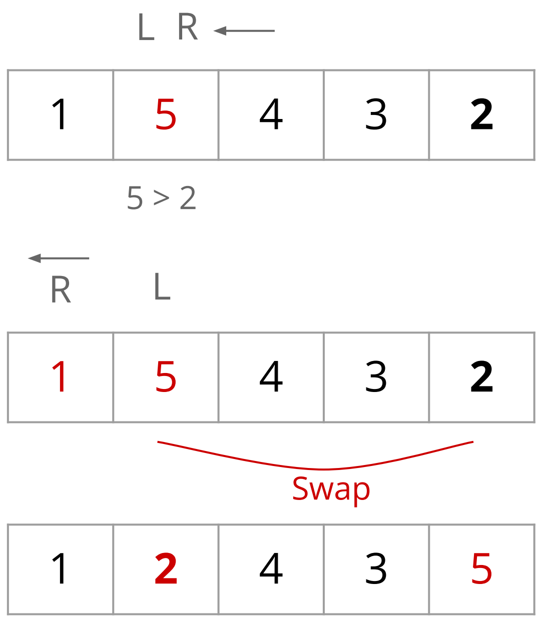
In-Place Quick-Sort

- Perform the partition using two indices to split S into L, E, G
- Algorithm Quicksort(leftBound, rightBound, S)
  - If(leftBound>=rightBound) return;
  - Set rightBound as the pivot (x = S[righBound])
  - Set j = leftBound; k = rightBound-1;
  - When j<k:
    - Scan j to the right (j++) until j >= k or the element S[j] > x.
    - Scan k to the left (k--) until j>=k or the element S[k]<=x.
    - Swap elements if j < k
  - Swap pivot with j
  - Quicksort(leftBound, j-1, S); Quicksort(j+1, rightBound, S)

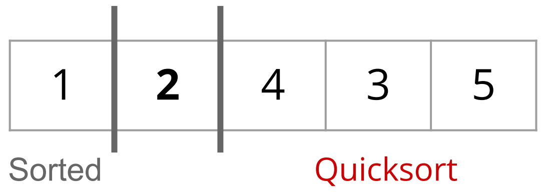
Way 1

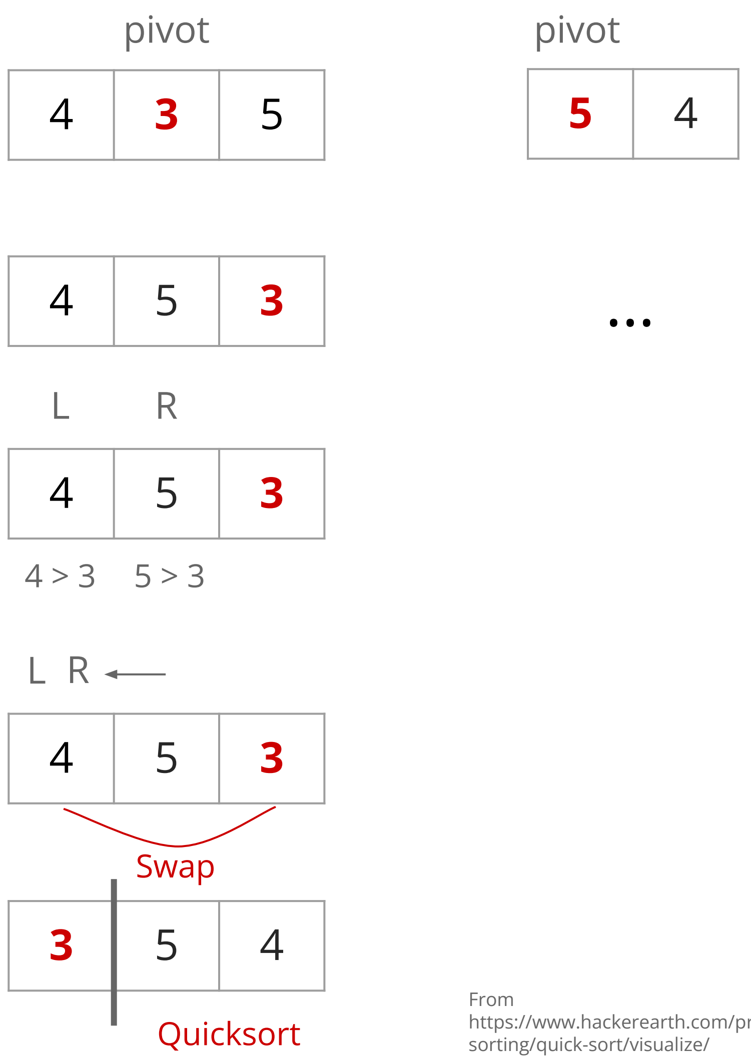
- Move the **left bound** to the right until it reaches a value **greater than** the pivot.
- Move the **right bound** to the left until it **crosses** the left bound or finds a value **less than or equal** to the pivot.





When the right bound crosses the left bound, all left elements are less than the pivot and all right elements are greater than or equal to the pivot. Move the pivot to its final location.

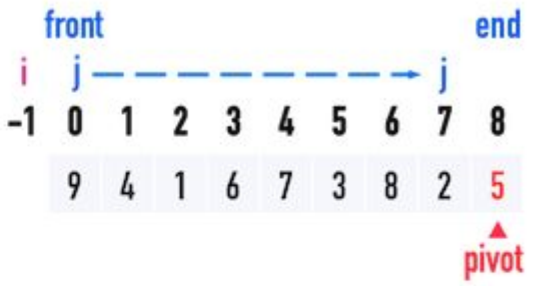




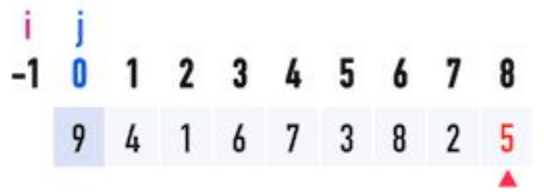
From https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/visualize/

Way 2

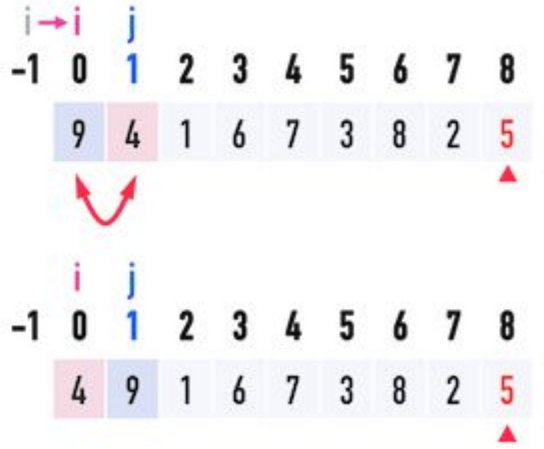
```
index: i = front -1
       j : front ~ end -1
Assign: pivot = array[ end ]
```



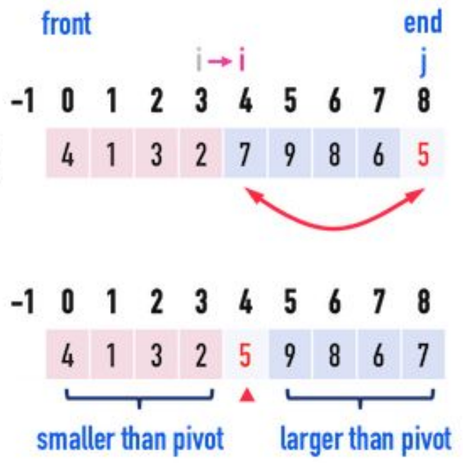
```
if array[ j ] < pivot
do i++
    swap( array[ i ], array[ j ] )
j++
```



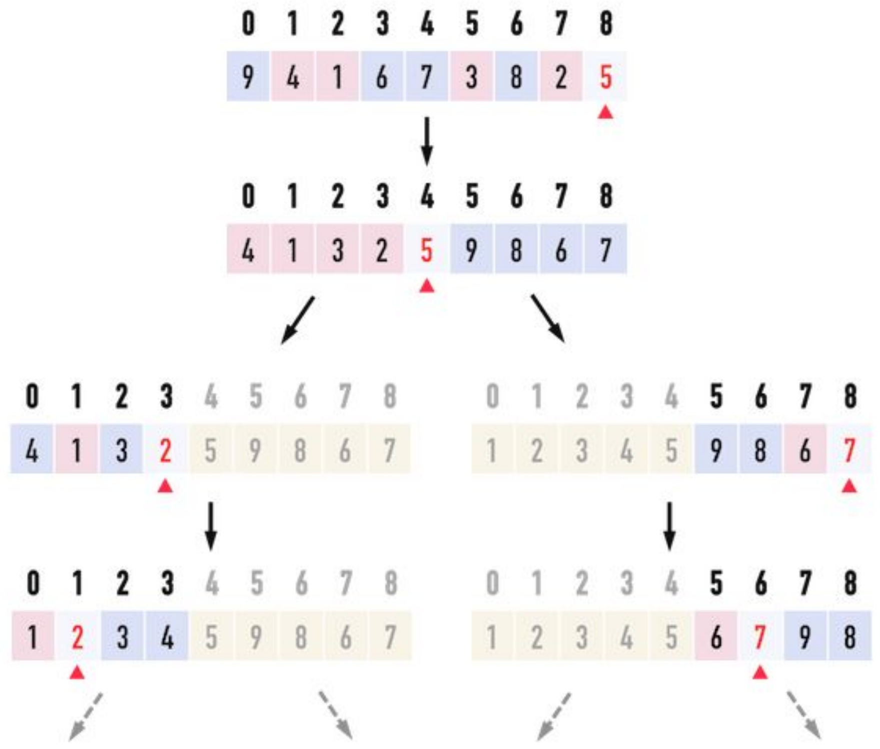
```
if array[ j ] < pivot
do i++
    swap( array[ i ], array[ j ] )
j++
```



```
when j reach end
do i++
    swap( array[ i ], array[ end ] )
```

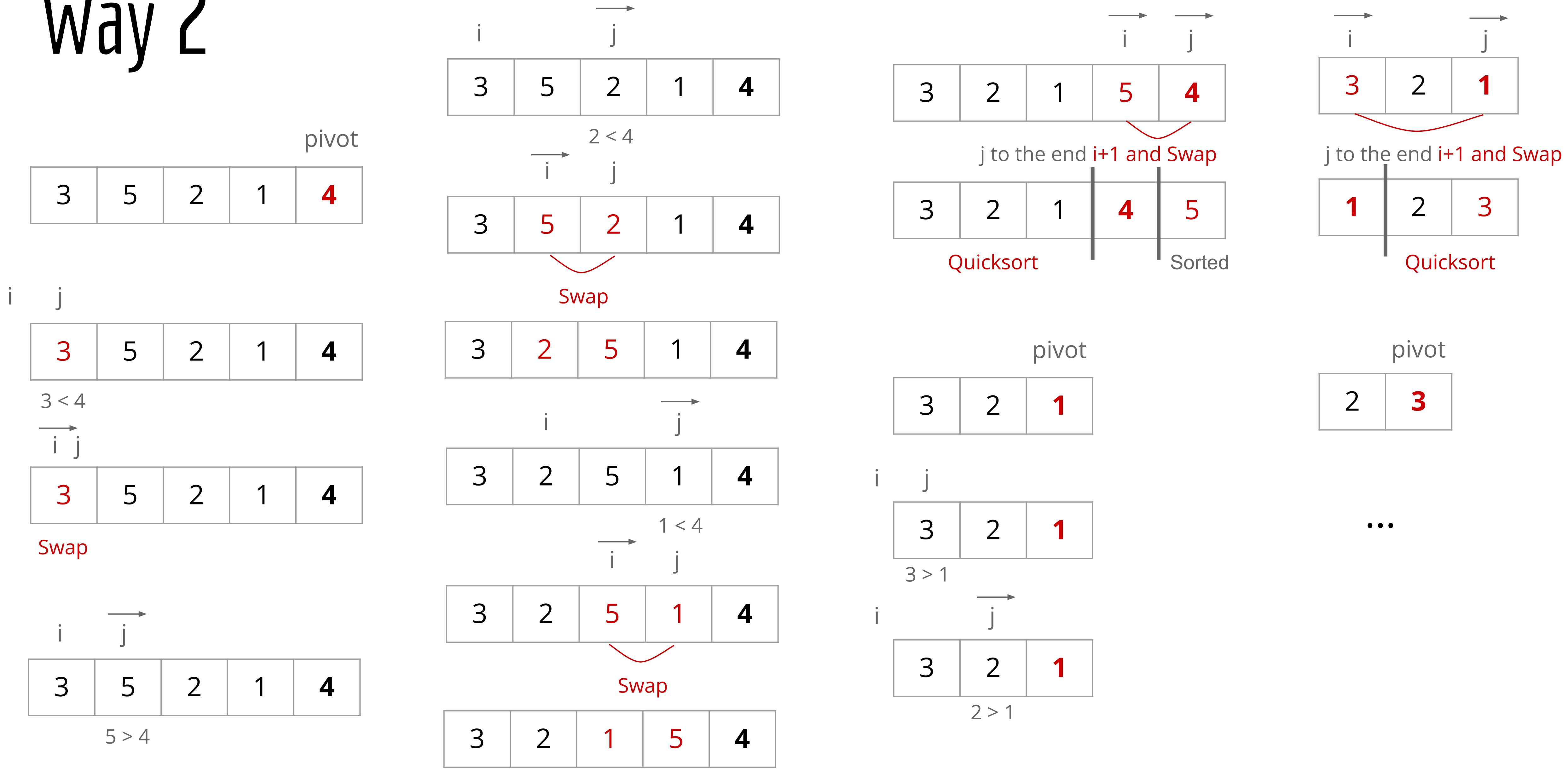


sort two subarrays recursively



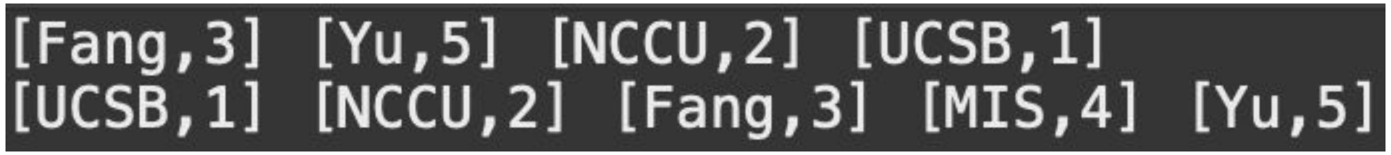
From http://alrightchiu.github.io/SecondRound/comparison-sort-quick-sortkuai-su-pai-xu-fa.html

Way 2



pivot
3 5 2 1 4
i j
3 5 2 1 4
3 < 4
i j
3 5 2 1 4
Swap
i j
3 5 2 1 4
5 > 4
i j
3 5 2 1 4
2 < 4
i j
3 5 2 1 4
Swap
3 2 5 1 4
i j
3 2 5 1 4
1 < 4
i j
3 2 5 1 4
Swap
3 2 1 5 4
i j
3 2 1 5 4
j to the end i+1 and Swap
3 2 1 4 5
Quicksort
Sorted
pivot
3 2 1
i j
3 2 1
3 > 1
i j
3 2 1
2 > 1
i j
3 2 1
j to the end i+1 and Swap
1 2 3
Quicksort
pivot
2 3
...

Output



```
[Fang,3] [Yu,5] [NCCU,2] [UCSB,1]
[UCSB,1] [NCCU,2] [Fang,3] [MIS,4] [Yu,5]
```