HW9

蔣其叡 111356024@nccu.edu.tw
陳卉縈 112356043@nccu.edu.tw

HW9

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

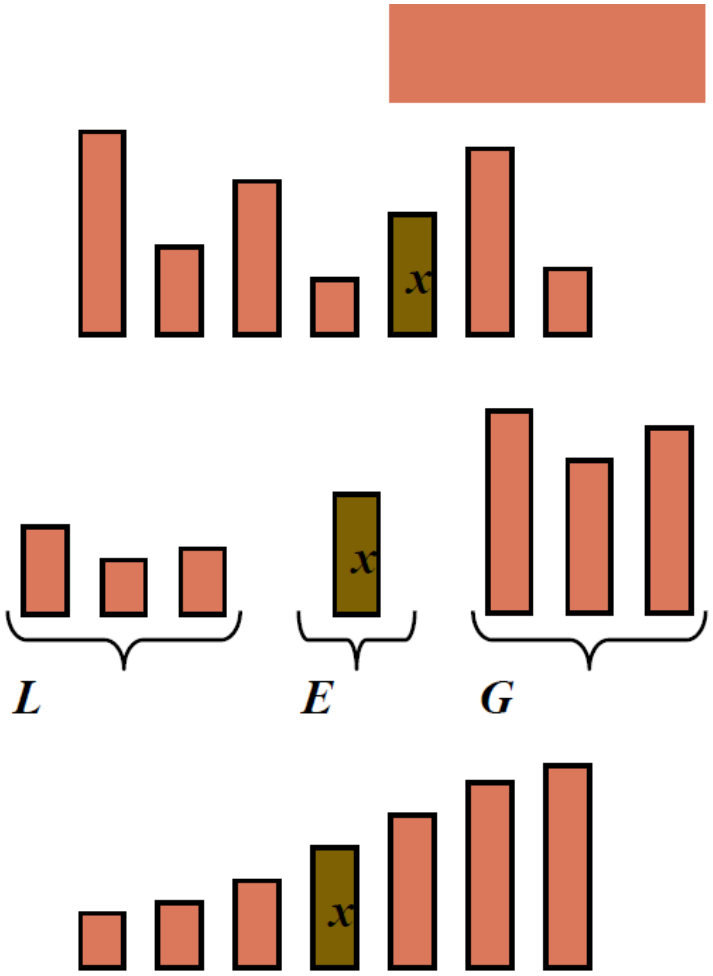
```
add Fang 3
add Yu 5
add NCCU 2
add UCSB 1
output
add MIS 4
sort
output
```

Way 1

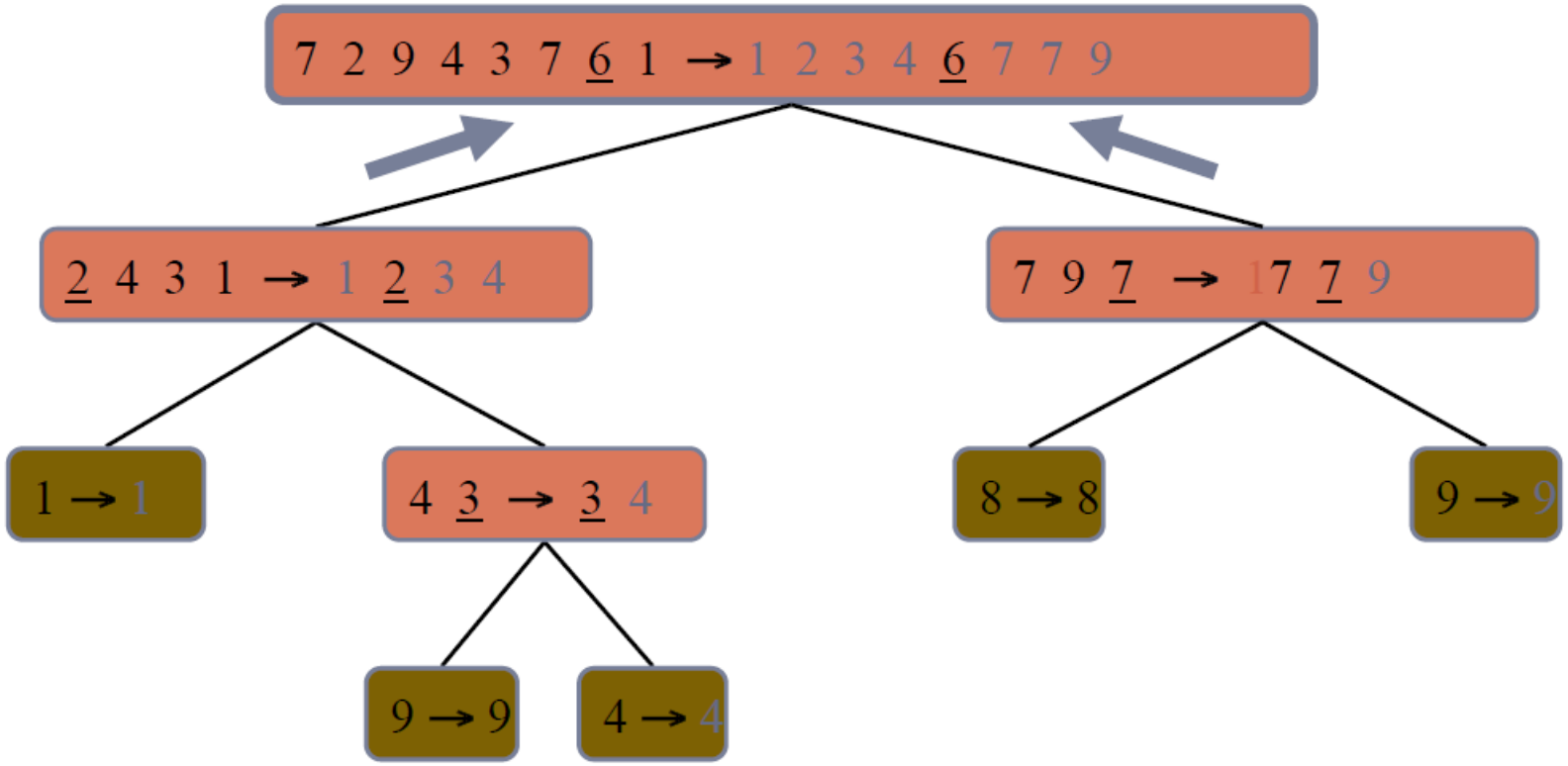
Quick-sort

A randomized sorting algorithm based on the divide-and-conquer paradigm:

- Divide: pick a random element x (called pivot) and partition S into
  - L elements less than x
  - E elements equal x
  - G elements greater than x
- Recur: sort L and G
- Conquer: join L, E and G



Way 1

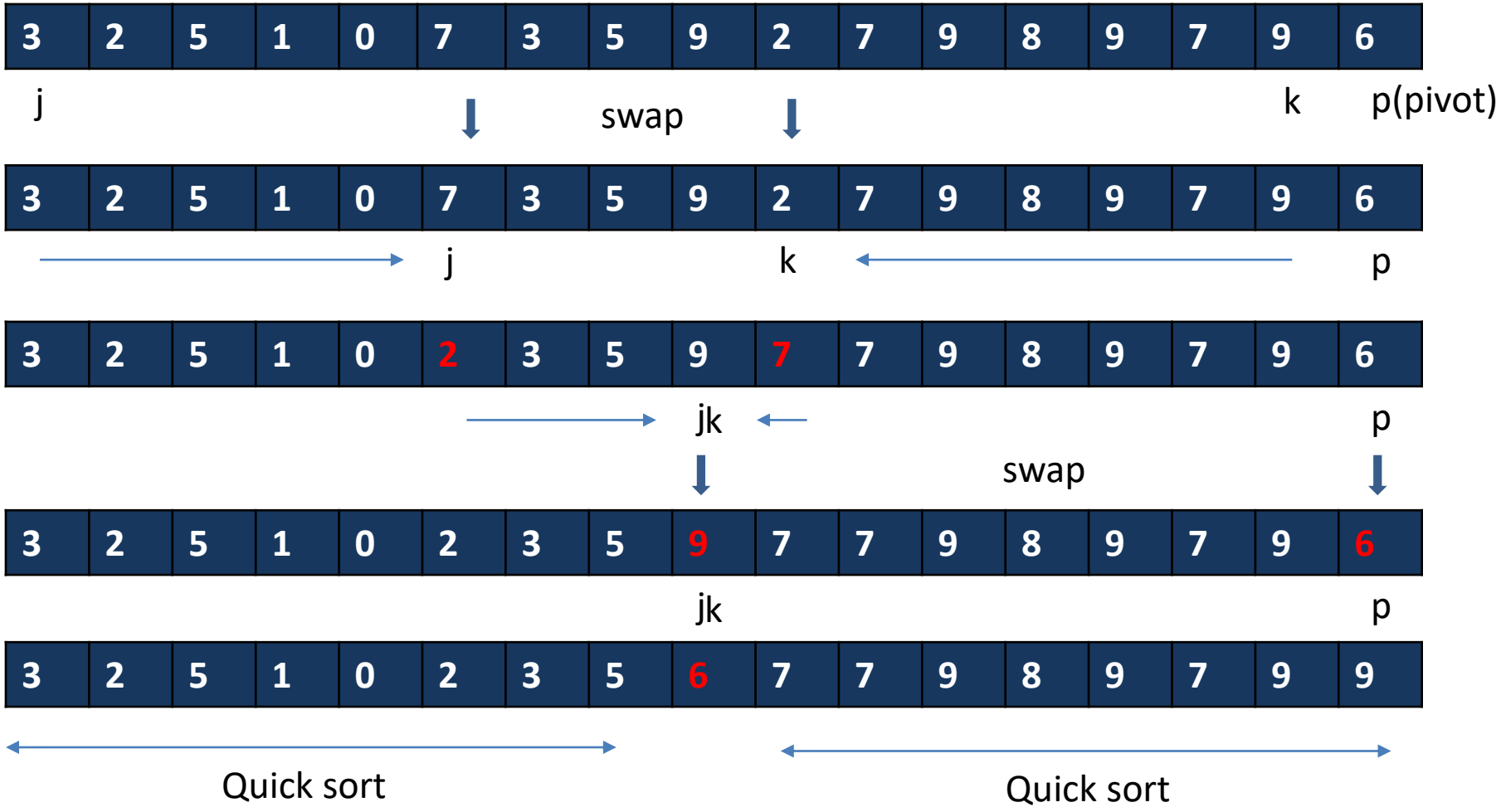


Way 2

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 2


3 2 5 1 0 7 3 5 9 2 7 9 8 9 7 9 6
j
swap
k
p(pivot)
3 2 5 1 0 7 3 5 9 2 7 9 8 9 7 9 6
j
k
p
3 2 5 1 0 2 3 5 9 7 7 9 8 9 7 9 6
jk
p
swap
3 2 5 1 0 2 3 5 9 7 7 9 8 9 7 9 6
jk
p
3 2 5 1 0 2 3 5 6 7 7 9 8 9 7 9 9
Quick sort
Quick sort

Output

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