Sorting & Searching III

DM2233 Advanced Data Structures & Algorithms

Module Schedule

Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Standard Template Library / Assignment 1	Vesak Day (Mon)
Week 8 and 9: Mid-Sem Break		
10	Sorting and Searching I	
11	Sorting and Searching II	
12	Sorting and Searching III	
13	Binary Tree I	Hari Raya Puasa (Fri)
14	Lab Test	
15	Binary Tree II	
16	Binary Tree III	SG50 Day (Fri)
17	Preprocessing / Assignment 2	National Day (Mon)

Introduction

- The previous 3 algorithms we have learnt are :
  - Bubble Sort
  - Selection Sort
  - Insertion Sort

- Today, the following sort algorithms will be covered:
  - Quick Sort
  - Merge Sort

- They are usually faster and more efficient

Introduction

- The algorithms uses divide-and-conquer technique to sort a list

- It means the list is divided into 2 sub lists. Each sub list is sorted individually before they are combined into one list

- The sorting of each sub list employs the divide-and-conquer technique again in which it is divided into 2 sub lists and each sub list is sorted individually before they are combined

Introduction

- With this pattern, one should have observed it is recursive in nature

- Hence, in most code implementation, Quick Sort and Merge Sort employs recursive functions

Quick Sort Overview

1. Find the middle location of the list
2. Use that value as “pivot”
3. Put all numbers smaller than the pivot to the left and put all numbers bigger than the pivot to the right
4. Then recursively use quick sort again to sort the left part of the pivot and then right part of the pivot

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

Quick Sort Overview

Find middle, value 8 will be your pivot

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

All numbers will be moved with reference to the pivot

0	1	2	3	4	5	6	7
3	7	8	13	12	30	15	20

Use recursion to repeat the above for lower half and upper half.

0	1
3	7

3	4	5	6	7
13	12	30	15	20

Quick Sort Algorithm

- Cyan highlight denotes unsorted list portion

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

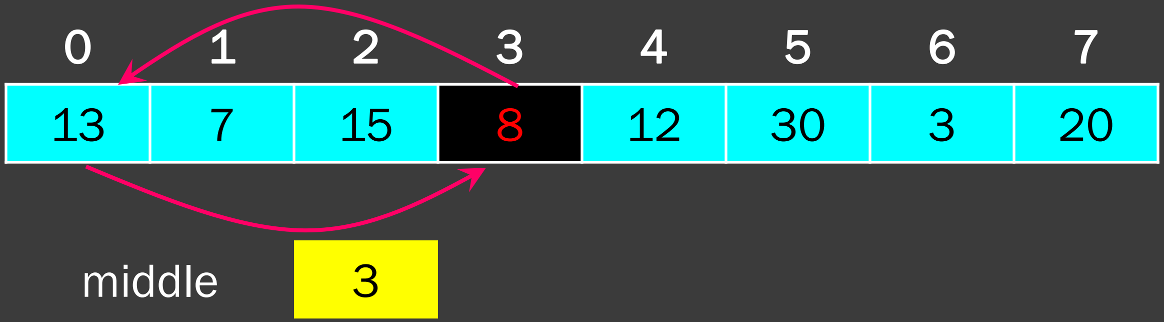
- Identify the pivot

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

middle	3

Quick Sort Algorithm

- Swap pivot with first element of unsorted list



0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

middle 3

- After the swap

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

middle 3

Quick Sort Algorithm

- Start from 2nd element, check whether is smaller than pivot (true)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 0 i 1

- If smaller, increment smallIndex

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 1

Quick Sort Algorithm

- Do a swap (in this case no change)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	1

- Increment i

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	2

Quick Sort Algorithm

- Check whether is smaller than pivot (false)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 2

- Increment i

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 3

Quick Sort Algorithm

- Check whether is smaller than pivot (false)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	3

- Increment i

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	4

Quick Sort Algorithm

- Check whether is smaller than pivot (false)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	4

- Increment i

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	1	i	5

Quick Sort Algorithm

- Check whether is smaller than pivot (false)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 5

- Increment i

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 6

Quick Sort Algorithm

- Check whether is smaller than pivot (true)

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 1 i 6

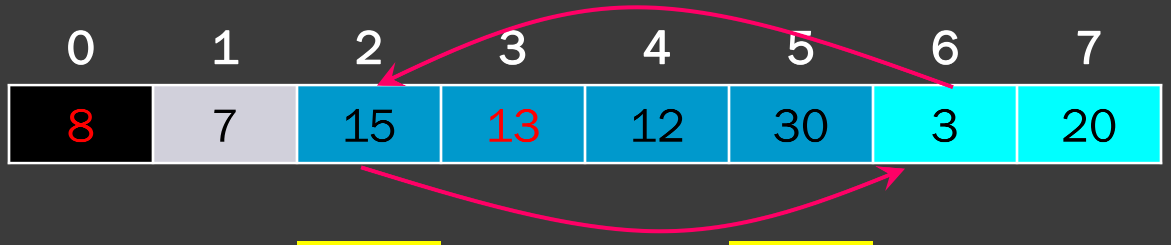
- If smaller, increment smallIndex

0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex 2 i 6

Quick Sort Algorithm

- Do a swap



0	1	2	3	4	5	6	7
8	7	15	13	12	30	3	20

smallIndex	2	i	6

- Increment i

0	1	2	3	4	5	6	7
8	7	3	13	12	30	15	20

smallIndex	2	i	7

Quick Sort Algorithm

- Check whether is smaller than pivot (false)

0	1	2	3	4	5	6	7
8	7	3	13	12	30	15	20

smallIndex	2	i	7

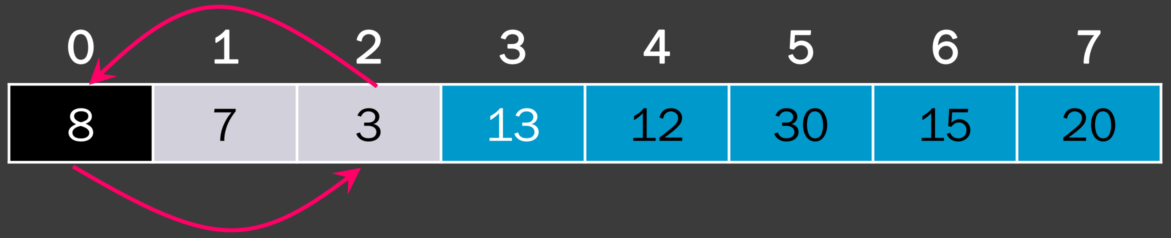
- Increment i (end of array list)

0	1	2	3	4	5	6	7
8	7	3	13	12	30	15	20

smallIndex	2	i	8

Quick Sort Algorithm

- Swap back the pivot with element indexed by smallIndex



0	1	2	3	4	5	6	7
8	7	3	13	12	30	15	20

smallIndex 2 i 8

0	1	2	3	4	5	6	7
3	7	8	13	12	30	15	20

smallIndex 2 i 8

Quick Sort Algorithm

- The general algorithm is as follows:

```
if(the list size > 1)
{
     partition into 2 sub lists
     quick sort lower sub list
     quick sort upper sub list
}
```

Quick Sort Code

```
void quickSort(int data[], int first, int last)
{
      if(first < last)
      {
              int pivotLocation = partition(data,
                                            first,
                                            last);
              quickSort(data, first, pivotLocation-1);
              quickSort(data, pivotLocation+1, last);
      }
}
```

Quick Sort Code

```
int partition(int data[], int first, int last)
{
        int middle = (first + last)/2;
        swap(data, first, middle); // swap the pivot element with first
                                   // element in the array

        int pivot = data[first];
        int smallIndex = first;

        for(int i = first+1; i <= last; i++) //start from 2nd element
        {
                if(data[i] < pivot)
                {
                        smallIndex++;
                        swap(data, smallIndex, i);
                }
        }
        swap(data, first, smallIndex);
        return smallIndex;
}
```

Merge Sort Algorithm

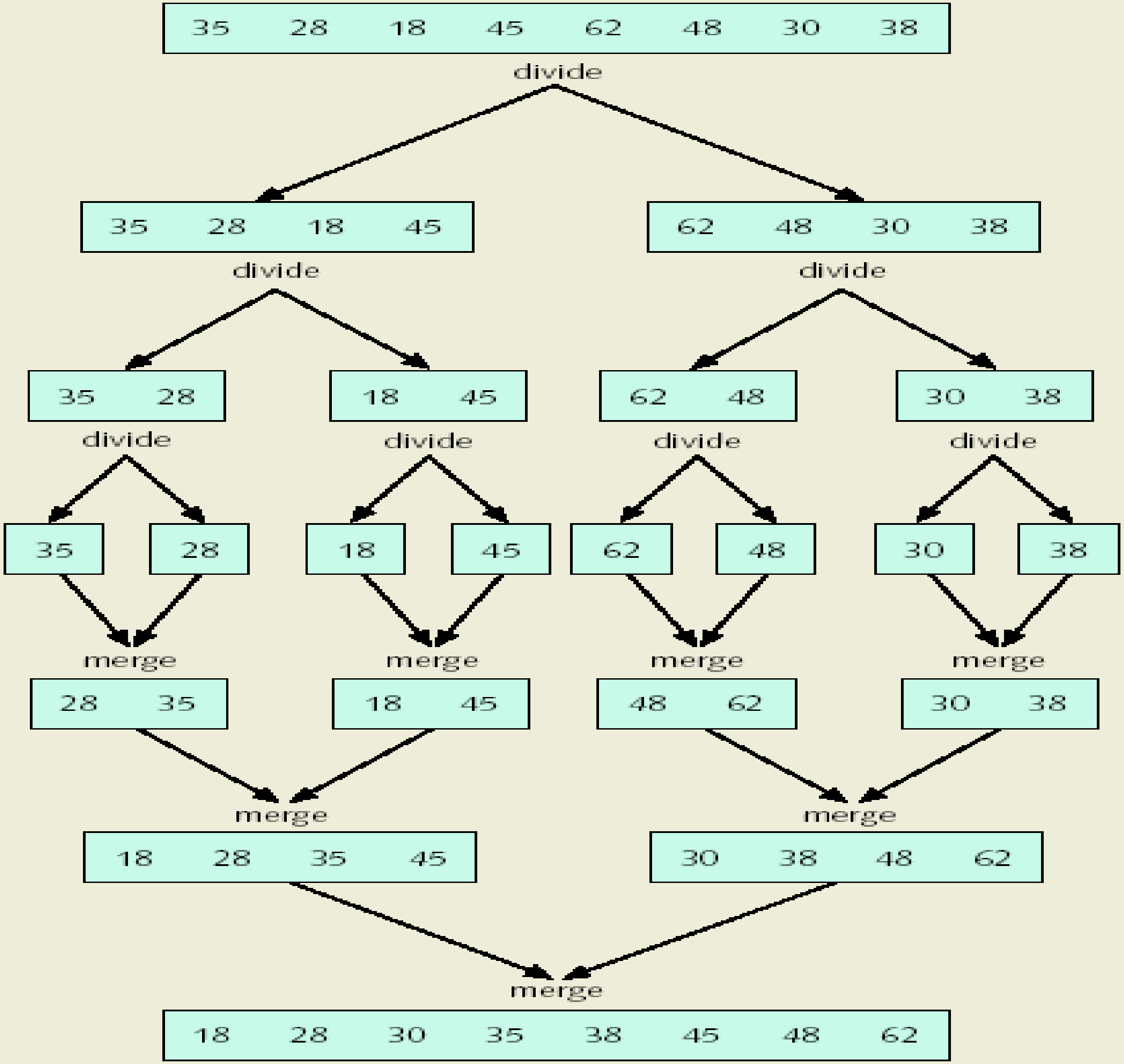
- Almost similar to Quick Sort

- Uses divide-and-conquer in which also partitions the unsorted list into 2 sub lists, sorts the sub lists and then combines them into 1 sorted list

- This algorithm performs better in worst case than Quick Sort

Merge Sort Algorithm

- The sorting takes place during merging of sub lists



35 28 18 45 62 48 30 38

divide

35 28 18 45

62 48 30 38

divide divide

35 28

18 45

62 48

30 38

divide divide divide divide

35 28 18 45 62 48 30 38

merge merge merge merge

28 35

18 45

48 62

30 38

merge merge

18 28 35 45

30 38 48 62

merge

18 28 30 35 38 45 48 62

Merge Sort Algorithm

- The general algorithm is as follows:

```
if(the list size > 1)
{
      Divide the list into 2 sub lists almost equally
      Merge Sort 1st sub list
      Merge Sort 2nd sub list
      Merge the 2 sub lists (main function to sort)
}
```

Merge Sort Code

```
void mergeSort(int data[], int first, int
last)
{
    if(first < last)
    {
        int middle = (first + last)/2;
        mergeSort(data, first, middle);
        mergeSort(data, middle+1, last);
        merge(data, first, middle, last);
    }
}
```

Merge Sort Algorithm

- Cyan highlight denotes unsorted list portion

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

- Identify the middle

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

middle	3

Merge Sort Algorithm

- Since it is recursive, identify the middle again

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

middle 1

- Since it is recursive, identify the middle again

0	1	2	3	4	5	6	7
13	7	15	8	12	30	3	20

middle 0

Merging Code

```
void merge(int data[], int first, int middle, int last)
{
       //temp array to hold 1st list
      int *temp = new int[middle-first+1];

      //i is index for temp array,
      //j is index for 2nd list,
      //k is index for combine list
      int i, j, k;
      for(j = first, i = 0; j <= middle; i++, j++)
      {
            temp[i] = data[j]; //duplicate 1st list
      }
```

Merging Algorithm (Example)

- Duplicate

	first			middle				last
	0	1	2	3	4	5	6	7
data (j)	13	31	32	47	12	30	35	45

	0	1	2	3
temp (i)	13	31	32	47

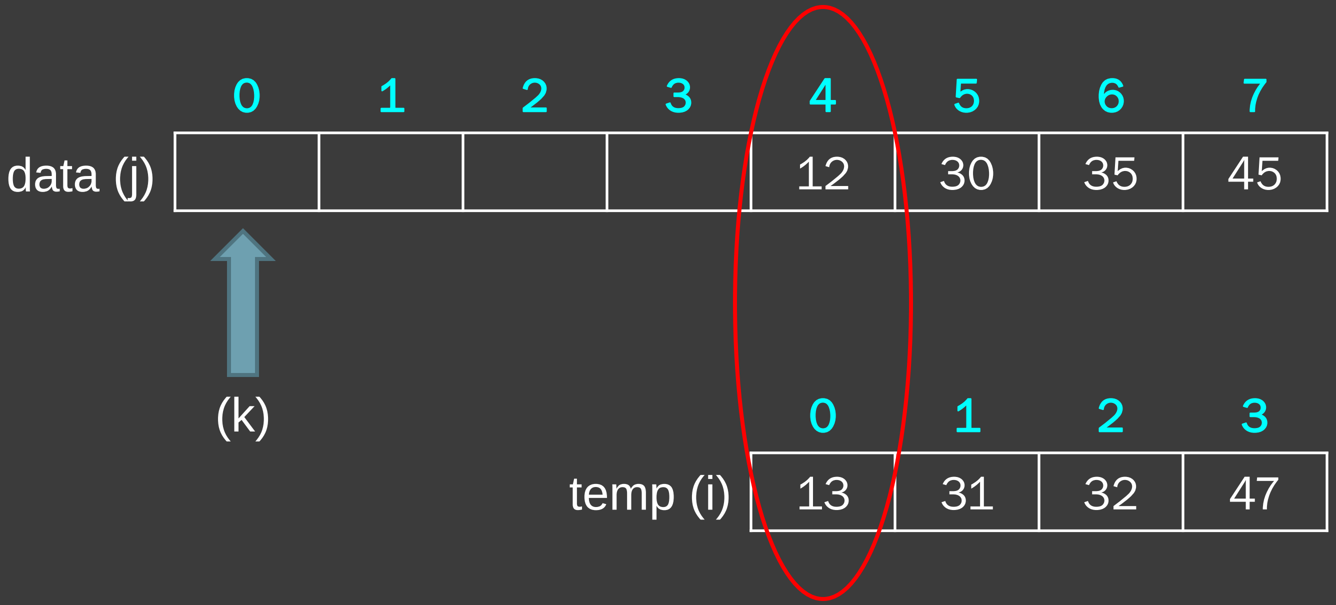
Merging Code (cont.)

```
        i = 0; k = first;
        while (k < j && j <= last)
        {
                //if element from 1st list < 2nd list
                if (temp[i] <= data[j])
                        data[k++] = temp[i++]; //copy from 1st list
                else
                        data[k++] = data[j++]; //copy from 2nd list
        }

        while (k < j) //copy remaining elements in temp, if any
        {
                data[k++] = temp[i++];
        }
        delete [] temp; //remove temp array
}
```

Merging Algorithm

- Compare and copy over



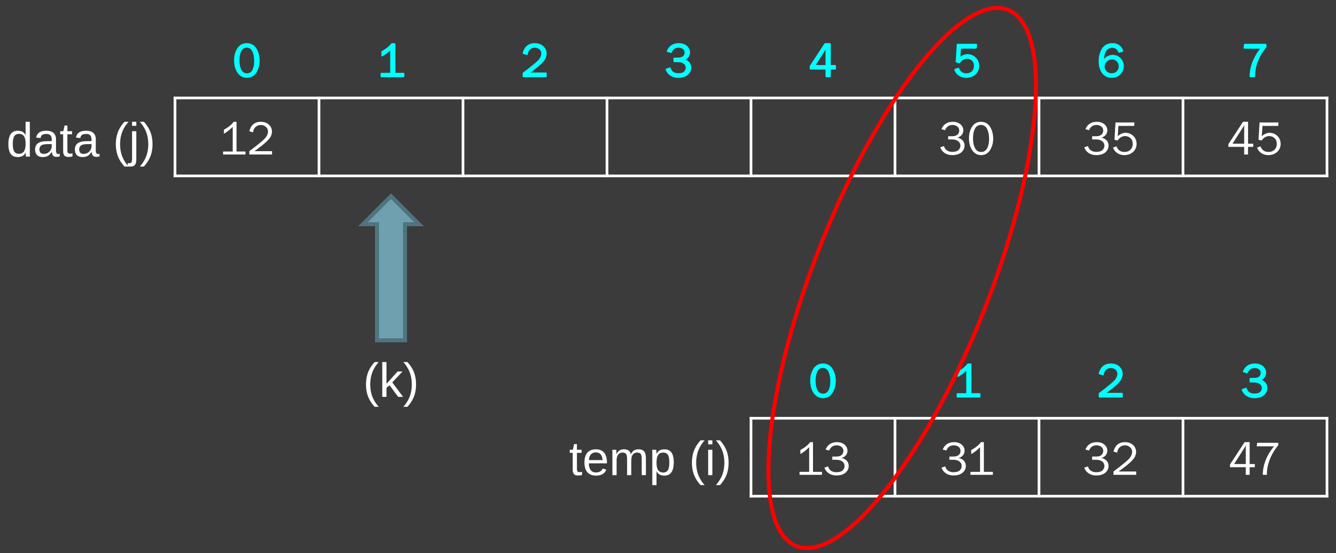
	0	1	2	3	4	5	6	7
data (j)					12	30	35	45

	0	1	2	3
temp (i)	13	31	32	47

//i is index for temp array,
//j is index for 2nd list,
//k is index for combine list

Merge Sort Algorithm

- Compare and copy over



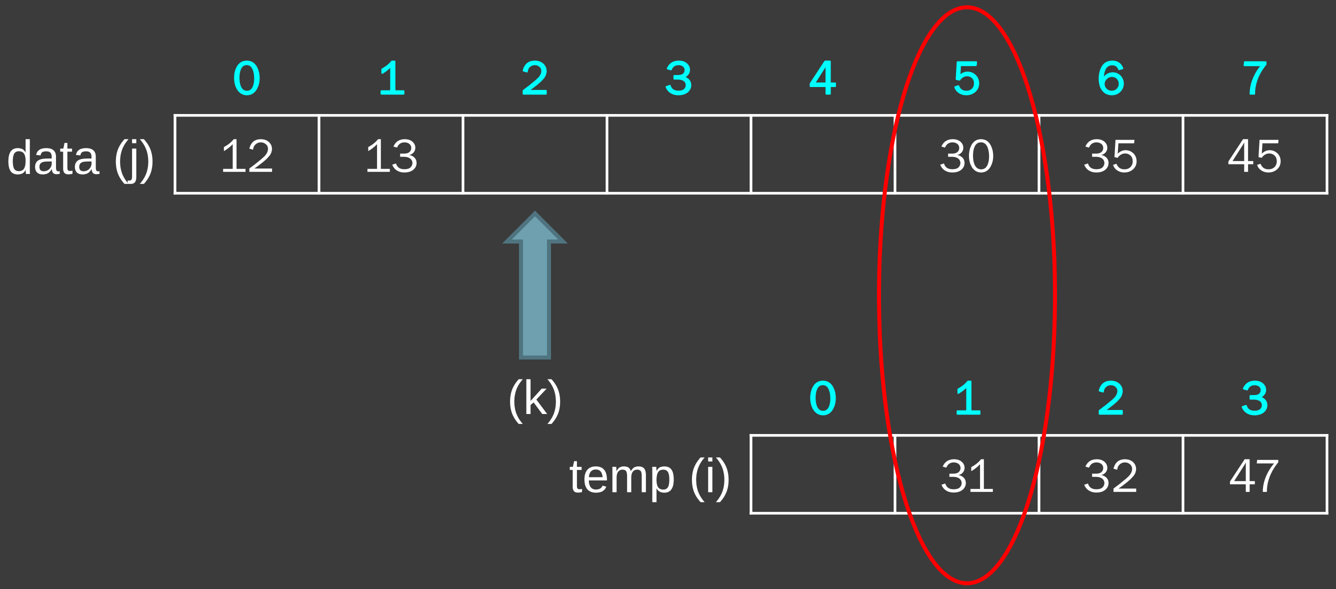
	0	1	2	3	4	5	6	7
data (j)	12					30	35	45

	0	1	2	3
temp (i)	13	31	32	47

//i is index for temp array,
//j is index for 2nd list,
//k is index for combine list

Merge Sort Algorithm

- Compare and copy over



	0	1	2	3	4	5	6	7
data (j)	12	13				30	35	45

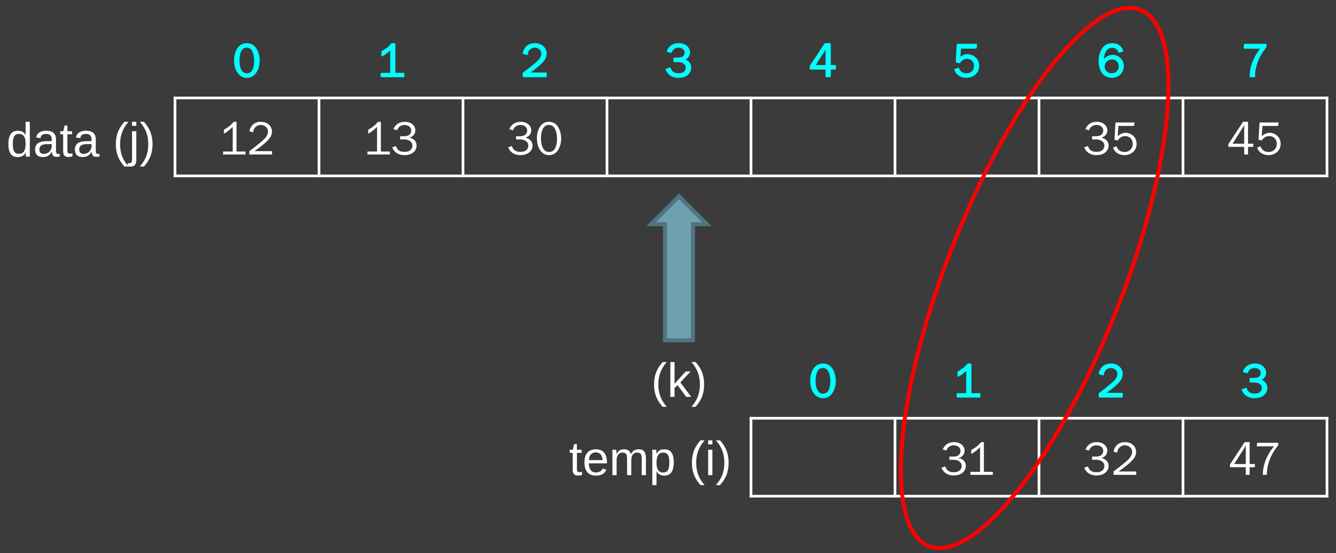
(k)

	0	1	2	3
temp (i)		31	32	47

```
//i is index for temp array,
//j is index for 2nd list,
//k is index for combine list
```

Merge Sort Algorithm

- Compare and copy over



	0	1	2	3	4	5	6	7
data (j)	12	13	30				35	45

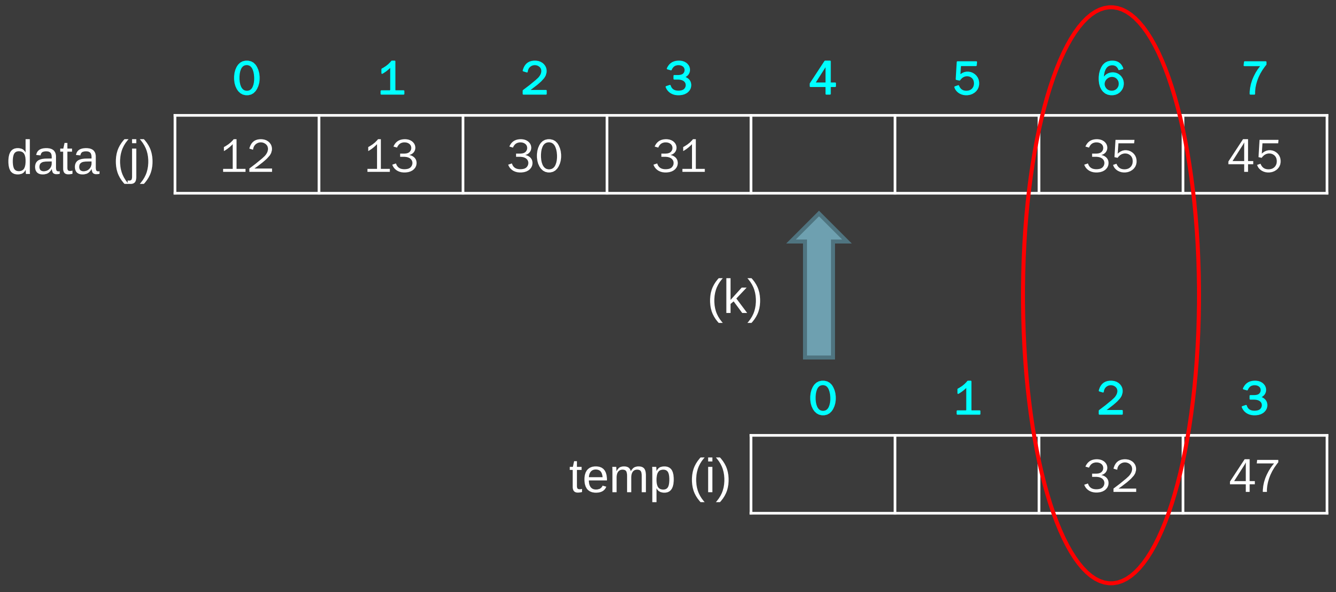
(k)

	0	1	2	3
temp (i)		31	32	47

//i is index for temp array,
//j is index for 2nd list,
//k is index for combine list

Merge Sort Algorithm

- Compare and copy over



	0	1	2	3	4	5	6	7
data (j)	12	13	30	31			35	45

	0	1	2	3
temp (i)			32	47

//i is index for temp array,
//j is index for 2nd list,
//k is index for combine list

Summary

- Understand and implement Quick Sort and Merge Sort Algorithms