Data Structure & Algorithm Basic Lab – week 11+12

Topics of this week

- Advanced Sorting Algorithm
  - Quick sort
  - Merge sort
  - Recursive processing
- Exercises

Quicksort Algorithm

Given an array of *n* elements (e.g., integers):

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Quicksort two sub-arrays
  - Return results

Example

- We are given array of n integers to sort:

Quick Sort (Hoare)

- Given $(R_0, R_1, \ldots, R_{n-1})$

  K_i: pivot key

  if K_i is placed in S(i),

  then $K_j \le K_{s(i)}$ for $j < S(i)$,

  $K_j \ge K_{s(i)}$ for $j > S(i)$.

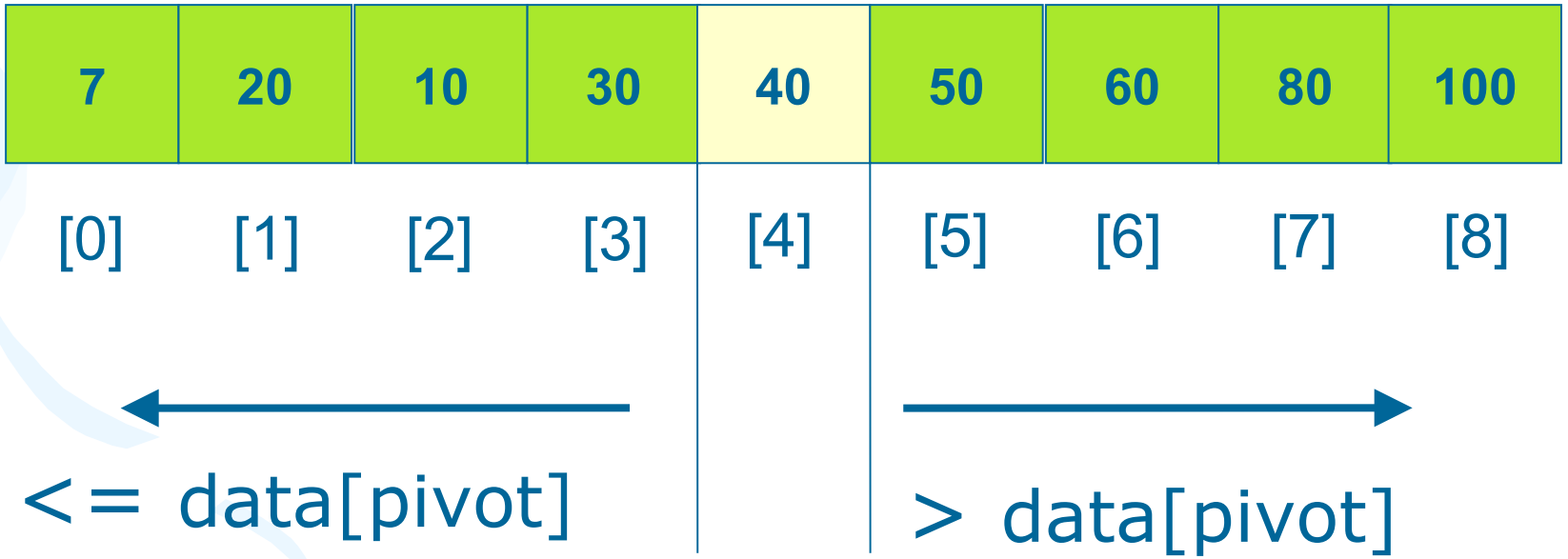
- $R_0, \ldots, R_{S(i)-1}, R_{S(i)}, R_{S(i)+1}, \ldots, R_{S(n-1)}$

two partitions

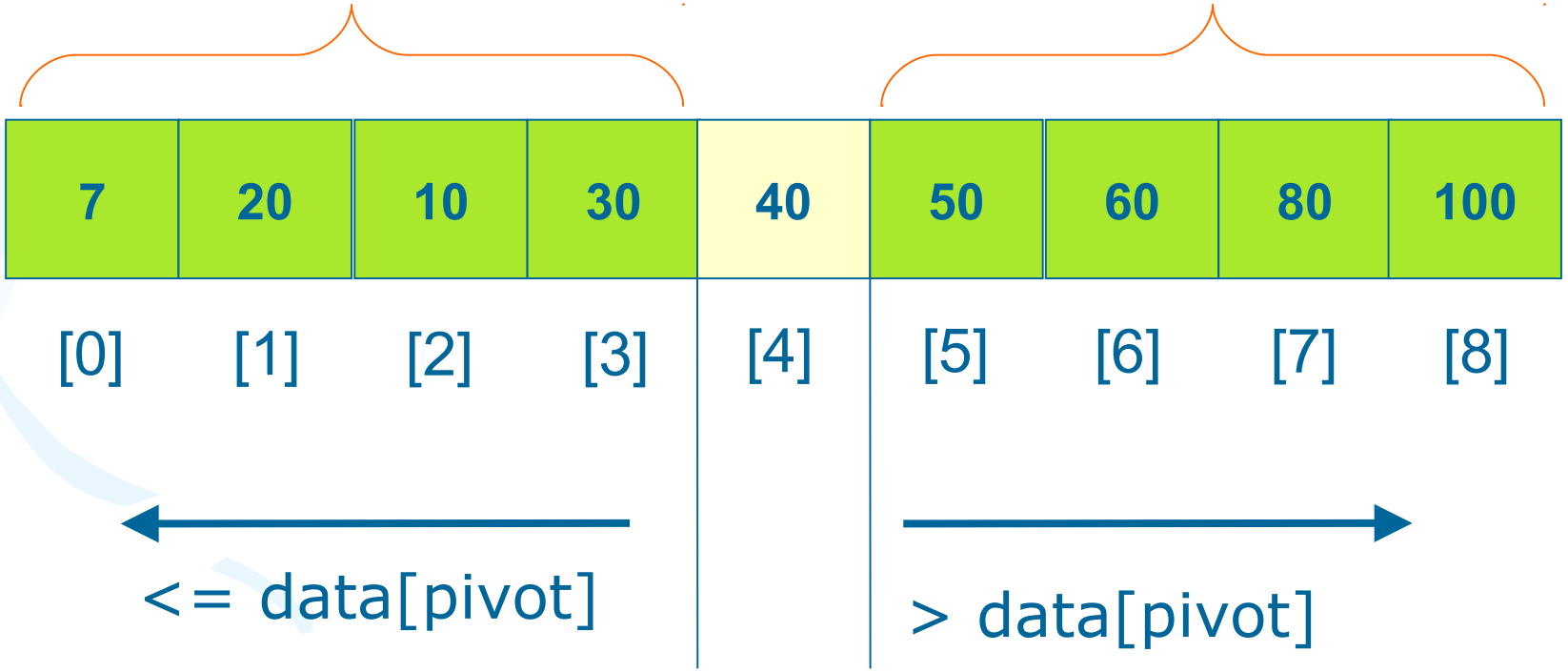
Partitioning Array

- Given a pivot, partition the elements of the array such that the resulting array consists of:
- 1. One sub-array that contains elements >= pivot
- 2. Another sub-array that contains elements < pivot
- The sub-arrays are stored in the original data array.
- Partitioning loops through, swapping elements below/above pivot.

Partition Result



Recursion: Quicksort Subarrays



Example for Quick Sort

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	left	right
{26	5	37	1	61	11	59	15	48	19}	0	9
{11	5	19	1	15}	26	{59	61	48	37}	0	4
{1	5}	11	{19	15}	26	{59	61	48	37}	0	1
1	5	11	15	19	26	{59	61	48	37}	3	4
1	5	11	15	19	26	{48	37}	59	{61}	6	9
1	5	11	15	19	26	37	48	59	{61}	6	7
1	5	11	15	19	26	37	48	59	61	9	9
1	5	11	15	19	26	37	48	59	61		

Quick Sort

```
void quicksort(element list[], int left,
                                int right)
{
  int pivot, i, j;
  element temp;
  if (left < right) {
    i = left;     j = right+1;
    pivot = list[left].key;
    do {
      do i++; while (list[i].key < pivot);
      do j--; while (list[j].key > pivot);
      if (i < j) SWAP(list[i], list[j], temp)
    } while (i < j);
    SWAP(list[left], list[j], temp);
    quicksort(list, left, j-1);
    quicksort(list, j+1, right);
  }
}
```

Exercise 11-1: Quick sort

- We assume that you make a mobile phone's address book.
- At the very least, you should declare the structure that can store "name", "phone number" and "e-mail address". And, you should declare the array that can store about 100 data that have this structure.
- You write a program that reads about 10 data from an input file to the array and writes the data to an output file after sorting in ascending order for name.
- You must use Quick sort for sorting.

Exercise 11-2

- Initiate an array of n random integers. n is entered by user.
- Sort the array with the insertion sort
- And using quicksort
- Compare the execution time of two algorithms.
- Run the program with various values of n to view the effect.

Exercise 11-3 Combination of quick sort and insertion sort

- When a program sorts a little number of the data, a program using insertion sort is faster than a program using quick sort and so on. So, a program sorts efficiently, if a program changes sorting algorithms by the number of data.
- You write a function that selects sorting algorithms – If number of the data is more than x numbers, the function selects quick sort. If not so, it selects insertion sort.
- Note: get the number “x” as the program argument.
- Read the text file that has more than 100 characters, sort the first 100 characters, and show the result by standard output.

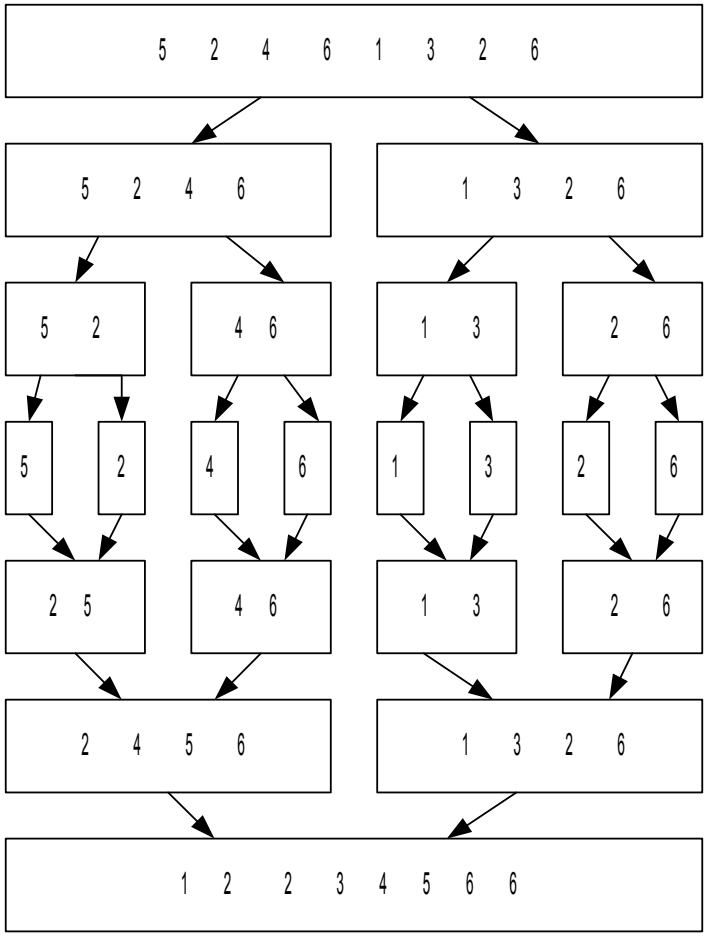
Merge Sort

- Problem: Given *n* elements, sort elements into non-decreasing order
- Apply divide-and-conquer to sorting problem
  - If n=1 terminate (every one-element list is already sorted)
  - If n>1, partition elements into two sub-arrays; sort each; combine into a single sorted array

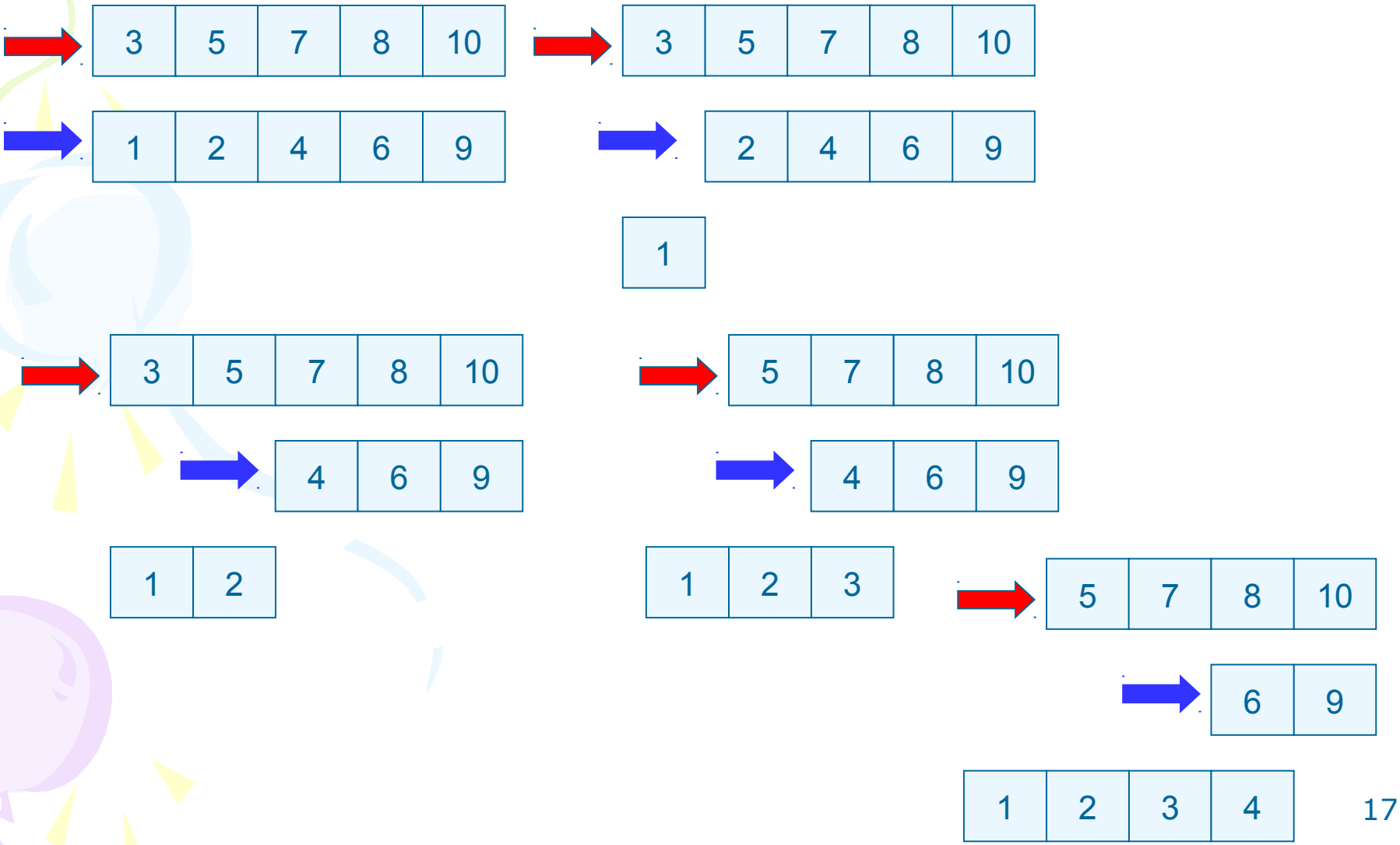
Algorithm

```
MergeSort (E[ 0 .. N])
   if N < threshold
      InsertionSort ( E[0..N] )
   else
      copy E[0.. N/2] to U[0.. N/2]
      copy E[N/2 .. N] to V[0 .. N-N/2]
      MergeSort(U[0 .. N/2])
      MergeSort(V[0 .. N-N/2])
      Merge( U[0 .. N/2], V[0 .. N-N/2},
   E[0 .. N] )
```

Merge Sort: Example



Process of merge



Merge algorithm

```
Merge(U[0..m],V[0..n],E[0..n+m])
  i =  0 ,  j = 0
  k = 0
  while k <  n+m
    if U[i] < V [j]
      E[k] = U[i] , i++
    else
      E[k] = V[j] , j++
    k++
```

Exercise: 11-3 Merge sort

- We assume that you make a mobile phone's address book.
- At the very least, you should declare the structure that can store "name", "phone number" and "e-mail address". And, you should declare the singly-linked list that can store about 100 data that have this structure.
- You write a program that reads about 10 data from an input file to the list and writes the data to an output file after sorting in ascending order for name.
- You must use Merge sort for sorting.

Exercise: Recursive Processing

- Write a recursive algorithm for dealing a deck of cards. The parameters should be (i) the deck of undealt cards, and (ii) the person who is to receive the next card. Assume:
  - the players are seated around a table;
  - dealing begins with the player to the dealer's left;
  - each dealing step involves dealing one card to a player, then the dealer's attention moves to the next player to the left; and
  - dealing continues until no cards are left in the deck.

Hint

```
function dealCards (deck, person)
{
    if (deck is empty)
        return;
    deal top card from deck to person;
    dealCards (rest of deck, personLeftOf(person));
}
```

Exercise: Recursive Processing

- Write a recursive function **void recurTriangle (int n, char ch)** which prints out an upside-down triangle. The parameter *ch* is the character to be used for drawing the triangle, and *n* is the number of characters on the first row. For example, if *n* is 7 and *ch* is **'+'**, then the output of the function should be:

```
+++++++
++++++
+++++
++++
+++
++
+
```

```
void recurTriangle(int n, char ch)
{
int i;

   if(n > 0)
        {
        for(i = 0; i < n; i++) printf("%c", ch);
        printf("\n");
        recurTriangle(n-1, ch);
        }
}
```

Exercise 11-4: String sorting

- Write a program that sorts strings with quick sort by alphabetical order based on the following instructions.

I. Compare the character strings

- Write the function “preceding()” to search which of two character strings comes before by alphabetical order.

  ```
  int preceding(char *first, char *second)
  ```

- A return value is by alphabetical order
  - Case that the character string of the argument “first” is before the character string of the argument “second” : 1
  - Case that the character string of the argument “first” is equal to the character string of the argument “second” : 0
  - Case that the character string of the argument “first” is after the character string of the argument “second” : -1

II. Input the character string from the file

- Write the function "`setup_nameList()`" to read the name of more than 2 persons and less than 25 persons from the file and set them to the array "`nameList[]`" of a character string (in fact, the array of the pointer to the character string)

```
int setup_nameList(char *namelist[], char *filename)
```

III. Implement Quicksort

- Write the function “qsort_name()” to sort the character string of the array “namelist[]” by alphabetical order with quick sort using the function you made ever.

Homework

- Write a quicksort function to sort a singly linked list. Add this function to the linked list library.
- Hint: You should have a function
  - *for getting the Nth element in the linked list*
  - *Swapping two nodes in listed*

Improve Quicksort

- Change the Pivot Selection strategy:
  - random element
  - median of three strategy