DATA STRUCTURES AND ALGORITHMS

Sorting algorithms

(insertion sort, bubble sort, selection sort)

Summary of the previous lecture

- Recursion
  - Definition
- Examples
  - Factorial
  - Fibonacci
  - Hanoi tower
  - Printing of Linked list numbers
- Homework
  - Assigning array element values recursively
  - Recursive printing (normal and reverse order)

Sorting

We sort many things in our everyday lives:

- Files in the directory
- Directories in the local disk
- Daily tasks
- Books in the shelves

Sorting is also one of the most frequently performed computing tasks.

Formulation of the sorting problem

Assume that we have data set A = (a_1; a_2 ….. a_N).

We can compare these data: $a_i \leq a_j$, when $i \mathrel{!}= j$.

We can to rearrange set of data so that A’ = (a'_1; a'_2 ….. a'_N)

$$a_i \leq a_j \quad \text{where } i = 0, 1, 2 \ldots N$$

Sorting problem allows input with two or more records that have the same key value.

Sorting algorithm is said to be **stable** if it does not change the relative ordering of records with identical key values.

Topology sort

Assume that we have set of tasks:

$$U = (u_1, u_2, \ldots, u_N)$$

If relations between tasks are as follows:

$$C = (u_{i_1} \prec u_{j_1},\ u_{i_2} \prec u_{j_2},\ \ldots,\ u_{i_M} \prec u_{j_M}).$$

$$u_i^{'} \prec u_j^{'}, \qquad i < j.$$

then we have case of **topology sort.**

Example

Assume that we have 9 tasks

$$U = \left(d_1, d_2, d_3, p_1, p_2, p_3, u_1, u_2, u_3 \right)$$

and relation between tasks are:

$$C = \left(d_j \prec p_j, \;\; p_j \prec u_j, \;\; j = 1, 2, 3 \right).$$

Set of sorted tasks can be:

$$U' = \left(d_1, p_1, u_1, \;\; d_2, p_2, u_2, \;\; d_3, p_3, u_3 \right).$$

$$U'' = \left(d_1, d_2, d_3, \;\; p_3, p_2, p_1, \;\; u_2, u_3, u_1 \right).$$

Time complexity of sorting

When analyzing sorting algorithms, it is traditional to measure the **number of comparisons made between keys**.

This measure is usually closely related to the running time for the algorithm and has the advantage of being machine and datatype independent.

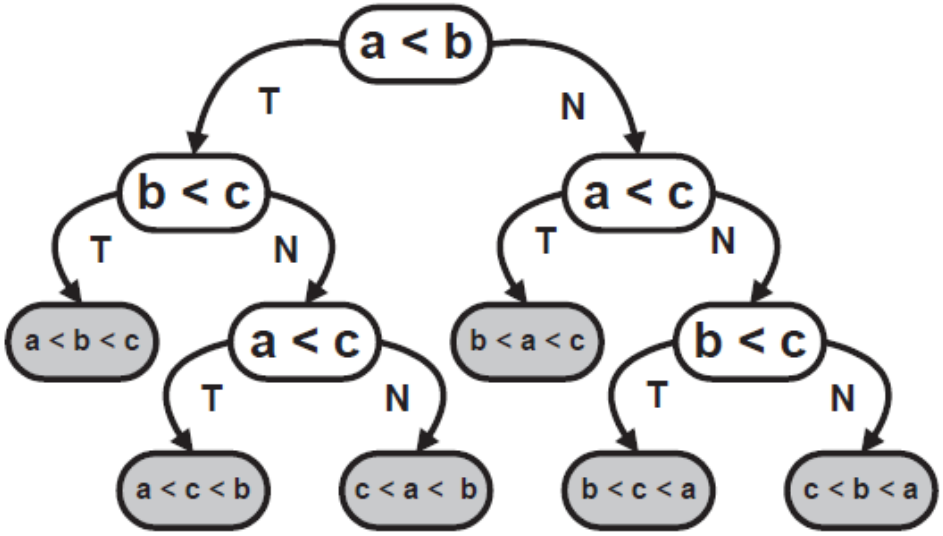
However, in some cases records might be so large that their physical movement might take a significant fraction of the total running time. If so, it might be appropriate to measure the number of **swap operations** performed by the algorithm.

Time complexity of sorting

Assume that we have 3 elements: a, b, c.

Sorting of these elements can be produced as binary tree.

Height of the tree is 4 and represent the worst case.



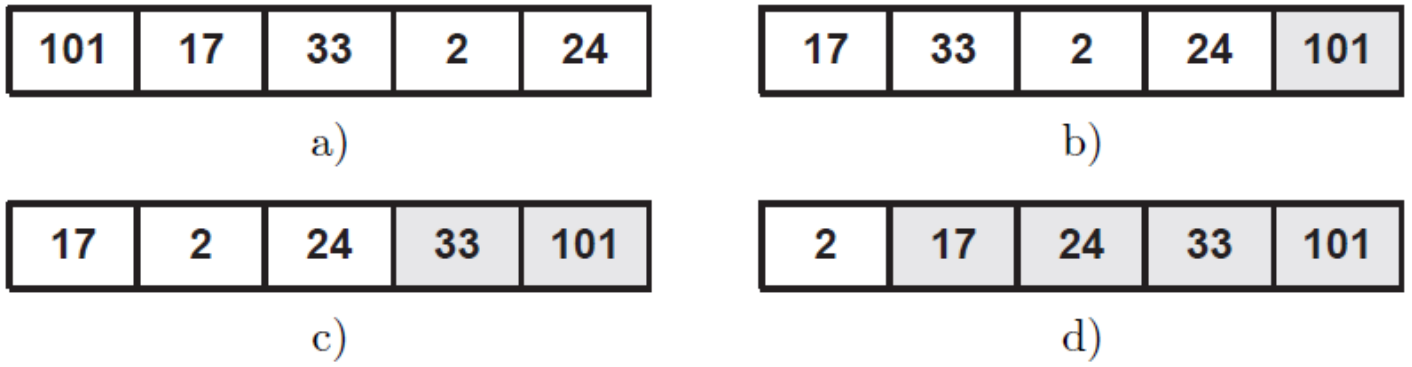
Number of comparisons K can be calculated as:

$$2^K \geq N!$$

Bubble sort

It is a relatively slow sort, but stable sorting algorithm. The sorting procedures is similar to rise air bubble in the water.

During each iteration one element is placed in it's proper position.



Bubble sort

```
void Bubble_Sort (int array[ ], int length)
{
     int i, j, flag = 1;          // set flag to 1 to start first pass
     int temp;                    // holding variable
     for (i = 0; (i < length -1 ) && flag; i++)
    {    flag = 0;
           for ( j = 0; j < (length - i -1); j++)
           {      if ( array [j+1] > array[j])
                 {     temp  = array [j];            // swap elements
                       array [j] = array [j+1];
                       array [j+1] = temp;
                        flag = 1;              // indicates that a swap occurred.
                 }
           }
    }
    return;
}
```

Time complexity

Time complexity of Bubble sort algorithm doesn't depend on initial situation in data series (presorted, or already sorted).

Number of comparison operations is equal to:

$$L_B(N) = \frac{N(N-1)}{2},$$

in any case of data series sorting.

Number of swapping operations:

Best case $S_G(N) = 0\,.$ Worst case $S_G(N) = \frac{N(N-1)}{2}\,.$

Time complexity is $O(n^2)$ in any case.

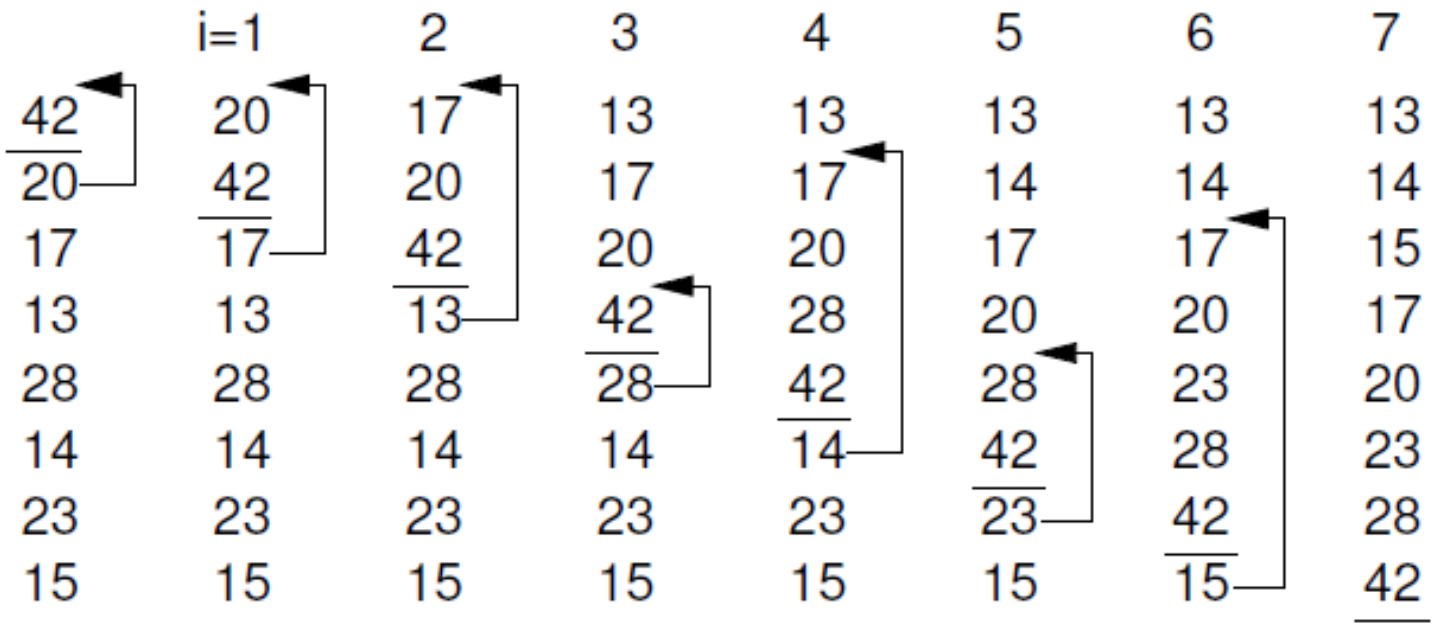
Insertion sort

Imagine that you have a **stack of phone bills** from the past two years and that you wish **to organize them by date**.

A fairly natural way to do this might be to look at the first two bills and put them in order. Then take the third bill and put it into the right order with respect to the first two, and so on.

As you take each bill, you would add it to the sorted pile that you have already made.

Insertion sort



Insertion sort

```
void insertion_Sort (int *array, int length) {
    int i, j, tmp;
    for ( i = 1;  i < length;  i++) {
        j = i;
        while ( j > 0  &&  (array [ j – 1 ]  > array [ j ] ))
        {     tmp = array[ j ];
              array[j] = array[ j – 1 ];
              array[ j – 1 ] = tmp;
              j - -;
        }
    }
}
```

Time complexity

Time complexity of Insertion sort algorithm depends on initial situation in data series (presorted, or already sorted).

Number of comparison operations is equal to:

- $L_G(N) = N - 1.$ in best case (sorted list)
- $L_B(N) = \sum_{i=2}^{N} i = \frac{(N+2)(N-1)}{2} = \frac{N^2}{2} + \mathcal{O}(N)$ in worst case
- $L_V(N) = \frac{N^2}{4} + \mathcal{O}(N)$ in average case

Selection sort

Consider again the problem of sorting a pile of phone bills for the past year.

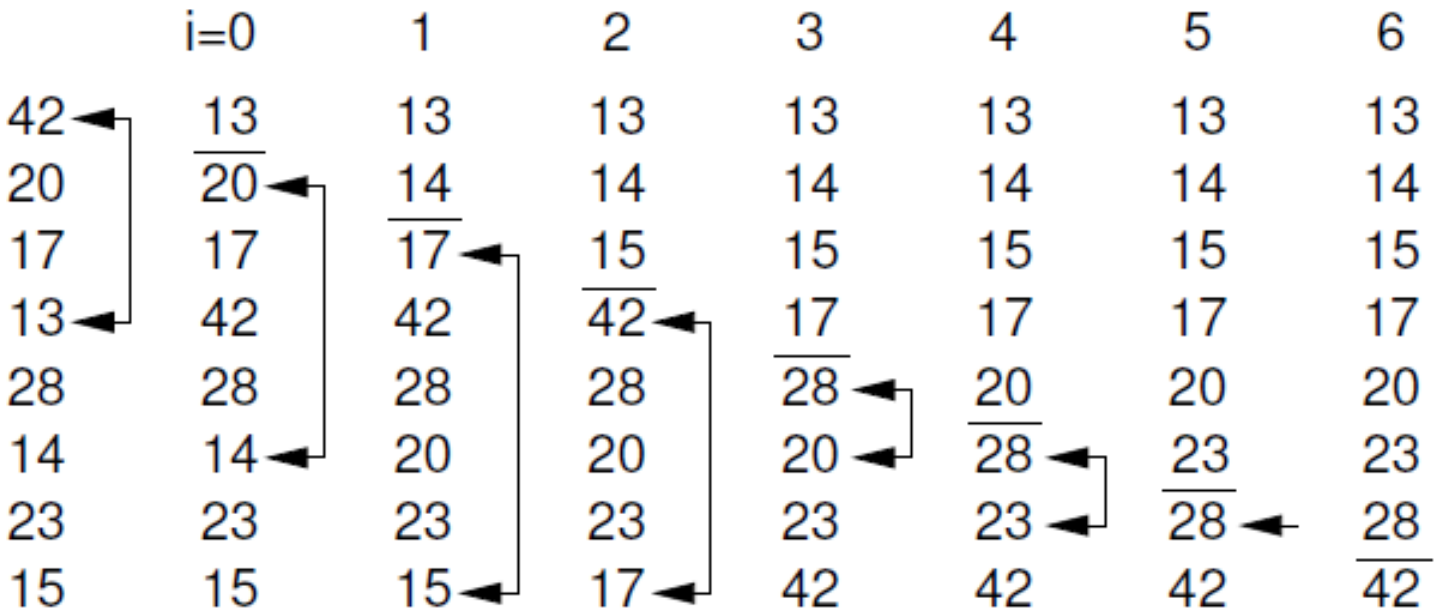
Another intuitive approach might be to look through the pile until you find the bill for January, and pull that out.

Then look through the remaining pile until you find the bill for February, and add that behind January.

Proceed through the ever-shrinking pile of bills to select the next one in order until you are done.

Selection Sort is particularly advantageous when the cost to do a swap is high, for example, when the elements are long strings or other large records.

Selection sort



Selection sort

```
void Selection_Sort ( int *array,  int length)
{
    if ( length <= 1 )
    return;
  for ( int i = 0; i < length – 1; i++)
  { int iSmallest = i;
      for ( int  j = i+1; j < length; j++ )
      {    if (array[iSmallest]  > array[j])
             iSmallest = j;
      }
    int tmp  = array[iSmallest];
    array[iSmallest] = array[i];
    array[i] = tmp;
  }
}
```

Time complexity

Number of swap operations used for Selection sort algorithm depends on initial situation in data series (presorted, or already sorted).

Number of comparison operations is equal to:

$$L(N) = \sum_{i=1}^{N-1} (N - i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}.$$

in any case of data series sorting.

Number of swapping operations:

Best case $S_G(N) = 0\,.$ Worst case $S_B(N) = N - 1.$

Time complexity is $O(n^2)$ in any case.

Practice tasks

No.1

Measure the **real (empirical) running time** of the selection, insertion and bubble sort algorithms and compare with asymptotic evaluation. Use random number generator. Number of the elements ~ 10^6.

No.2

Write a code for finding **greatest common divisor** (gcd) of two number. Two algorithms are possible:

1. Euclidean algorithm (gcd divide difference of two numbers)
2. Factorization (using prime numbers)

Example

```
#include <time.h>       /* clock_t,  clock, CLOCKS_PER_SEC */

int main ()
{   clock_t time1, time2;
    float total_time;

                time1 = clock();
                 ....
                time2 = clock() - time1;

    total_time = ((float)time2) / CLOCKS_PER_SEC;
    return 0;
}
```