COS 460 – Algorithms (week 01)

Spring 2018, Wednesday, 10:45 – 12:00 and 14:15 – 15:30, MB 116

Prof. Kelevedjiev (Келеведжиев)

office: MB 201
office hours: Wednesday, 13:00-14:00
e-mail: ekelevedjiev@aubg.bg, keleved@math.bas.bg

Prerequisites:

- Fundamental Data Structure
- Programming in C++

Goals:

- Theory of design and analysis of computer algorithms
- Practice of implementation in programming language C++

SYLLABUS (see)

Source of information: the Internet
Type in any search engine the word "Algorithm" and you will find many fine results

- http://en.wikipedia.org
- http://mathworld.wolfram.com
- http://www.webopedia.com
- http://www.nist.gov/dads (Dictionary of Algorithms and Data Structures at US National Institute of Standards and Technology)

ALGORITHM

- In mathematics, computing, and related disciplines, an **algorithm** is a procedure (a finite set *of well-defined* instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state.
- The computational complexity and efficient implementation of the algorithm are important in computing, and this depends often on the suitable choice of data structures.
- Informally, the concept of an algorithm is often illustrated by the example of a recipe, although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as comparison).
- Algorithms can be composed to create more complex algorithms.
- Most algorithms can be directly implemented by **computer programs**; any other algorithms can at least in theory be *simulated* by computer programs. In many programming languages, algorithms are implemented as functions or procedures.
- Algorithms can be expressed in many kinds of notation, including natural languages, *pseudocode*, and programming languages.

EXAMPLE

One of the simplest algorithms is to find the largest number in an (unsorted) list of numbers. The solution necessarily requires looking at every number in the list, but only once at each. From this follows a simple algorithm, which can be stated in a high-level description English prose, as:

High-level description:
Assume the first item is the largest.
Look at each of the remaining items in the list and if it is larger than the largest item so far, make a note of it.
The last noted item is the largest in the list when the process is complete.

(Quasi-) Formal description:
Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

```
Algorithm LargestNumber
  Input: A non-empty list of numbers $L$.
  Output: The largest number in the list $L$.

  $largest \leftarrow L_0$
  for each next item in the list $L$, do
    if the $item > largest$, then
      $largest \leftarrow$ the item
  return largest
```

C++ implementation:

```
#include<iostream>
using namespace std;

int item, largest;

int main()
{
 cin >> largest;
 while(cin >> item)
  if(largest<item) largest=item;
 cout << largest;
}
```

A TASK AND AN INSTANCE OF A TASK

The set of all possible instances of a given task that can be solved by a given implementation of an algorithm is often **restricted**.

Example: a task - find the largest number in a given list of numbers.
An instance – find the largest number in the following list of numbers: 1,2,3,4,5.

In the above example, restrictions may concern:

- The length (i.e. number of elements) of the list
- The range in which each element of the list can be appeared.

Classification of algorithms

by field of study

Every field of science has its own problems and needs efficient algorithms. Related problems in one field are often studied together. Some example classes are:

search algorithms,
sorting algorithms,
merge algorithms,
numerical algorithms,
graph algorithms,
string algorithms,
computational geometric algorithms,
combinatorial algorithms,
data compression
parsing techniques.

by design paradigm

divide and conquer
dynamic programming
greedy method
linear programming
search and enumeration

probabilistic
genetic
heuristic

by implementation

recursion or iteration
serial or paralle
deterministic or non-deterministic
exact or approximate

HISTORY NOTES

Al-Khwārizmī , born c. 780, Baghdad, Iraq (or, Khwarizm (now Khiva, Uzbekistan)); died c. 850

In the 12th century a work by al-Khwārizmī introduced Hindu-Arabic numerals and their arithmetic to the West. It is preserved only in a Latin translation, "Algoritmi de numero Indorum" ("Al-Khwārizmī Concerning the Hindu Art of Reckoning"). From the name of the author, rendered in Latin as algoritmi, originated the term **algorithm**.

Timeline: Before computers:
C. 1600 BC - Babylonians develop earliest known algorithms for factorization and finding square roots
C. 300 BC - Euclid's algorithm
C. 200 BC - the Sieve of Eratosthenes
263 - Gaussian elimination described by Liu Hui
Between 813 and 833 Al-Khawarizmi described an algorithm for solving Linear equations and Quadratic equations; the word *algorithm* comes from his name
1614 - John Napier develops method for performing calculations using logarithms
1671 - Newton-Raphson method developed by Isaac Newton
1805 - FFT-like algorithm known by Carl Friedrich Gauss
1926 - Boruvka's algorithm
1934 - Delaunay triangulation developed by Boris Delaunay
1936 - Turing machine, an abstract machine developed by Alan Turing, with others developed the modern notion of *algorithm*.

Timeline: Computer epoch:
1945 - Merge sort developed by John von Neumann
1947 - Simplex algorithm developed by George Dantzig
1952 - Huffman coding developed by David A. Huffman
1956 - Kruskal's algorithm developed by Joseph Kruskal
1957 - Prim's algorithm developed by Robert Prim
1957 - Bellman-Ford algorithm developed by R. Bellman and L. R. Ford
1959 - Dijkstra's algorithm developed by Edsger Dijkstra
1962 - Quicksort developed by C. A. R. Hoare
1962 - Bresenham's line algorithm developed by Jack E. Bresenham
1972 - Graham scan developed by Ronald Graham
1973 - RSA encryption algorithm discovered by Clifford Cocks
1976 - Knuth-Morris-Pratt algorithm developed by Donald Knuth and Vaughan Pratt and independently by J. H. Morris
1977 - Boyer-Moore string search algorithm for searching the occurrence of a string into another string.
1977 - RSA encryption algorithm rediscovered by Ron Rivest, Adi Shamir, and Len Adleman
1977 - LZ77 algorithm developed by Abraham Lempel and Jacob Ziv
1979 - Khachiyan's ellipsoid method developed by Leonid Khachiyan

Future:
20th century science is formula based
21st century science is algorithm based

Impact of Algorithms:
Internet. Web search, packet routing, distributed file sharing.
Biology. Human genome project, protein folding.
Computers. Circuit layout, file system, compilers.
Computer graphics. Movies special effects, video games.
Security. Cell phones, e-commerce, voting machines.
Multimedia. CD player, DVD, MP3, JPG, DivX, HDTV.
Transportation. Airline crew scheduling, map routing.
Physics. N-body simulation, particle collision simulation
And many others

C++ REVISITED

- Main function (Do you remember "Hello, word!" program?)
- Variables, arrays, strings
- Control flow: **`if`**, **`for`**, **`while`**
- Standard input and standard output (**`cin`** and **`cout`**)

Swap two values. Let the values of some two variables `a` and `b` be equal to 3 and 4, respectively. How can we exchange variables' values? Does the sequence `a=b; b=a;` perform the desired exchange? Explain why the task can be solved by introducing of an additional variable:

```
int c=a; a=b; b=c;
```

Use STL (Standard Template Library). It is an advantage of C++. We will use this library in a reasonable way. The first use: the **`swap`** function:

```
#include<iostream>
using namespace std;
int main()
{
   int a,b;
   cin >> a >> b;
   swap(a,b);
   cout << a << " " << b << endl;
   system("pause");
   return 0;
}
```

Explore the above program using your **C++** programming environment. Write, compile and execute. Use the “console application option” (or an empty project in MS Visual C++). Is it necessary to make some additions and/or changes in the source code?

Use the standard input and output in the following way:

```
Your_program < your_input.txt > the_output.txt
```

ARRAYS

An array is a fixed-length structure for storing multiple values of the same type. The elements of the array are placed in contiguous memory locations and each element can be individually referenced by an index. Examples of array's definitions:

```
double a[100]; int b[200]; char c[300];
double d[]={1.0, 2.5, 3.6};
```

```
#include<iostream>
using namespace std;

const int N=100;
int a[N];
int n;

int main()
{ cout << "n: "; cin >> n;
  int i;
  for(i=0;i<n;i++)
   {cout << "a[" << i << "]"; cin >> a[i];}
  cout << "The array is:\n";
  for(i=0;i<n;i++)
   {cout << "a[" << i << "]="<< a[i] << "\n";}
  return 0;
}
```

Let an array is defined by `int a[N]`.

Remember: This array has `N` elements and they are: `a[0]`, `a[1]`, ..., `a[N-1]` .

Remember: Do not use `a[i]`, unless the value of `i` is in the range `0`, ..., `N-1`.

Simple problems, involving arrays:

Find the sum of all elements in the array:

```
int s=0;
for(int i=0; i<n; i++) s += a[i];
cout << s;
```

Find the maximal value among array's elements:

```
int max=a[0];
for(int i=1; i<n; i++)
 if(max<a[i]) max=a[i];
cout << max;
```

Check if a given value presents among the values of array's elements:

```
int v; cin >> v;
int k=-1;
for(int i=0;i<n;i++)
  if(v==a[i]) {k=i; break;}
if(k>-1) cout << k << " " << a[k];
else cout << "not found";
```

Insert the value `v` into the array at a place `i`

```
void insert(int i, int v)
{
 for(int j=n;j>i;j--) a[j]=a[j-1];
 a[i]=v;
 n++;
}
```

Sorting algorithms: Selection sort.

```
void sort()
{
 for(int i=0;i<n;i++)
 {
  int m=a[i];
  int p=i;
  for(int j=i+1;j<n;j++)
   if(a[j]<m){m=a[j];p=j;}
  a[p]=a[i];
  a[i]=m;
 }
}
```

Generating random numbers. Each time calling `rand()` you get a random integer number in a range 0...`MAXINT`. Example:

```
for(int i=0;i<n;i++) a[i]=rand()%100;
```

The algorithm that generates random numbers is *deterministic*, i.e. each time you run the program, the sequence will be the same. In order to prevent this, you may call the `srand(value)` function using as a `value` some number, which is obtained by the system timer: `srand(time(0));` (`time(0)` returns the number of seconds since January 1, 1970).

Exercises:

1. Write a function to output all elements of an array.
2. Write a complete program to test `insert` function.
3. Write and test `remove` function to exclude an element so that no empty slot has to remain.
4. Explain the selection sort algorithm. Write a complete program to test it. Supply random data.
5. Write a program that inputs a sequence of numbers and outputs them in a reverse order.
6. Write a program that inputs a sequence of numbers and outputs all the positive elements of the input, followed by all the negative elements.
7. Write a program that inputs a sequence of numbers and outputs the sum of all exact square numbers in the input, if such numbers exist, otherwise outputs zero.
8. Implement the "bubble" sort algorithm: taking many times in a systematic fashion all pairs of two neighboring elements and swap the elements in the pair when they are not in the desired order. Stop the process at the moment, when there does not exist more pairs containing two neighboring elements that are not in order.
9. "Inter-galaxy" hotel had infinite many rooms, numbered by 1, 2, 3, ... All doors were closed. At half minute to 12 o'clock I opened all doors. At 1/4$^{\text{th}}$ minute to 12 o'clock I closed each second door. At 1/8$^{\text{th}}$ minute to 12 I visited each third door and opened it, if it was closed, or I closed it, if it was opened, and so on: At each 1/2$^{k\text{-th}}$ minute to 12 I visit each k^{th} door and opened it, if it was closed, or I closed it, if it was opened... Write a program that inputs an integer n ($1 < n < 10^6$) and outputs how many doors among those numbered between 1 and n were opened at 12 o'clock (attributed to Stanislav Lem, "Ijon Tichy stories from Star Diaries").

Arrays of strings and STL sorting: all together
Explain the purpose of the program. Explain all the elements of the source.

```
#include<iostream>
using namespace std;

string s[99];

int main()
{
 int n=0; string w;
 while(cin>>w) {s[n]=w; n++;}
 sort(s,s+n);
 for(int i=0;i<n;i++) cout << s[i] << endl;
 system("pause");
}
```

Write your own function to control the standard `sort()` algorithm: In the next example, the function `cmp()` is defined to do the opposite of the < operator. When `sort()` is called with `cmp()` used as the comparison function, the result is sorted in descending, rather than ascending, order. Write a complete program including the following commands:

```
int a[99];
bool cmp(int x, int y) {return x > y;}
sort(a,a+99,cmp);
```

Case study: The famous *Collatz number sequences* ("3n + 1 problem"). Consider the following *algorithm:*

```
1. input $n$
2. print $n$
3. if $n = 1$ then STOP
4. if $n$ is odd then $n := 3n+1$
5. else $n := n/2$
6. GOTO 2
```

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate for any integer input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers *n* such that $0 < n < 1{,}000{,}000$ (and, in fact, for many more numbers than this.)

Given an input *n*, it is possible to determine the count of numbers printed (including the 1). For a given *n* this is called the *cycle-length* of *n*. In the example above, the cycle-length of 22 is 16.

Write a program, that for any two given numbers *i* and *j* determines the maximum cycle-length over all numbers between *i* and *j*. The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0. You should process all pairs of integers and for each pair determine the maximum-cycle length over all integers between and including *i* and *j*. You can assume that no operation overflows a 32-bit integer.

For each pair of input integers *i* and *j* you should output *i*, *j*, and the maximum cycle-length for integers between and including *i* and *j*. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers *i* and *j* must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input	Sample Output

`1 10`
`100 200`
`201 210`
`900 1000`
	`1 10 20`
	`100 200 125`
	`201 210 89`
	`900 1000 174`

Exercises:

1. Write a program that inputs several lines of text and outputs them lexicographically sorted in increasing order.
2. Write a program that inputs a sequence of numbers and outputs them in decreasing order.
3. Write a program, that for given integers *a*, *b* and *c*, solves the equation $ax+by=c$ for integer values of *x* and *y*.
4. In the input, there are given integers, each of them coming twice, but only one exception. Find this exception. Example: 3 5 6 6 5 4 3 7 7, ouput: 4.
5. Find words in a grid of letters. Write a program that inputs a word, then inputs a rectangular grid of 10 by 10 letters and outputs "yes" or "no" depending on whether the given word exists, or not, in the given grid in horizontal or in vertical position. As an example, the word "peace" exists in the following grid:

```
abcdefghij
abcdefghij
abcdefghij
abcdefphij
abcdefehij
abcdefahij
abcdefchij
abcdefehij
abcdefghij
abcdefghij
```

ALGORITHMS IN THE ELEMENTARY THEORY OF NUMBERS

Print out all positive divisors of a given integer `a`

```
for(int i=1;i<=a; i++)
if(a%i==0) cout << i << " ";
```

Print out all prime divisors of a given integer `a`

```
int p=2;
while(p<=a)
{while(a%p==0){cout << p << " "; a /= p;}
 p++;
}
```

The Sieve of Eratosthenes. An algorithm for making tables of primes: Sequentially write down the integers from 2 to the highest number you wish to include in the table. Mark out all numbers which are divisible by 2 (every second number). Find the smallest remaining number. It is 3. Then mark out all numbers which are divisible by 3 (every third number). Find the smallest remaining number. It is 5. Then mark out all numbers which are divisible by 5 (every fifth number), and so on... The numbers remaining are prime numbers.

Euclid algorithm to find the *Greatest common divisor* of `a` and `b`

```
int gcd(int a, int b)
{
 while(a != b)
  if(a>b) a=a-b; else b=b-a;
 return a;
}
```

Extended Euclid algorithm. Problem: find integers x and y, such that $ax + by = c$.

Let us consider the case, when c = gcd(a, b). An example for a = 12, b = 16; gcd(a, b)=4.

12	16	A	b	$(1)a+(0)b$	$(0)a+(1)b$
12	4	A	$b–a$	$(1)a+(0)b$	$(–1)a+(1)b$
8	4	$a–(b–a)$	$b–a$	$(2)a+(–1)b$	$(–1)a+(1)b$
4	4	$a–(b–a)–(b–a)$	$b–a$	$(3)a+(–2)b$	$(–1)a+(1)b$
				`x1*a+y1*b`	`x2*a+y2*b`

i.e. $3a–2b=4$, or $–a+b=4$.

```
int x1=1, y1=0, x2=0, y2=1;
int gcd(int a, int b)
{
 while(a != b)
  if(a>b){a=a-b;x1=x1-x2;y1=y1-y2;}
  else{b=b-a;x2=x2-x1;y2=y2-y1;}
 return a;
}
```

Exercises:

1. Write a program that inputs a positive integer and determines whether it is a *prime*, or not.
2. Write a program that inputs a positive integer and outputs all divisors of it; each divisor should be printed once, followed by its multiplicity. An example: input 3528, output 2 3 3 2 7 2 ($3528=2^3 3^2 7^2$).
3. Write a function `bool rel_prime(int a, int b)` that determine whether `a` and `b` are *relatively prime*, or not.
4. Write a function `int Lcm(int a, int b)` that returns the *Least common multiple* of `a` and `b`.
5. A positive integer is called *perfect*, if it is equal to the sum of its proper divisors (the divisors excluding the given integer itself). An example 6 = 1 + 2 + 3. Write a program to find several perfect numbers.
6. A pair of distinct positive integers is called an *amicable pair*, if each member of the pair is equal to the sum of the proper divisors of the other member. E.g. (220, 284) is an amicable pair. Write a program to find more amicable pairs.
7. Implement the **Sieve of Eratosthenes** algorithm.
8. *Twin primes* are pairs of primes of the forms (p, p+2). Write a program that inputs a positive integer N and outputs the total number of twin primes in the interval 1, ... , N.
9. The *Goldbach conjecture* asserts that every positive even integer n (n > 2) can be expressed as a sum of two primes. Write a program to check this conjecture for possibly larger values of n (or, to find a counterexample – "who knows?"). There was a $1000000 prize for anyone who proved the Goldbach conjecture (`http://mathworld.wolfram.com/GoldbachConjecture.html`)

Horner scheme. Compute a value of a polynomial. Converting between number systems.

Given a polynomial $p(x)=a_0+a_1x+a_2x^2+a_3x^3+...+a_nx^n$, we can rewrite it in a form:

$a_0 + x(a_1 + x(a_2 + ...x(a_{n-2} + x(a_{n-1} + xa_n))...))$

which is easy to program:

```
int p=a[n];
for(int i=n-1;i>=0;i--) p = a[i]+x*p;
```

Input an integer as a string

```
int a[99];int n;
string s;
cin >> s;
n=s.length()-1;
for(int i=0;i<=n;i++) a[n-i]=(int)s[i]-(int)'0';
for(int i=n;i>=0;i--) cout << a[i]; cout << endl;
```

Long numbers addition

```
int a[99];int na;
int b[99];int nb;
int c[99];int n;
// given values: a,na,b,nb
if(na<nb) //alignment
 {n=nb; for(int i=na;i<=nb;i++) a[i]=0;}
else
 {n=na; for(int i=nb;i<=na;i++) b[i]=0;}
int t=0; // carry
for(int i=0;i<=n;i++)
 {c[i]=a[i]+b[i]+t;
  t=0;
  if(c[i]>10){t=1;c[i]=c[i]-10;}
 }
```

Exercises:

1. Write a program to find an integer solution (*x*, *y*) of *ax* + *by* = *c*.
2. Write a program to find all integer solutions (*x*, *y*) of *ax* + *by* = *c*, which belong to given intervals for *x* and *y*.
3. Write a program to find the result of division of a polynomial by binomial.
4. Write a complete program to input two long numbers, find their sum, and output the result.
5. Implement long numbers subtraction.
6. Write a program to multiply a long number by a 1-digit number
7. Consider the "Russian Peasant Method of Multiplication" (double and halve method). In the example: Multiply 39 by 52:

```
           39 x 52
           78 x 26
          156 x 13      (double and halve)
          312 x 6
          624 x 3
         1248 x 1

     156 + 624 + 1248 = 2028     (add only left with odd right)
```

Implement this method for long numbers.