ចំណុចសំខាន់ទាំង ៦ របស់ Algorithms

១- ជាមេតូត is used in computer science to describe a problem solving for implementation as a program

២- Algorithm ភាគច្រើនត្រូវបានប្រើជាចាំបាច់ក្នុងការរៀបចំឱ្យមានដំណើរការ data in the computation ។ ទាំងនេះ ជាដំណើរនៃការបង្កូរឱ្យមានជា Objects ដែលហៅថា Data Structure ហើយវាក៏ជា central objects of study in computer science

៣- Computer ជាជំនួយមួយដ៏ប្រសើរក្នុងការដោះស្រាយបញ្ហា, ករណីមានបញ្ហាតូចក្តី ធំក្តី តំរូវឱ្យមានការឈានឱ្យដល់ នៅការ solves the problem correctly, ដូច្នេះ ក៏តំរូវធ្វើយ៉ាងណាឱ្យមានលក្ខណៈ រហ័ស រស់រវើក ដោយបែង ចែកមេតូត use time or space as efficiently as possible

៤- ក្នុងការធ្វើដំណោះស្រាយបញ្ហាធំៗ ត្រូវមានការប្រុងប្រយ័ត្ន លើប្រសិទ្ធភាពនៃ process of solving => to be developed computer program ដែលត្រូវតែយល់ដឹងឱ្យច្បាស់ នឹងកំណត់អត្ថន័យ problem to be solved => managing its complexity និងបំបែកធាតុជាបំណែកតូចៗ ដែលអាចឱ្យមានការងាយស្រួលក្នុងកាអនុវត្តន៏

៥- Computer systems គឺជាអ្នករួមចំណែកមួយក្នុង program => មានការផ្លាស់ប្តូរវិធីសាស្រ្ត computing environments (HW & SW) ដើម្បីសំរេចបានជោគជ័យល្អនៃបញ្ហា (make our solutions more portable and longer lasting)

៦- ការជ្រើសរើស best algorithm សំរាប់ផ្នែកណាមួយនៃការងារ អាចផ្តល់ទង្វើបានជា complicated process, ឬក៏ជា sophisticated mathematical analysis. => លក្ខណៈការសិក្សាបែបនេះ ហៅថា analysis of algorithms (ការសិក្សាវិភាគលើ performance ព្រមនឹង experiences, to comparative performance of the methods)

Algorithm មាន running times សមាមាត្រទៅតាមអនុគមន៍

Algorithms ភាគច្រើនមាន Parameter សំខាន់មួយគឺ（N）ដែលមានឥទ្ធិពលទៅលើ Running time; នឹងដែលមានលក្ខណៈ N degree of polynomial（មានដូចជា ទំហំ files, ចំនួន characters ក្នុង text string）។ ទាំងនេះនាំឱ្យមានជាដំណើរ ដោយពឹងលើ mathematical formulas នឹងបន្ទាប់ផ្តល់បានជា Algorithms មាន running times សមាមាត្រទៅនឹងអនុគមន៍:

1 ជាដំណើរ Instruction ដែលបាន Executes once or a few times => running time is constant.

logN ជាដំណើរនៃ Program slower as N grows -> program នេះត្រូវដោះស្រាយបញ្ហាធំ, N=1000=> logN=3, N អាច double បាន ប៉ុន្តែ logN អាច double ទាល់តែ N កើនឡើងដល់ N^2

N Running time របស់ Program នេះជា Linear, លក្ខណៈការងារប្រភេទនេះ Algorithm វាច្រើនតែ process N inputs ឬ produce N outputs

NlogN ជា Algorithm ដោះស្រាយបញ្ហាដោយបំបែកជាផ្នែកៗនៃ sub problems（ដំណោះស្រាយវា Independently）រួចបន្ទាប់ទើបបញ្ចូលគ្នាវិញ។ ពេល N double => running time មានតំលៃច្រើនជាង N double

N^2 ជា Running time នៃ Algorithm ជា quadratic, អាចកើតមានឡើងនៅពេលវា process all pairs of data items（ករណីវា double -> nested loop）បើ N double => running time កើនដល់ជាបួនដង

2^N ជា algorithm មាន running time ជា exponential, ប្រើក្នុងករណីបង្ខំឱ្យមានដំណោះស្រាយអ្វីមួយ ។ បើ N double => running time is squares

ជារួម running time នៃផ្នែកណាមួយរបស់ program មានតំលៃជា **some constant * by one of these terms + some smaller terms** (values of the constant coefficient and the terms រាប់បញ្ចូលលើ results of the analysis on implementation details).

Collection of simple algorithms in C++

Recursive Algorithm of factorial

```
long fact (int n)
{ long y;
  if (n==0)
    return 1;
  else
      { y=fact(n-1);
        return n*y;
       }
}
           or  return n*fact(n-1);
```

Algorithm Iteration of factorial

```
long fact (int n)
{ long f;
  int i;
  f=1;
  for (i=1; i<=n; i++)
      f=f*i;
      return f;
}
```

Algorithm of Euclid's (GCD)

```
int gcd (int u; int v)
{ int t;
  while (u>0)
  { if (u<v)
       { t=u;
         u=v;
         v=t;
       }
       u=u-v;
   }
   return v;
}
```

Algorithm Tower of Hanoi

```
typedef char column;
int n;
void change(column x, column y)
{ cout <<”Change Disk “ <<x <<” to ”<<y <<”\n”; }

void change_column(int n, column a, column b, column c)
{ if (n>0)
  { change_column(n-1, a, c, b);
    change(a, c);
    change_column(n-1, b, a, c);
   }
}
```

Algorithm of Fibonacci

```
fib(n)=  if (n=0) then 0
         if (n=1) then 1
         else fib (n-1) + fib(n-2)
```

$f_0=0; f_1=1; f_n=f_{n-2}+f_{n-1}$

n	0	1	2	3	4	5	6	7	8	9
f_0	0	1	1	2	3	5	8	13	21	33

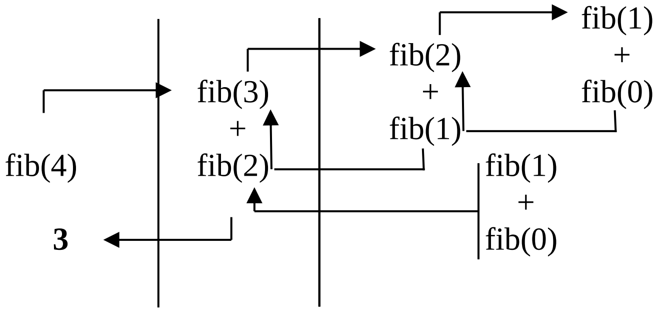
Recursive Algorithm of Fibonacci

```
long fib(int n)
{ long x,y;
  if (n<=1)
    return n;
  else
     { x=fib(n-1);
       y=fib(n-2);
       return x+y;
     }
}
        or  return fib(n-1) + fib(n-2);
```

Algorithm Iteration of Fibonacci

```
long fib(int n)
{ long fib2=0;
  long fib1=1;
  long fibn; int i;
  if (n<=1)
    fibn=n;
  else
      { for (i=2; i<=n; i++)
          { fibn=fib1+fib2;
            fib2=fib1;
            fib1=fibn;
          }
        }
   return fibn;
}
```

Example solution of Fibonacci showed as graphic

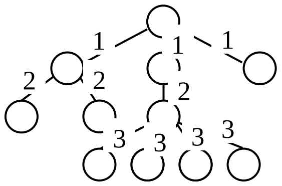


Algorithm of Prime number

```
const int N=100;
void main()
{ int i, j, a[N+1];
  for (a[1]=0; i=2; i<=N; i++)
        a[i]=1;
     for (i=2; i<=N/2; i++)
          for (j=2; j<=N/i; j++)
               a[i*j]=0;
               for (i=1; i<=N; i++)
   if (a[i])
      cout<<i <<’ ‘ ;
      cout<<’\n’;
}
```

How to calculate path length in the tree

PL=∑level nodes = ∑lengths
= (1+1+1)+(2+2+2)+(3+3+3+3)=21
= (1+2+2)+(1+2+(3+3+3+3)+1=21



Big-Oh Notation

The mathematical artifact that allows us to suppress detail when we are analyzing algorithms is called the *O-notation, or "big-Oh notation,"* which is defined as follows.

Definition 3.1 *A function g(N) is said to be O(f (N)) if there exist constants c_o and N_o such that g(N) < c_o f (N) for all N > N_o.*

We use the O-notation for three distinct purposes:

- To bound the error that we make when we ignore small terms in mathematical formulas
- To bound the error that we make when we ignore parts of a program that contribute a small amount to the total being analyzed
- To allow us to classify algorithms according to upper bounds on their total running times

Sorting Methods

```
BubbleSort(size)
{   for (i=size-1; i > 0; i--)
    {      for (j=0; j < i; j++)
        {         if (a[j] > a[j+1])
                  swap a[j] and a[j+1];
        }
    }
} Array size:   10
Last time:      43.182 sec
```

```
InsertionSort(size)
{   for (i = 1; i < size; i++)
    {      if (a[i] < a[i-1])
        {         temp = a[i];
                  for (j = i; j>0 && a[j-1]>temp; j--)
                        a[j] = a[j-1];   a[j]=temp;
        }
    }
} Array size:   10
Last time:      33.365 sec
```

```
BubbleSort(size)
{   for (i=size-1; i > 0; i--)
    {      swaps = false;
           for (j=0; j < i; j++)
        {         if (a[j] > a[j+1])
            {         swap a[j] and a[j+1];
                      swaps = true;
            }
        }
        if (!swaps) break;
    }
} Array size:   10
Last time:      51.258 sec
```

```
ShellSort(size)
{   for (inc = size/2; inc>0; inc /= 2)
    {      for (i = inc; i < size; i++)
        {         j = i - inc;
                  while (j >= 0)
            {      if (a[j] > a[j+inc])
                {     swap a[j] and a[j+inc];   j -= inc;
                }  else  {      j = -1;       }
            }
        }
    }
} Array size:   10
Last time:      40.66 sec
```

```
QuickSort(low, high)
{   if (high-low <= 1) return;
        pivot = a[high-1];
        split = low;
    for (i=low; i<high-1; i++)
          if (a[i] <pivot)
        {      swap a[i] and a[split];
               split++;
        }
          swap a[high-1] and a[split];
         QuickSort(low, split);
         QuickSort(split+1, high);
    return;
} Array size:   10
Last time:      48.819 sec
```

```
QuickSort(low, high)
{   if (high-low <= 1) return;
        pivot = MedianOf3(low, high);
        split = low;
    for (i=low; i<high-1; i++)
         if (a[i] <pivot)
        {     swap a[i] and a[split];    split++;     }
         swap a[high-1] and a[split];
        QuickSort(low, split);
        QuickSort(split+1, high);
      return;
}
MedianOf3(low, high)
{   middle = (low + high) / 2;
          if (a[low] > a[middle])
          swap a[low] and a[middle];
          if (a[low] > a[high-1])
          swap a[low] and a[high-1];
         if (a[middle] < a[high-1])
        swap a[middle] and a[high-1];
      return a[high - 1];
} Array size:   10
Last time:      38.075 sec
```

```
ShellSort(size)
{   for (inc = size/2; inc>0; inc /= 2)
    {      for (i = inc; i < size; i++)
        {         j = i - inc;
```

```
MergeSort(low, high)
{   if (high-low <= 1) return;
        split = (low+high) / 2;
        MergeSort(low, split);
```

```
              while (j >= 0)
        {           if (a[j] > a[j+inc])
          {              swap a[j] and a[j+inc];
                         j -= inc;
          } else  {   j = -1;     }
        }
      }
    }
} Array size:   10
Last time:       40.66 sec
```

```
        MergeSort(split, high);
       copy a[low ... split-1] to scratch array;
            m1 = 0;   m2 = split;   i = low;
       while (i < m2 && m2 < high)
         if (scratch[m1] <= a[m2])
            a[i++]=scratch[m1++];
         else    a[i++]=a[m2++];
       while (i < m2)   a[i++]=scratch[m1++];
} Array size:   10
Last time:       46.75 sec
```

```
HeapSort(size)
{   for (i = size/2; i >= 0; i--)
      ReHeap(size, i);
         for (i = size-1; i > 0; i--)
  {          swap a[i] and a[0];
             ReHeap(i, 0);
  }
}
ReHeap(len, parent)
{   temp = a[parent];
    child = 2*parent + 1;
    while (child < len)
  {      if (child<len-1 && a[child]<a[child+1])
           child++;
        if (temp >= a[child]) break;
          a[parent] = a[child];
          parent = child;
          child = 2*parent + 1;
  }
    a[parent] = temp;
   return;
} Array size:   10
Last time:       60.302 sec
```

```
MergeSort(low, high)
{   if (high-low <= 1) return;
        split = (low+high) / 2;
        MergeSort(low, split);
        MergeSort(split, high);
       copy a[low ... split-1] to scratch array;
            m1 = 0;
            m2 = split;
            i = low;
      while (i < m2 && m2 < high)
      if (scratch[m1] <= a[m2])
         a[i++]=scratch[m1++];
      else
          a[i++]=a[m2++];
      while (i < m2)
         a[i++]=scratch[m1++];
} Array size:   10
Last time:       46.75 sec
```

```
Radix(size)
{   allZero = false;
    for (i=0; !allZero; i++)
   {  allZero = true;
       for (j=m=n=0; j<size; j++)
     {   tmp = a[j] >> i;
               if (tmp > 1) allZero = false;
               if ((tmp & 1) == 1)
               bucket[m++] = a[j];
        else  a[n++] = a[j];
     }
        m = 0;
       while (n<size)
        a[n++] = bucket[m++];
   }
} Array size:   10
Last time:       70.517 sec
```

```
void SelectionSort(apvector <int> &num)
{    int i, j, first, temp;
      int numLength = num.length( );
      for (i= numLength - 1; i > 0; i--)
     {    first = 0;
          // initialize to subscript of first element
           for (j=1; j<=i; j++)
         // locate smallest between positions 1 and i.
          {     if (num[j] < num[first])
                 first = j;
          }    temp = num[first];
 // Swap smallest found with element in position i.
                num[first] = num[i];
                num[i] = temp;
     }  return;
}
```