МАТЕМАТИЧЕСКИ ОСНОВИ В АНАЛИЗ НА АЛГОРИТМИ

- 1. експоненти: (X ^A) *(X^B) = X^(A+B) ; (X^A)/(X^B)= X^(A-B)
  X^N + X^N = 2(X^N) != X^(2N) 2^N +2^N = 2^(N+1)

- 2. логаритми: X^A = B ако и само ако $\log_X B = A$ (ln; lg)

- T1: $\log_a B = \log_c B / \log_c A$ при A,B,C >0, A!= 1
  док: нека X=$\log_c B$; Y= $\log_c A$; Z= $\log_a B$; тогава C^X =B, C^Y = A, A^Z = B

  комбинираме от горните зависимости:

  B = C^X = A^Z =(C^Y)^Z и можем да запишем за степените:

  X = Y*Z

  Откъдето, очевидно, Z = X / Y което и искахме да докажем

- T2: log AB = logA +logB ; за A,B >0
  док: нека X = logA, Y= logB, Z = logAB;

  при основа 2, можем да запишем:

  2 ^X =A, 2^Y = B, 2^Z = AB

  комбинирайки горните уравнения:

  2 ^X* 2^Y = AB = 2^Z откъдето достигаме до: X + Y = Z

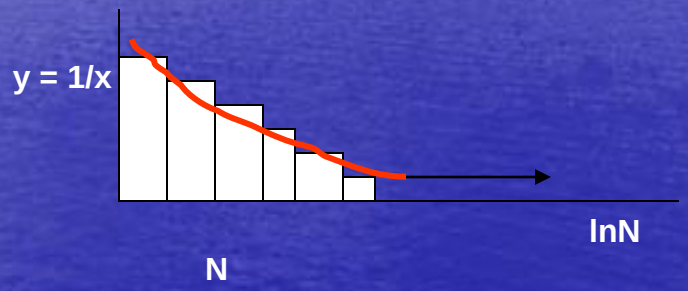
3. главен параметър на нарастване на ф-ии (интерпретация):

1; logN; N^3; NlogN; N^2 ; N ; 2^N ;

пр: оп./сек		размер 1000 000			размер 1 000 000 000		
	N	NlgN	N ^2	N	NlogN		N^2
10^6	сек	сек	седмици	час	час		никога
10^9	миг	миг	сек	миг	миг		седмици

4. математически обозначения при алгор. анализ:

$\lfloor X \rfloor$ lg N $\lceil X \rceil$ N!; хармонични числа: H = 1 + ½ +1/3 +….1/N



5. редици: $\sum_{i=0}^{N} 2^i = 2^{(N+1)} - 1$; $\sum_{i=0}^{N} A^i = (A^{(N+1)} - 1) / (A - 1)$.При 0 < A <1 то

Горната сума се апроксимира с

$\sum_{i=0}^{N} A^i <= 1 / (1 - A)$; при N→ безкр., сумата клони към 1/ (1-A)

док: нека S = 1 + A + A^2 + .. ; тогава AS = A + A^2 + A^3 +…..;

то S – AS = 1 и следоват: S = 1 / (1-A)

аритметични редици $\sum_{i=1}^{N} I = (N (N+1)) / 2$ което е приблизително: $N^2 / 2$

наистина: 2 + 5 +8+ … (3k -1) което е 3(1 + 2 +… +k) – (1 + 1+ ….1),

което е представимо и като (k (3k +1)) /2

следователно: ((3k + 1)k) /2 = 3 (X) - k → X = (k (k + 1)) / 2 или приблизителмо: $k^2 / 2$

За редица:

$\sum_{i=1}^{N} I^2 = (N (N + 1)(2N +1)) / 6$ приблизително $N^3 / 3$. Доказателсто: индукция; противоречие

док.: вярно за N = 1 → вярно за 1 <= k < = N. Да проверим за N + 1:

$\sum_{i=1}^{N+1} I^2 = \sum_{i=1}^{N} I^2 + (N + 1)^2 = (N(N + 1) (2N + 1)) / 6 + (N+1)^2 = (N + 1) * [(N (2N 1) / 6) + (N + 1)] =$

$= (N + 1) * [(2 N^2 + 7N + 6) / 6] = [(N +1) (N + 2) (2N + 3)] / 6$

3+4

6. рекурсии

пр: int f (int x)

{ if (x == 0) return 0;

else return (2 * f (x – 1) + x * x); }

т.е f(0) = 0; f(x) = 2f (x – 1) + x^2 ;

base case ; недостатъци ; грешки: f(-1) ; f(- 2) ….

пр: int bad (int n)

{ if (n == 0) return 0;

else return bad (n / 3 + 1) + n + 1; }

/ / bad (1) ???

правила при рекурсия:

base case

прогрес при всяка стъпка (пр. речник)

рекурсивните алгоритми работят:

пр: void printout (int n)

{ if (n > = 10) printOut (n / 10);

printDigit(n % 10) ; }

Т: този рекурсивен алгоритъм работи за n > = 0

док: чрез индукция → вярно за "к" цифри; следователно работи за " к + 1 "....

за предпочитане е реализация с цикъл for (ако е възможно) вместо рекурсия

никога в 1 стъпка повече от 1 рекурсия : → множествена рекурсия

пр: числа на Fibonaci: f (N) = f (N – 1) f (N- 2)

- **пр: void DoubleCountDown (int N)**
  { if (N < = 0) ;
  { DoubleCountDown (N – 1); DoubleCountDown (N – 1); }

/* времето се удвоява ???

нека имаме означение T (N) и T (0) = 1 ; времето на изпълнение се подчинява на формулата:

1 + 2 T (N – 1)

N	0	1	2	3	4	10
T (N)	1	3	7	15	31	2047

виждаме T (N) = 2^(N+!) - 1 или O (2 ^N)

***/**

индиректна рекурсия: A  B

пр: криви на Шерпински O (4 ^N)

разход на памет при рекурсия:

```
int BadForMemory ( )
{ int * x ;
  GetMemory ( x, 10 000 ); BadForMemory ( ) ; }
```

- **Типове рекурсии и оценки**

А . премахва 1 ел. от N входящи числа при циклично прохождане

$C_n = C_{n-1} + N ; C_1 = 1$

док: $C_n = C_{(n-1)} + N = C_{(n-2)} + (N - 1) + N = C_1 .. + 2 + ... N = ...$

$= 1 + 2 + ... + (N - 2) + (N - 1) + N = [N (N + 1)] / 2.$

- **Cn прибл.= $N^2 / 2$ /* вероятно T(раздробяване) + T (рекурентни изчисления)**

Б. рекурсии с разполовяване на входния поток на всяка стъпка

$C_n = 2 C_{(n/2)} + 1 ; C_1 = 1$

нека $N = 2^n$ (напр. битове за представяне на N)

$C(2^n) = C(2^{(n-1)}) + 1 = C_{(2^{(n-2)})} + 1 + 1 = C_{(2^{(n-3)})} + 3 = ... = C_{2^0} + n = n + 1$

- **→ оценката C_N зависи от броя битове , необходими за представяне на N → lg N**

В. рекурсивни програми, разполовяващи входа , но преглеждащи всеки елемент

$C_n = C_{(n/2)} + N; C1 = 0$

док: разписва се като $N + N/2 + N/4 + N/8 + ...$ → 2N

Г. при рекурсии с линейно прохождане на входната поредица, освен разполовяване

$C_n = 2C(n/2) + N;\quad C1 = 0$ **(алгоритми разделяй и владей)**

док: $C(2^n) = 2C(2^{(n-1)}) + 2^n$ **;делим на** 2^n **:** $C(2^n)/2^n = [C(2^{(n-1)})/2^{(n-1)}] + 1 =$

$= [2\,C\,2^{(n-2)} + 2^{(n-1)}] / 2^{(n-1)} + 1 = C\,2^{(n-2)}/2^{(n-2)} + 1 + 1 = \ldots = n$

→ $C_N = N \lg N$

n

Д. при рекурсивни програми, разполовяващи входа с изпълняване на константна работа

$C_n = 2C(n/2) + 1;\quad C_1 = 1.$

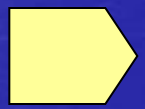
док: подобно на Д. C_N приблиз.= 2N.

- **Техники на формално преобразуване на рекурсивни в нерекурсивни алгоритми**

1.премахване на опашна рекурсия

```
procedure Recurse( A: Integer );
        begin

  // извършва нещо
  Recurse ( B)
  end;
```



```
procedure NonRecursive( A : Integer )

        begin

  while ( not done ) do
  begin
  // извършва нещо
  A : = B;
  end;          end;
```

2.запаметяване на междинни резултати
пр: Fibonacci числа: за Fib(29) → Fib(1) и Fib(0) се изчисляват 832 040 пъти ???
Междинните резултати се съхраняват в таблица след първо изчисляване.

3. подменяне посоката top- down към botton- up (пр. с Fibonacci числа)
при тази техника оценката за алгоритъма на Fibonacci е O(N) за Fib (N) вместо O(Fib (N))

пр: за Pentium 166MHz и Fib(40) ------ 155 sec с рекурсивен алгоритъм, докато с тази модификация
Fib(1476) се смята за < sec.

4. подмяна на алгоритъма чрез преструктуриране на кода:

оформяне на отделни процедури за съхраняване инф, стъпка и възстановяване :

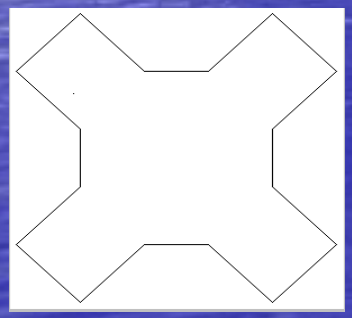
- **пр: procedure Recurse (num : Integer);**
  begin
  < block1 of code >
  Recurse(<parameters>)
  <block2 of code>
  end;
- **преструктурираме:**

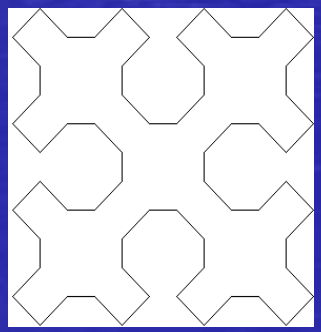
```
procedure Recurse(num: Integer);
                begin
1               <block1 of code>
                Recurse(<parameters>)
2               <block2 of code>
                end;
```

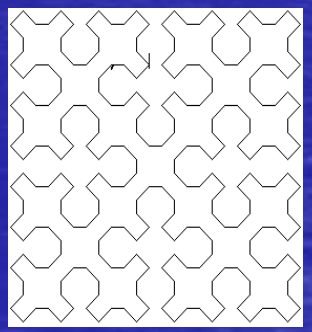
и:

```
procedure non_Recurse(num: Integer);
var                          pc: Integer;
begin
                pc := 1;
                repeat
                case pc of
1:              begin  <code block1>
                if(базов случай) then pc :=2          // прескачаме рекурсията
                else  begin
// съхраняваме пром.  за след рекурсивното викане и pc = 2; установяваме пром.
// нужни при рекурсия ( напр. n:= n-1)  и преход към начало на рекурсията:
                pc := 1;      end;                    end;
2:              begin
                <code of block2>
                pc := 0;      end;
0:  //  код след рекурсията
```

5. алгоритмична промяна при сложни случаи на взаимна рекурсия: пр. криви на Sierpinski



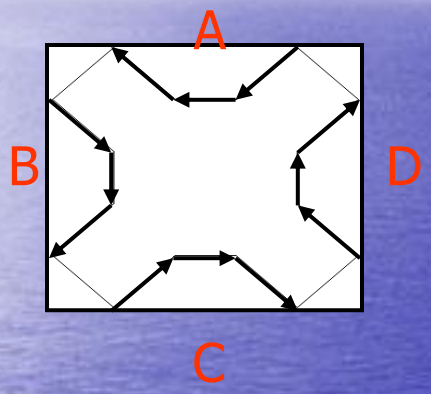


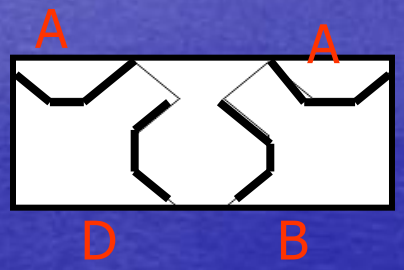


оформяме отделни процедури с взаимна рекурсия:

SierpA, SierpB, SierpC and SierpD

всяка от тях вика останалите, които от своя страна я викат рекурсивно. Всяка служи за изчертаване на горна, лява, долна и дясна част от общата картина





- A : A B D A
  B: B C A B
  C: C D B C
  D: D A C D

следва рекурсивен вариант на една от процедурите:

```
procedure TSierp1Form.SierpA(depth, dist : Integer);
begin
with DrawArea.Canvas do
   begin
      if (depth = 1) then
      begin
         LineTo(PenPos.X - dist, PenPos.Y + dist);
         LineTo(PenPos.X - dist, PenPos.Y + 0   );
         LineTo(PenPos.X - dist, PenPos.Y - dist);
      end else begin
         SierpA(depth - 1, dist);
         LineTo(PenPos.X - dist, PenPos.Y + dist);
         SierpB(depth - 1, dist);
         LineTo(PenPos.X - dist, PenPos.Y + 0   );
         SierpD(depth - 1, dist);
         LineTo(PenPos.X - dist, PenPos.Y - dist);
         SierpA(depth - 1, dist);
      end;       end;                        end;
```

оценката на алгоритмичната сложност е: $O(4^N)$

следва нерекурсивен вариант на преобразувания алгоритъм:

```
procedure DrawSubcurve(depth, dist, func : Integer);
        begin
                case func of
                        1:                      // SierpA
                        2:                      // SierpB
                        3:                      // SierpC
                        4:                      // SierpD
                end;    end;
```

Обобщената процедура е масивно рекурсивна: вика себе си 4 * 4 пъти.

- **Номерираме повикванията:**

```
// SierpA code fragment, according the algorithmic transformation
    with DrawArea.Canvas do
      begin
    11    if (depth = 1) then
        begin
          LineTo(PenPos.X - dist, PenPos.Y + dist);
          LineTo(PenPos.X - dist, PenPos.Y + 0   );
          LineTo(PenPos.X - dist, PenPos.Y - dist);
        end
          else begin
          SierpA(depth - 1, dist);
    12      LineTo(PenPos.X - dist, PenPos.Y + dist);
          SierpB(depth - 1, dist);
    13      LineTo(PenPos.X - dist, PenPos.Y + 0   );
          SierpD(depth - 1, dist);
    14      LineTo(PenPos.X - dist, PenPos.Y - dist);
          SierpA(depth - 1, dist);
        end;
      end;
```

подмяната на рекурсията става чрез преходи от вида (напр. от SierpA към SierpB):

```
PushValues( depth, 13)                    // resume at 13
depth := depth – 1;
pc := 21;                                 // transfer control to the start of SierpB code
```

- ето окончателния, номериран , нерекурсивен вариант на същата програма(само част за SierpA (за SierpB, SierpC, SierpD – кодът е аналогичен и следва да се добави):

```
procedure TSierpinskiForm.DrawSubcurve(depth, pc, dist : Integer);
begin
   with DrawArea.Canvas do
   begin
      while (true) do
      begin
         case pc of                                          // * SierpA *
         11:
            begin
               if (depth <= 1) then
               begin
                  LineTo(PenPos.X - dist, PenPos.Y + dist);
                  LineTo(PenPos.X - dist, PenPos.Y + 0   );
```

```
LineTo(PenPos.X - dist, PenPos.Y - dist);
                pc := 0;
              end else begin
                PushValues(12, depth); // Run SierpA
                depth := depth - 1;
                pc := 11;                          end;                    end;
        12:
          begin
            LineTo(PenPos.X - dist, PenPos.Y + dist);
            PushValues(13, depth); // Run SierpB
            depth := depth - 1;
            pc := 21;
          end;
        13:
          begin
            LineTo(PenPos.X - dist, PenPos.Y + 0);
            PushValues(14, depth); // Run SierpD
            depth := depth - 1;
            pc := 41;
          end;
        14:
          begin
            LineTo(PenPos.X - dist, PenPos.Y - dist);
            PushValues(0, depth); // Run SierpA
            depth := depth - 1;
            pc := 11;
          end;
```

$O(N^4)$

Оценката е много по-добра:

- **Допустима е много по-голяма дълбочина на вложеност. Разбираемостта на кода е по-лоша.**