АНАЛИЗ ТРУДОЕМКОСТИ РЕКУРСИВНЫХ АЛГОРИТМОВ МЕТОДОМ ПОДСЧЕТА ВЕРШИН ДЕРЕВА РЕКУРСИИ

РЕКУРСИВНАЯ ТРИАДА

Рекурсивную триаду составляют

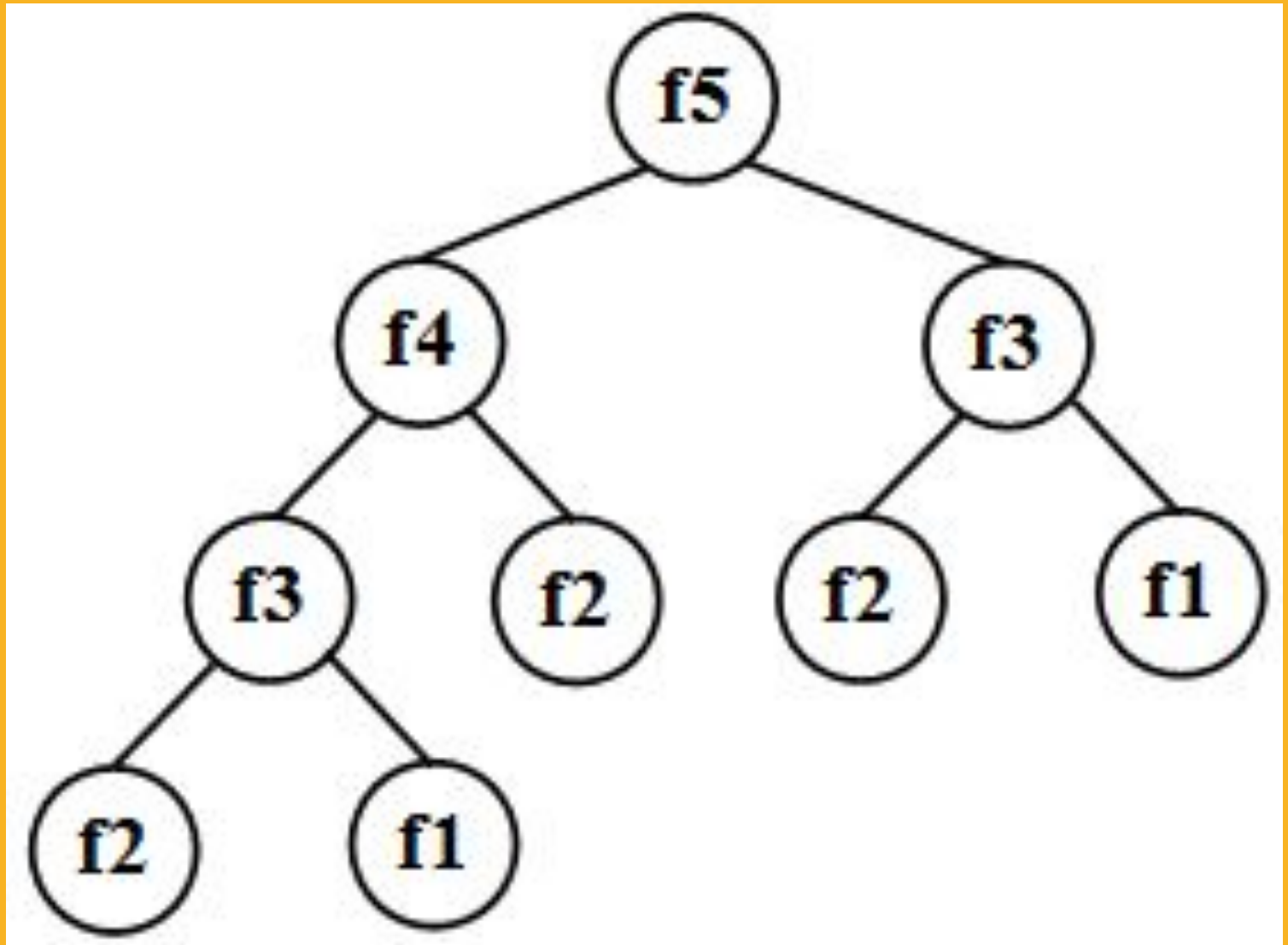
- параметризация
- выделение базы
- декомпозиция

обозначения для конкретного входного параметра D:

- R(D) – общее число вершин дерева рекурсии,
- R_V(D) – объем рекурсии без листьев (внутренние вершины),
- R_L(D) – количество листьев дерева рекурсии,
- H_R(D) – глубина рекурсии.

ПОСЛЕДОВАТЕЛЬНОСТЬ ФИБОНАЧЧИ

```
int Fib(int n){ //n – номер члена последовательности
  if(n<3) return 1; //база рекурсии
  return Fib(n-1)+Fib(n-2); //декомпозиция
}
```

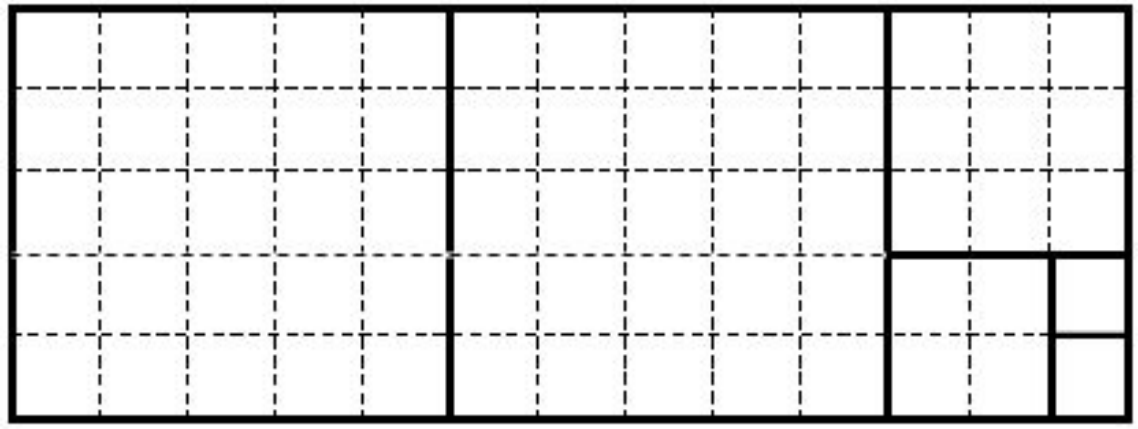


ПОЛНОЕ ДЕРЕВО РЕКУРСИИ ДЛЯ ПЯТОГО ЧЛЕНА ПОСЛЕДОВАТЕЛЬНОСТИ ФИБОНАЧЧИ

ХАРАКТЕРИСТИКАМИ РАССМАТРИВАЕМОГО МЕТОДА ОЦЕНКИ АЛГОРИТМА БУДУТ СЛЕДУЮЩИЕ ВЕЛИЧИНЫ.

D = 5	D = n
$R(D)=9$	$R(D)=2fn-1$
$R_V(D)=4$	$R_V(D)=fn-1$
$R_L(D)=5$	$R_L(D)=fn$
$H_R(D)=4$	$H_R(D)=n-1$

ЗАДАЧА О РАЗРЕЗАНИИ ПРЯМОУГОЛЬНИКА НА КВАДРАТЫ.



Разработаем рекурсивную триаду.

Параметризация: m, n – натуральные числа, соответствующие размерам прямоугольника.

База рекурсии: для m=n число получившихся квадратов равно 1, так как данный прямоугольник уже является квадратом.

Декомпозиция: если m!=n , то возможны два случая m < n или m > n.

Пример разрезания прямоугольника 13х5 на квадраты

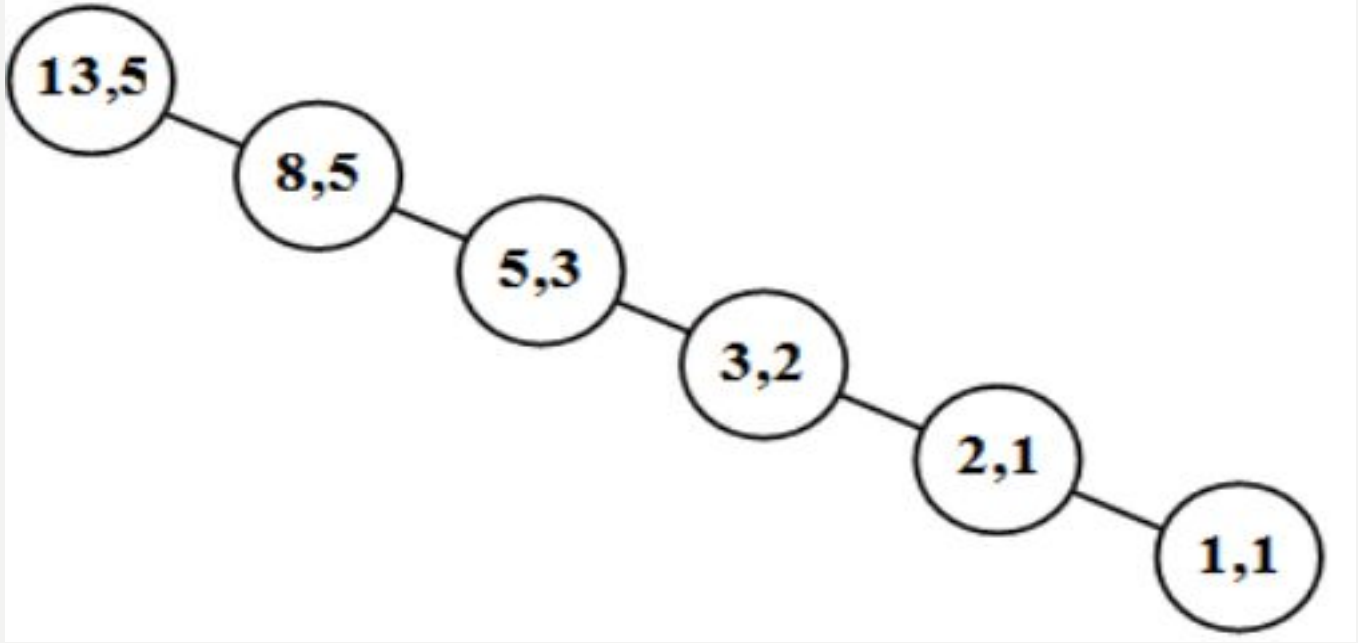
```
#INCLUDE "STDAFX.H"
#INCLUDE <IOSTREAM>
USING NAMESPACE STD;
INT KV(INT M,INT N);

INT _TMAIN(INT ARGC, _TCHAR* ARGV[]) {
  INT A,B,K;
  PRINTF("ВВЕДИТЕ СТОРОНЫ ПРЯМОУГОЛЬНИКА->");
  SCANF("%D%D",&A,&B);
  K = KV(A,B);
  PRINTF("ПРЯМОУГОЛЬНИК СО СТОРОНАМИ %D И %D МОЖНО РАЗРЕЗАТЬ
         НА %D КВАДРАТОВ",A,B,K);
  SYSTEM("PAUSE");
  RETURN 0;
}

INT KV(INT M,INT N){ //M,N – СТОРОНЫ ПРЯМОУГОЛЬНИКА
  IF(M==N) RETURN 1; //БАЗА РЕКУРСИИ
  IF(M>N) RETURN 1+KV(M-N,N); //ДЕКОМПОЗИЦИЯ ДЛЯ M>N
  RETURN 1+KV(M,N-M); //ДЕКОМПОЗИЦИЯ ДЛЯ M<N
}
```

ХАРАКТЕРИСТИКАМИ РАССМАТРИВАЕМОГО МЕТОДА ОЦЕНКИ АЛГОРИТМА БУДУТ СЛЕДУЮЩИЕ ВЕЛИЧИНЫ

D = (13, 5)	D = (m, n), m $\geq$ n, худший случай
$R(D)=6$	$R(D)=m$
$R_V(D)=4$	$R_V(D)=m-2$
$R_L(D)=1$	$R_L(D)=1$
$H_R(D)=6$	$H_R(D)=m$

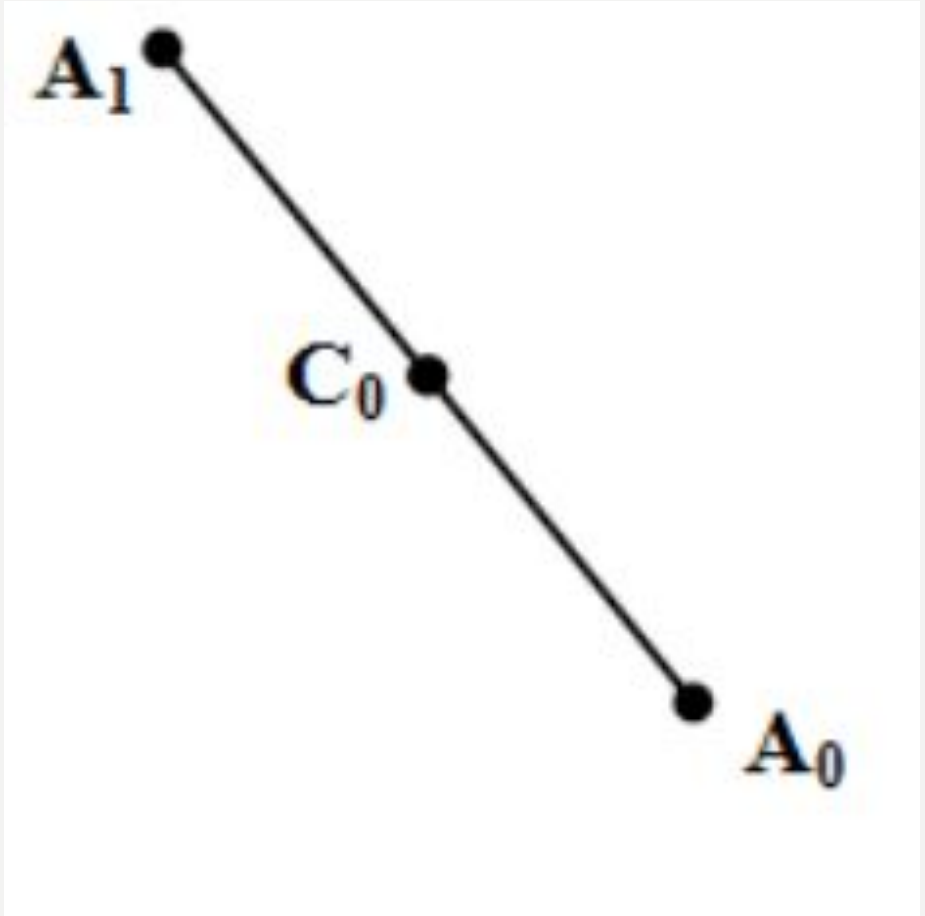


Пример полного дерева рекурсии для разрезания прямоугольника 13x5 на квадраты

ЗАДАЧА О НАХОЖДЕНИИ ЦЕНТРА ТЯЖЕСТИ ВЫПУКЛОГО МНОГОУГОЛЬНИКА.

Разработаем рекурсивную триаду.
Параметризация: x, y – вещественные массивы, в которых хранятся координаты вершин многоугольника; n – это число вершин многоугольника, по условию задачи, n>1 так как минимальное число вершин имеет двуугольник (отрезок).

База рекурсии: для n=2 в качестве многоугольника рассматривается отрезок, центром тяжести которого является его середина

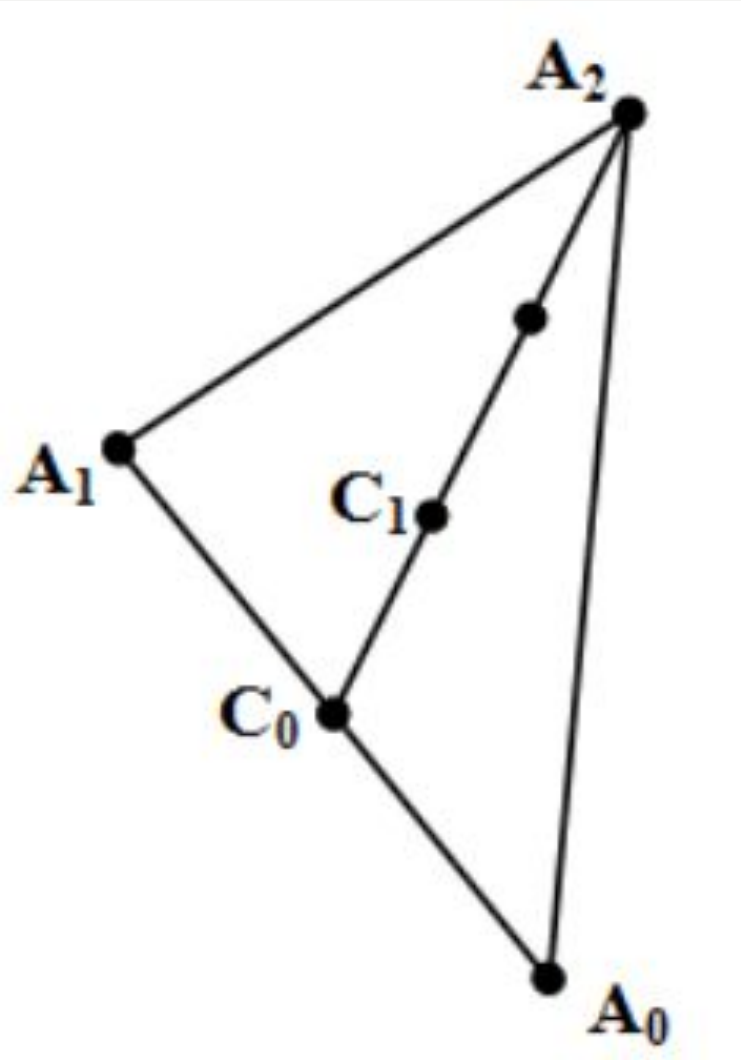


Если координаты концов отрезка заданы как (x_0,y_0) и (x_1,y_1), то координаты середины вычисляются по формуле:

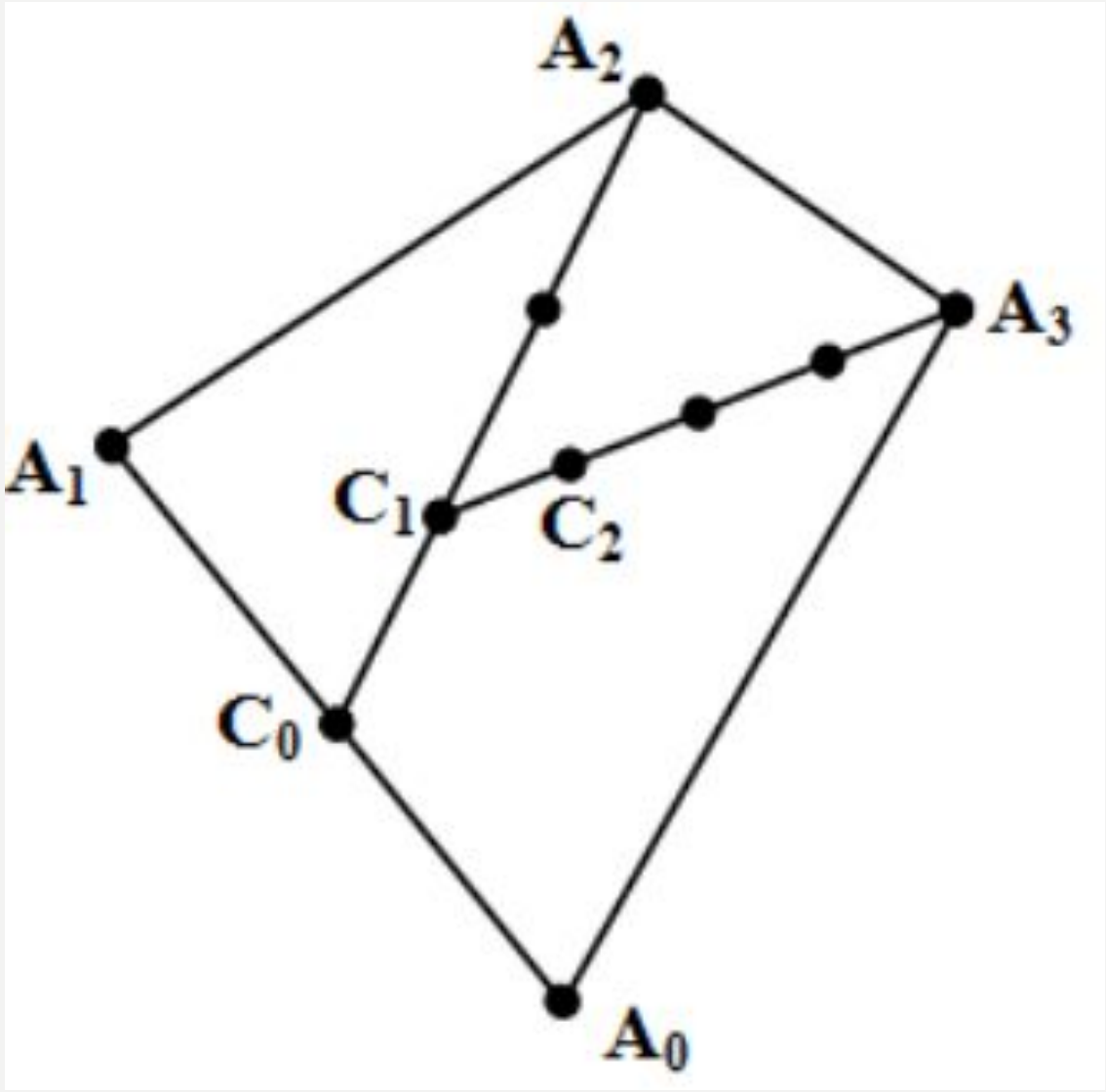
$$cx = \frac{x_0 + x_1}{2}, \quad cy = \frac{y_0 + y_1}{2}.$$

Декомпозиция: если n>2, то рассмотрим последовательное нахождение центров тяжести треугольника, четырехугольника и т.д.

Для n=3 центром тяжести треугольника является точка пересечения его медиан



Для n=4 центром тяжести четырехугольника является точка, делящая в отношении 3 : 1, считая от вершины, отрезок



Если концы отрезка заданы координатами вершины (x_n,y_n) и центра тяжести (n-1) -угольника (cx_{n-1},cy_{n-1}), то при делении отрезка в данном отношении получаем координаты:

$$cx_n = \frac{x_n + (n-1)cx_{n-1}}{n}, \quad cy_n = \frac{y_n + (n-1)cy_{n-1}}{n}$$

```
    fscanf(f, "%d",&m);
    printf("\n%d",m);
    if ( m < 2 || m > max ) //вырожденный многоугольник
      printf ("Вырожденный многоугольник");
    else {
     float *px,*py,*pc;
      px = new float[m];
      py = new float[m];
      pc = new float[2];
      pc[0] = pc[1] = 0;
      while(i<m) {
        fscanf(f, "%f %f",&px[i], &py[i]);
        printf("\n%f %f",px[i], py[i]);
        i++;
      }
      centr(m,px,py,pc);
      printf ("\nЦентр тяжести имеет координаты:
              (%.4f, %.4f)",pc[0],pc[1]);
      delete [] pc;
      delete [] py;
      delete [] px;
    }
    fclose(f);
  }
  system("pause");
  return 0;
}

void centr(int n,float *x, float *y, float *c){
```

```
FILE *f;
if ( ( f = fopen("in.txt", "r") ) == NULL )
  perror("in.txt");
else {
  fscanf(f, "%d",&m);
  printf("\n%d",m);
  if ( m < 2 || m > max ) //вырожденный многоугольник
    printf ("Вырожденный многоугольник");
  else {
   float *px,*py,*pc;
    px = new float[m];
    py = new float[m];
    pc = new float[2];
    pc[0] = pc[1] = 0;
    while(i<m) {
      fscanf(f, "%f %f",&px[i], &py[i]);
      printf("\n%f %f",px[i], py[i]);
      i++;
    }
    centr(m,px,py,pc);
    printf ("\nЦентр тяжести имеет координаты:
            (%.4f, %.4f)",pc[0],pc[1]);
    delete [] pc;
```

```
void centr(int n,float *x, float *y, float *c){
//n - количество вершин,
//x,y - координаты вершин,
//c - координаты центра тяжести
  if(n==2){ //база рекурсии
          c[0]=(x[0]+x[1])/2;
          c[1]=(y[0]+y[1])/2;
         }
  if(n>2) { //декомпозиция
          centr(n-1,x,y,c);
          c[0]= (x[n-1] + (n-1)*c[0])/n;
          c[1]= (y[n-1] + (n-1)*c[1])/n;
```

Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины.

D = 4	D = n
$R(D)=3$	$R(D)=n-1$
$R_V(D)=1$	$R_V(D)=n-3$
$R_L(D)=1$	$R_L(D)=1$
$H_R(D)=3$	$H_R(D)=n-1$

ЗАДАЧА О РАЗБИЕНИИ ЦЕЛОГО НА ЧАСТИ.

Например, разбиение числа 6 будет представлено 11 комбинациями:

6
5+1
4+2, 4+1+1
3+3, 3+2+1, 3+1+1+1
2+2+2, 2+2+1+1, 2+1+1+1+1
1+1+1+1+1+1

Пусть зависимость R(n,k) вычисляет количество разбиений числа n на сумму слагаемых, не превосходящих k

свойства R(n,k).

2.R(n,k)=1

3.Если второй параметр превосходит значение первого , то имеет место равенство R(n,k)=R(n,n)

4.Если в сумму входит слагаемое, равное первому параметру, то такое представление также единственно (содержит только это слагаемое), поэтому имеет место равенство: R(n,n)=R(n,n-1)+1.

5. (n>k), R(n,k)=R(n-k,k)+R(n,k-1).

РАЗРАБОТАЕМ РЕКУРСИВНУЮ ТРИАДУ.

- *Параметризация*: Рассмотрим разбиение натурального числа n на сумму таких слагаемых, которые не превосходят натурального числа k.
- *База рекурсии*: исходя из свойств рассмотренной зависимости, выделяются два базовых случая:
- при n=1 R(n,k)=1,
- при k=1 R(n,k)=1.

- *Декомпозиция*: общий случай задачи сводится к трем случаям, которые и составляют декомпозиционные отношения.
- при $n=k$ $R(n,k)=R(n,n-1)+1$,
- при $n<k$ $R(n,k)=R(n,n)$,
- при $n>k$ $R(n,k)=R(n-k,k)+R(n,k-1)$.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
unsigned long int Razbienie(unsigned long int n,
                            unsigned long int k);

int _tmain(int argc, _TCHAR* argv[]){
  unsigned long int number, max,num;
  printf ("\nВведите натуральное число: ");
  scanf ("%d", &number);
  printf ("Введите максимальное натуральное слагаемое в  сумме: ");
  scanf ("%d", &max);
  num=Razbienie(number,max);
   printf ("Число %d можно представить в виде суммы с максимальным слагаемым %d.",
number, max);
  printf ("\nКоличество разбиений равно %d",num);
```

ЗАДАЧА О ПЕРЕВОДЕ НАТУРАЛЬНОГО ЧИСЛА В ШЕСТНАДЦАТЕРИЧНУЮ СИСТЕМУ СЧИСЛЕНИЯ.

- $n_{10}=k_p$
- *Параметризация*: n – данное натуральное число, p – основание системы счисления.
- *База рекурсии*: на основании правил перевода чисел из десятичной системы в систему счисления с основанием p, деление нацело на основание системы выполняется до тех пор, пока неполное частное не станет равным нулю, то есть: если целая часть частного n и p равна нулю, то k = n. Данное условие можно реализовать иначе, сравнив n и p: целая часть частного равна нулю, если n < p.
- *Декомпозиция*: в общем случае k формируется из цифр целой части частного n и p, представленной в системе счисления с основанием p, и остатка от деления n на p.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#define maxline 50
void perevod( unsigned long n, unsigned int p,FILE *pf);

int _tmain(int argc, _TCHAR* argv[]){
  unsigned long number10;
  unsigned int osn=16;
  char number16[maxline];
  FILE *f;
  if ((f=fopen("out.txt", "w"))==NULL)
    perror("out.txt");
  else {
    printf ("\nВведите число в десятичной системе: ");
    scanf("%ld", &number10);
    perevod(number10, osn, f);
   fclose(f);
  }
  if ((f=fopen("out.txt", "r"))==NULL)
```