Рекурсията е метод за изпълняване на задача при който фрагмент от програмата се обръща сам към себе си. Това може да стане като програмния фрагмент извиква директно себе си, това е пряка рекурсия, вторият вариант е когато първият фрагмент извиква втори а вторият пряко или непряко извиква първия, това е непряка рекурсия. Рекурсията е метод който често довежда програмите до зацикляне. Това се получава когато един фрагмент извиква пряко или непряко себе си до безкрайност, като по този начин никога не се излиза от цикъла и крайното условие никога не се изпълнява. Друг проблем при рекурсията е, това че тя води до нечетливост на кода, защото ако рекурсията извиква повече от веднъж себе си на принципа на дървовидна структура се губи ориентация за текущото положение на програмата и на клона в който се намираме.

Пример за факториел:

Рекурсивен.

```
long int factorial(int n)
 {if (n<=1)
    return(1);
  else
    n=n*factorial(n-1);
    return(n); }
```

Нерекурсивен.

```
int Factoriel(int N)
{
Fact = 1;
for (i=1; i<=N; i++)
  {  Fact=Fact*i;  }
}
```

Пример за Фибоначи:

Рекурсивен.

```
long int fibonachi(int n)
{if (n<=2)
    return(1);
  else
    n=(fibonachi(n-1)+fibonachi(n-2));
    return(n);
}
```

Нерекурсивен.

```
int Fibonachi(int n)
{int fib1 = 0;
 int fib2 = 1;
 int fib;
 for ( int i = 2; i < n; i++ )
 {  fib = fib1 + fib2;
    fib1 = fib2;
    fib2 = fib; }
}
```

Бързите методи за сортиране са се базират на стратегията разделяй и владей. На принципа на тази стратегия се ускорява времето за работа и сложността на алгоритмите става N*log(N). Примери за методи за бързо сортиране са: quicksort, сортирането чрез сливане и пирамидалната сортировка. В това упражнение ще разгледаме първите два вида.

Сортиране чрез сливане:

Идеята е цялата последователност от елементи да се разбие на 2 равни части, които да се сортират поотделно и след това сортираните части да се слеят в една обща последователност от N елемента. Разбиването на части може да се извърши рекурсивно до получаване на 1 елемента, които се приема за сортиран сам по себе си. След което елементите на обратния принцип започват да се обединяват, като при обединението им се подреждат в под масивите. Това се повтаря докато се достигне до обединение на основния масив.

```
#include <iostream.h>
int a[N];
void merge(int low,int middle,int high)
{
 int h,i,j,b[50],k;
 h=low;
 i=low;
 j=middle+1;
 while((h<=middle)&&(j<=high))
 {  if(a[h]<=a[j])
     { b[i]=a[h];
       h++; }
    else
     { b[i]=a[j];
       j++; }
    i++;
 }
 if(h>middle)
 { for(k=j;k<=high;k++)
   { b[i]=a[k];
     i++; }
 }
 else
 {  for(k=h;k<=middle;k++)
    { b[i]=a[k];
      i++; }
 }
  for(k=low;k<=high;k++) a[k]=b[k];
}
```

```
void merge_sort(int low,int high)
{
 int middle;
 if(low<high)
 {
  middle=(low+high)/2;
  merge_sort(low,middle);
  merge_sort(middle+1,high);
  merge(low,middle,high);
}}
```

```
void main()
{ int num,i;
cout<<"Въведете брой елементи: "<<endl;
cin>>num;
cout<<"Въведете елементите: "<<endl;
for(i=1;i<=num;i++)
{
cin>>a[i] ;
}
merge_sort(1,num);
for(i=1;i<=num;i++)
cout<<a[i]<<", ";
}
```

Метод чрез сливането. Кодът е показан по-горе, а също така можете да го гледате от картинката с цветните фонове. Ще разгледаме цветните блокове.
В първият жълт блок е показан логическия начин на действие на алгоритъма: масивът който имаме се разделя на 2 под масива без елементите в него да следват някаква зависимост на разпределение. Като при разделянето си за всеки от двата под масива се извиква рекурсивно (отново) същата функция за разделяне. Това се повтаря докато във всеки под масив остане по 1 елемент. След което започва обединение на под масивите от по един елемент, при което се сравняват левият и десният и се подреждат един спрямо друг. След което масивите от по 2 и повече елемента като при обединението им пак се сравняват елементите и се подреждат сами по себе си. Сравнението става в първия и втория зелен блок. Ако елемент от левия съответно десния под масив е по по малък той се записва в обединяващия масив на двата под масива и пойнтерът на под масива от който е записан елемент се премества с един индекс. Това се повтаря докато елементите на единият от под масивите свършат, след това елементите на другия се добавят автоматично към обединяващия ги масив. Автоматичното добавяне на елементи става в жълтия блок над кафявия. При всички тези обединения елементите се записват в допълнителен масив докато стане обединението след което те се пренасят в основния. Пренасянето в основния масив се извършва от кафявия участък от кода. Обединението на елементите в под масивите се прави за всяка стъпка за която е било извикано преди това разделяне защото обединението е част от тялото на рекурсивно извикваната функция. Както се забелязва тук елементите от масивите и под масивите се разделят без определени белези но при връщането им в масивите и под масивите те се сравняват и се връщат в определен ред.

Бързо сортиране (Quick Sort):
За разлика от сортирането чрез сливане, където масива първо се разделя на елементи и след това при обединяването му те се сортират, то при бързото сортиране елементите първо грубо се разпределят на две групи, по определен критерии и след това масива се разделя. За групиране на елементите се използва следния критерии. Елементите се разделят според стойността си на по-малки и на по-големи спрямо един елемент. Въпросният елемент се избира на произволен принцип от останалите елементи и се нарича Пивотен елемент! След като се избере пивотния елемент, останалите се разделят на две групи. Примерно група номер едно се състои от по-малки и равни на пивотния, а група номер две се състои от по-големи от пивотния. След което отново се използва рекурсия, като този метод се извиква по отделно за всяка от тези групи. Избират се нови пивотни елементи и останалите елементи се групират спрямо тях. Това действие се повтаря, докато във всяка група остане само един елемент. След като всички под групи са разделени, то тяхното връщане в масива гарантира, че той ще бъде подреден само с добавяне на елементите, без да е необходимо допълнително сравнение, както в сортирането чрез сливане. Да обърнем малко внимание на 3-те цветни блока. В зеления блок елементите които са по-големи от пивота и са отдясно остават там. Ако някой елемент е по-малък той се записва на в позицията към която е левия пойнтер и се излиза от зеления блок. В синия блок се сравняват елементи който са по-малки от него и са отляво. Ако се достигне до елемент който е по-голям той се записва на мястото където е пойнтера за дясната част. Тоест върху старото място на елемента който преди това от зеления блок се е записал отляво, след това се излиза от синия блок и мястото от ляво което е направил синия блок се запълва с стойността на пивота. Процедурата се извършва за останалия участък докато while (left<right) – сивият ред.

```
//#include "stdafx.h"
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int *numbers, NUM_ITEMS;
using namespace std;
void quickSort(int numbers[], int array_size);
void q_sort(int numbers[], int left, int right);

int main()
{
  int i;
  printf("\nNUM_ITEMS = ");
  cin>>NUM_ITEMS;
  numbers = new int[NUM_ITEMS];
  for (i = 0; i < NUM_ITEMS; i++)
  {
  printf("\n numbers[%d] = ", i);
  int p;
  cin>>p;
  numbers[i] = p;
  }
  quickSort(numbers, NUM_ITEMS);
  cout<<"\nDone with sorting!\n";
  for (i = 0; i < NUM_ITEMS; i++)
  printf("numbers[%d] = %d\n", i, numbers[i]);
  return 0;
}

void quickSort(int numbers[], int array_size)
{
 q_sort(numbers, 0, array_size - 1);
}
```

```
void q_sort(int numbers[], int left, int right)
{
 int pivot, l_hold, r_hold;
 l_hold = left;
 r_hold = right;
 pivot = numbers[left];
 while (left < right)
 {
   while ((numbers[right] >= pivot) && (left < right))
     right--;
   if (left != right)
   {
     numbers[left] = numbers[right];
     left++;
   }
   while ((numbers[left] <= pivot) && (left < right))
     left++;
   if (left != right)
   {
     numbers[right] = numbers[left];
     right--;
   }
 }
 numbers[left] = pivot;
 pivot = left;
 left = l_hold;
 right = r_hold;
 if (left < pivot)
   q_sort(numbers, left, pivot-1);
 if (right > pivot)
   q_sort(numbers, pivot+1, right)
}
```