**Сравняване на алгоритмите за сортиране**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10000 elements | | 100000 elements | |
|  | Memory | time | memory | time |
| Name | MB | sec |  |  |
| Bubble | 1.144 | 0.656 |  |  |
| insertion | 1.144 | 0.203 | 15.578 | 1.496 |
| selection | 1.144 | 0.313 | 21.188 | 1.496 |
| quicksort | 1.144 | 0.016 | 0.016 | 1.496 |
| heapsort | 1.144 | 0.031 | 0.109 | 1.496 |
| shell | 1.144 | 0.016 | 0.109 | 1.496 |
| merge | 1.176 | 0.047 | 0.094 | 1.988 |

**Bubble sort -** Масивът се обхожда отзад-напред (отдолу-нагоре), за да се намери минималния елемент и по този начин отгоре да изплува най-малкият.

1. За оптимизация може да използваме Flag от булев тип, което да показва дали някъде е правена размяна и ако не е – се прекратява обработката.
2. Може да се следи къде е правена последната размяна

3.Обхожда се в несортираната част в две посоки

Независимо от всички подобрения, сложността му си остава O(n2)

**Insertion sort -**Чрез сравняващо сортиране сортираният списък се допълва с по един елемент всеки път. [Алгоритъмът](http://bg.wikipedia.org/wiki/Алгоритъм) е доста неефикасен в сравнение с quicksort, heapsort или mergesort, ако се прилага върху големи списъци, но от друга страна има и доста предимства.

В най-лошия случай алгоритъмът има [сложност](http://bg.wikipedia.org/w/index.php?title=Изчислителна_сложност&action=edit&redlink=1) O(n2)

## Принцип на действие

1. Списъкът с елементи, които ще бъдат сортирани се разделя на две части: частта със сортираните елементи и частта с несортираните
2. При всяка стъпка се взема първия елемент от несортирания списък и се вмъква на правилната позиция в сортираната част от списъка
3. Сортирането продължава докато елементите от несортираната част на списъка се изчерпят

**Selection sort -** намира най-малкия елемент като сканира всички n елемента (n-1 сравнения) и го поставя на първа позиция в списъка. Намирането на следващият най-малък елемент изисква сканирането на останалите n-1 елемента и така, докато списъкът бъде подреден или общо сравненията са (n-1)+(n-2)+...+2+1 = Θ(*n*2). Всяко от тези сканирания изисква размяна на елементите или общо n-1 размени. Безполезен е, когато трябва да се сортират големи по обем списъци.

**Quick sort -** Основната част на алгоритъма се състои в сравняващо сортиране.

В най-лошия случай алгоритъмът има [сложност](http://bg.wikipedia.org/w/index.php?title=Изчислителна_сложност&action=edit&redlink=1) O(n2). В средния случай алгоритъмът има сложност О(nlogn), а в амортизирания се правят 1.386nlogn сравнения

## Принцип на действие

1. Избира се "главен" елемент от списъка с елементи, които ще бъдат сортирани.
2. Списъкът се пренарежда така, че всички елементи, които са по-малки от "главния" се поставят вляво от него, а всички, които са по-големи - вдясно от него.
3. Рекурсивно се повтарят горните стъпки върху списъка с по-малките и списъка с по-големите елементи.
4. Получените списъци се сливат (конкатенация) и се получава сортираният списък.

**Merge sort -** Основната идея е , че масивът се дели на 2 равни (почти) части в зависимост от четността на n и впоследствие лявата и дясна част се сортират и се сливат.

най-големият брой сравнения е n1+ n2 –1 .

n = 2k = 8= 23

Т(n) = 0 за n=1

T(n) = 2T(n/2)+ n – 1, за n>1

Tср (n) = O(nlogn)

procedure Mergesort ( i, j : integer);

begin

if i < j then

begin

m := [ (i + j ) / 2] ; { цялата част}

mergesort( i,m );

mergesort(m+1,j);

Сливане ( i,m,m+1,j );

end;

end;

Използва се прехвърляне на стойностите на масив, като сливащите се части се записват с помощен масив, което намалява бързодействието, но е в основа на всички сортировки бърху външен носител.