Трудоемкость. Сортировка. Поиск

«Для решения любой сколь угодно простой задачи можно написать программу, которая будет работать сколь угодно медленно». Афоризм, рожденный из практики приема курсовых работ.

Основные характеристики программы:

- производительность (время работы, время реакции)
- эффективность использования памяти
    - размещение данных в динамической памяти
    - Утечки памяти (Си++)
    - распределение обращений по адресам памяти (виртуальная память, подкачка и вытеснение страниц – кэширование, «пробуксовка»)

Основной параметр – объем (размерность) входных данных – **N**

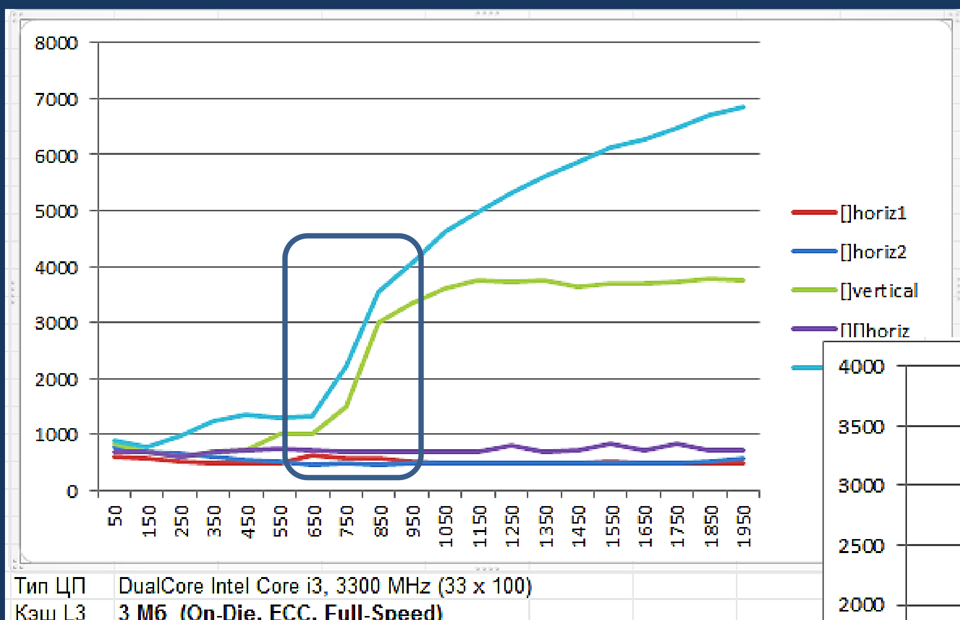
Производительность – свойство программы:

- «грязное» время работы программы:
    - зависимость от «железа»
    - зависимость от окружения
    - «смесь» операций

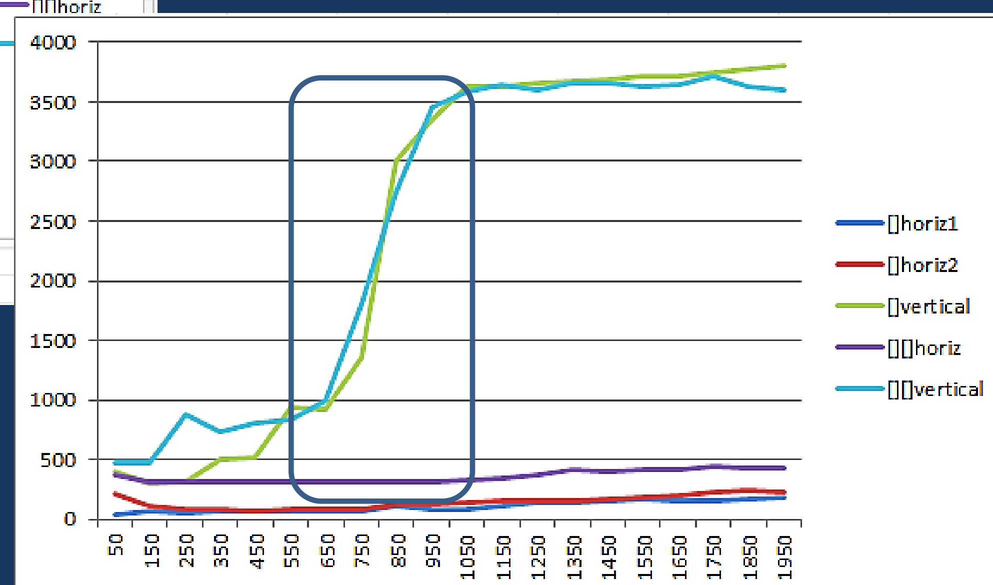
Пример дефекта производительности

Парадокс двумерного массива (http://habrahabr.ru/post/211747/)

- обработка по столбцам на порядок медленнее, чем по строкам.
- предположение – влияние процессорного кэша
- моделирование двумерного массива через одномерный [i][j]=[i*n+j]



двумерный массив Си++ – линейка одномерных массивов (построчно)



двумерный массив Java – массив ссылок на объекты – линейные массивы

Трудоемкость

Трудоемкость - зависимость количества массовых операций (сравнения, обмены, сдвиги, повторения цикла) от размерности обрабатываемых данных - свойство алгоритма. Функция трудоемкости - **T(N)**

Свойства трудоемкости:

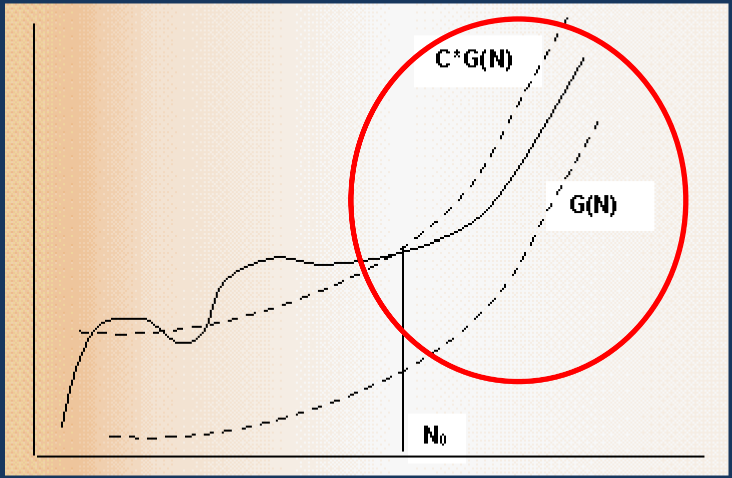
- определяется отдельно для каждого вида операций – **Tshift, Tstep, Tcall**
- чувствительность к данным - может зависеть от входных данных – статистическая величина: лучшее, худшее, среднее - T_{min}, T_{max}, T_{cp} .
- асимптотический характер оценки функции трудоемкости - скорость (степень) роста функции

Скорость роста для функции T(N) (обозначается как O(T(N)) – это такая функция G(n) если $\exists\ C, N_0 : \forall N > N_0\ C*G(N) > T(N)$, т.е. T(N) растет не быстрее G(N)

Проблема: C, N_0 сложно оценить

Свойства O(f(n)):

- f(n) = O(f(n))
- c*O(f(n)) = O(f(n))
- O(f(n))+O(f(n)) = O(f(n))
- O(O(f(n))) = O(f(n))
- O(f(n))*O(g(n)) = O(f(n)*g(n))

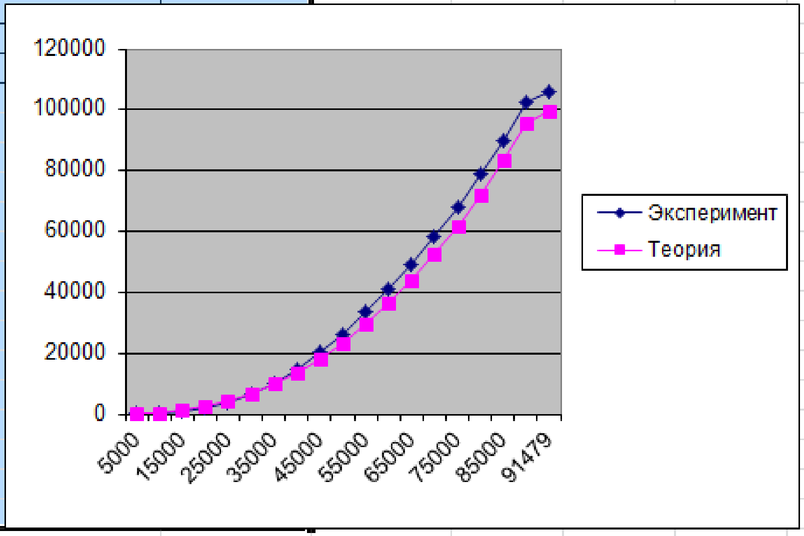


Трудоемкость

Сущность и использование трудоемкости:

- оценка производительности программы «в перспективе» по мере роста размерности данных **N**
- закономерности, проявляющиеся при достаточно большом **N.** Само понятие «достаточно большое» не определено.
- определяется для каждого вида операций
- использование: измерение T(N) и «грязного» времени работы для доступного **N,** оценка T(N) и «грязного» времени для новой размерности N_1 в соответствии с видом трудоемкости (см. CPROG – задания – оценка производительности программ, архив cprog/tutor/ozenka.xls)

	A	B	C	D	E	F
1	N	T1(эскп)	T1*(матем)	отклонение	коэф по N1	Степень
2	5000	93	93,0	0%	0,0000001233	2,4
3	10000	421	490,9	-14%		
4	15000	906	1298,9	-30%		
5	20000	1750	2590,8	-32%		
6	25000	3625	4426,0	-18%		
7	30000	6531	6855,6	-5%		
8	35000	10187	9924,7	3%		
9	40000	14765	13674,1	8%		
10	45000	20140	18141,2	11%		
11	50000	26343	23360,5	13%		
12	55000	33312	29364,7	13%		
13	60000	40843	36184,1	13%		
14	65000	49031	43847,7	12%		
15	70000	58250	52383,0	11%		
16	75000	68062	61816,2	10%		
17	80000	78687	72172,4	9%		
18	85000	90015	83475,8	8%		
19	90000	102218	95749,6	7%		
20	91479	105734	99569,5	6%		



Виды и источники трудоемкости

- **единичная** O(1) – постоянная, не зависящая от N, примитивная операция или «чудесная» трудоемкость (хеширование)
- **линейная** O(N) – *арифметическая прогрессия* T(N+1) = a + T(N), одиночный цикл for(i=0;i<N…)
- **логарифмическая** $O(\log_2 N)$ – «идеальная» по отношению к линейной – количество шагов деления пополам или удвоения. «Через логарифм не перепрыгнешь»
- **квадратичная –** $O(N^2)$ - двойной цикл, пары элементов
- **линейно-логарифмическая** $O(N*\log_2 N)$ – «идеальная» по отношению к квадратичной – количество шагов деления пополам или удвоения

- **степенная –** $O(N^m)$ – m вложенных циклов, каждый цикл добавляет 1 степень к трудоемкости

```
char s[…]; for(int i=0; i<strlen(s);i++)…s[]…
```

$O(N^2)$

- **экспоненциальная** – $O(m^N)$ – каждый шаг добавляет цикл, рекурсия с циклом, «цепная реакция», комбинаторный перебор, *геометрическая прогрессия* $m^{N+1}=m * m^N$ T(N+1) = m*T(N)
- **факториал** - O(N!) T(N+1) = N*T(N)

Шкалы роста размерностей

- **линейная** $O(N)$ – *арифметическая прогрессия* $T(N+1) = a + T(N)$, *количественная* шкала
- **экспоненциальная** – $O(m^N)$ – *геометрическая прогрессия* $m^{N+1}=m * m^N$ $T(N+1) = m*T(N)$, порядковая (*качественная*) шкала
- **числа Фибоначчи –** $Fb(N+1) = Fb(N)+Fb(N-1))$ – промежуточная *количественно-качественная* шкала $Fb(N) \approx 2^{0.694N} \approx 2^{N/2}$

Поиск

Терминология БД:

- **Запись –** единица данных, состоит из именованных элементов - **полей**
- **Ключ –** элемент записи, однозначно идентифицирующий запись (Иванов_1, Иванов_2. А.П.Чехов, «Жалобная книга»… за начальника станции Иванов седьмой)
- **Поиск по ключу -** получение всей записи по значению ключа

Алгоритмы поиска:

- неупорядоченная последовательность записей – линейный поиск
- упорядоченная последовательность – двоичный поиск
- поиск вычислением адреса – хеширование

Линейный поиск – последовательный перебор всех записей

- $T_{step,min} = 1$
- $T_{step,max} = N$
- $T_{step,mid} = N/2$

Поиск

Двоичный поиск – проверка середины интервала упорядоченных записей и выбор половины. Трудоемкость – «гарантированный логарифм»

- $T_{step,min} = 1$
- $T_{step,max} = \log_2(N)$
- $T_{step,mid} = \ldots$

```
//------------------------------------------------------46-01.cpp
//------Двоичный поиск в упорядоченном массиве
 int binary(int c[], int n, int val){    // Возвращает индекс найденного
 int a,b,m;                                   // Левая, правая границы и
 for(a=0,b=n-1; a <= b;) {          // середина
          m = (a + b)/2;               // Середина интервала
            if (c[m] == val)           // Значение найдено -
      return m;                          // вернуть индекс найденного
      if (c[m] > val)
           b = m-1;                      // Выбрать левую половину
      else
            a = m+1;                     // Выбрать правую половину
            }
return -1; }                           // Значение не найдено
```

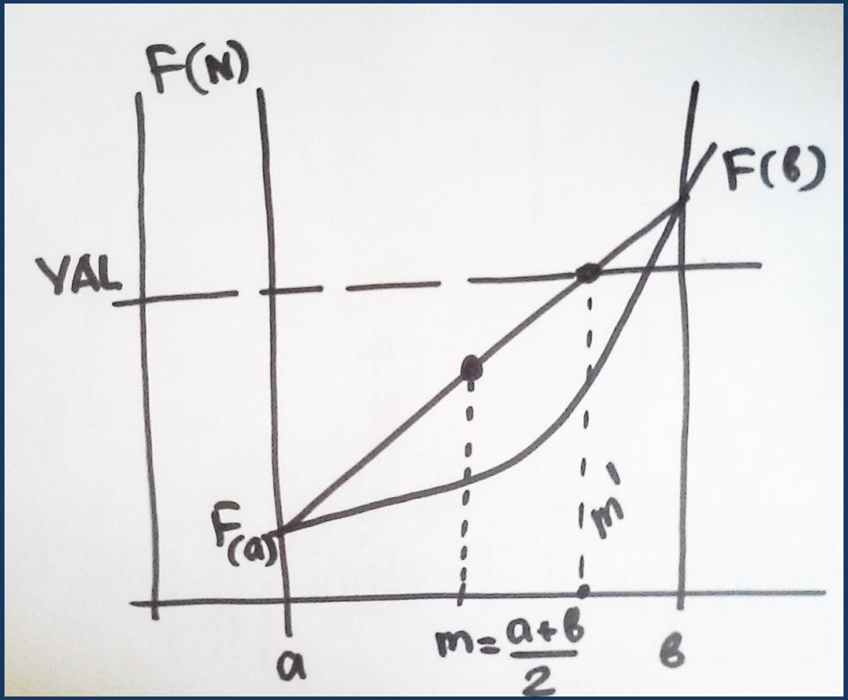
Поиск

Двоичный поиск – попытка улучшения (вар.10): гипотеза о линейном возрастании значений ключей на интервале

$(m-a) / (b-a) = (val-F(a)) / (F(b)-F(a))$

$T_{step,max} = N$ для a[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,…,10000} и val=1 будет перебирать m=0,1,2,3…

- в среднем – лучше
- в некоторых случаях - вырождение



Законы информационного поиска

Индекс в БД – таблица ссылок на основную таблицу в БД, упорядоченных по возрастанию индексируемого поля. При отсутствии индексирования поля поиск в нем линейный, при наличии – двоичный

- добавление записи – вставка индекса с сохранением порядка
- создание индекса – заполнение индекса 0…n-1 + сортировка индексов при сравнении указуемых элементов
- редактирование записи – изменение индексируемого поля: удаление и вставка (перемещение) индекса
- соединение записей по совпадению ключей (ссылок): сервер генерирует программу для select с линейным перебором записей таблицы Student и линейным (двоичным) поиском записей в Group

Индекс LastName

0	3
1	5
2	0
3	4
4	6
5	2
6	1

Student

ID	LastName	GroupId	Bitrh
0	Иванов	0	01.12.1988
1	Сидоров	2	01.12.1995
2	Петров	1	11.6.1812
3	Андреев	2	01.10.1965
4	Киреев	2	03.01.1965
5	Бухарин	1	5.5.1888
6	Ленин	0	22.4.1870

Индекс Birth

2	0
6	1
5	2
3	3
4	4
0	5
1	6

Group

ID	Name
0	АВТ-609
1	АВТ-610
2	АВТ-615

Иванов	АВТ-609
Сидоров	АВТ-615
Петров	АВТ-610
Андреев	АВТ-615
Киреев	АВТ-615
Бухарин	АВТ-610
Ленин	АВТ-609

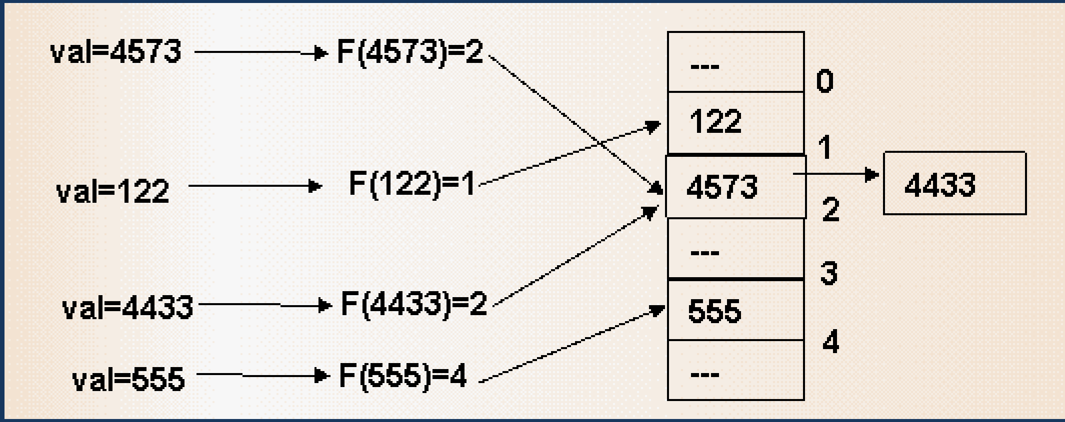
Select LastName,Group from Student,Group JOIN Student.GroupId = Group.ID

Поиск. Хеширование (cprog 8.7)

Хеширование – (hash - кромсать), поиск на основе размещения элемента данных в ячейке, индекс которой получен формальным преобразованием ключа A[i] = v -> i = Fh(v)

- «перемешивание» разрядов ключа, псевдо-случайный характер функции размещения
- столкновения (коллизии), когда два разных значения получают от функции один и тот же адрес размещения
- коллизии разрешаются либо за счет размещения «лишних» значений в соседних свободных ячейках, либо путем создания из них линейных цепочек, например, списков
- частота коллизий зависит от степени заполнения таблицы
- вырождение - все значения попадут в одну и ту же ячейку

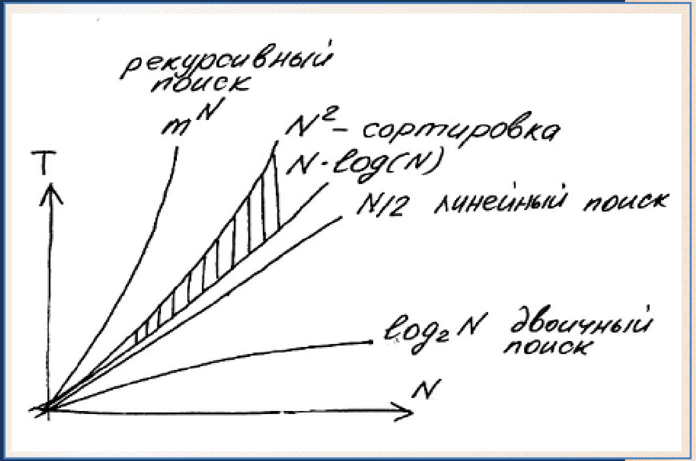
$T_{step,min} = 1$
$T_{step,mid} \approx 1$
$T_{step,max} = N$

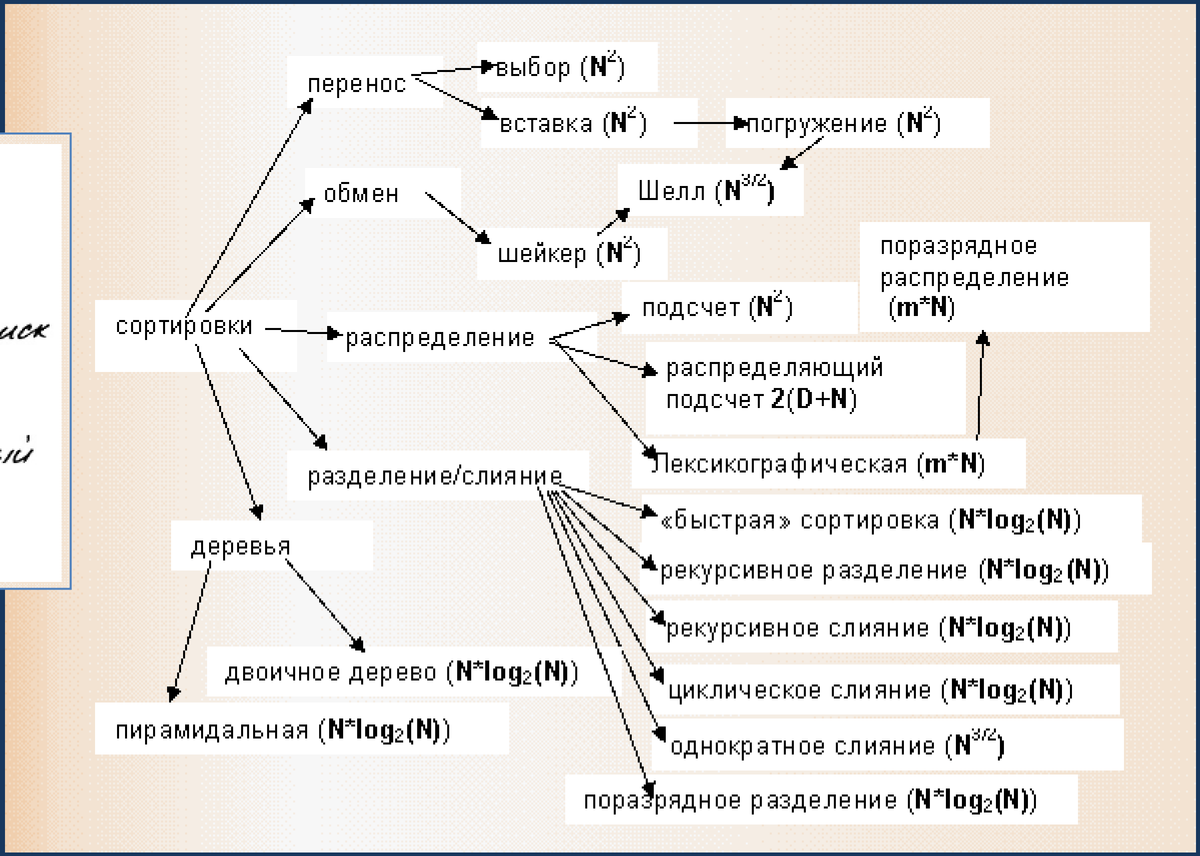


- в среднем – близок к 1
- в некоторых случаях - вырождение

Сортировка

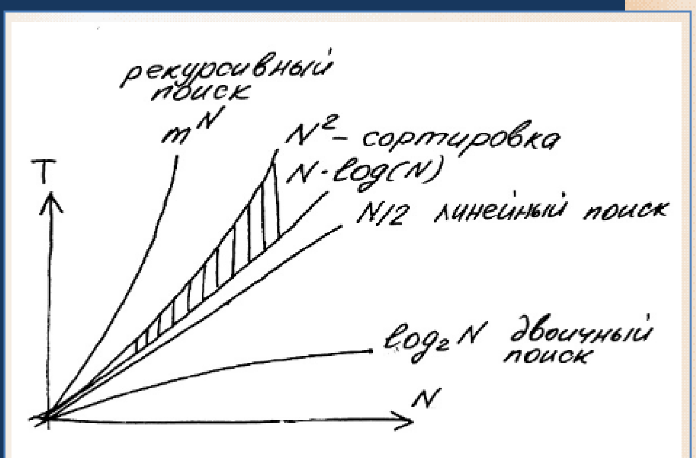
- многообразие способов (идей) решения задачи
- трудоемкость
  - квадратичная $O(N^2)$ – «тупые сортировки», каждый с каждым
  - линейно-логарифмическая - $O(N*\log_2 N)$ «хитрые» - для каждого элемента удается применить $\log_2 N$ шагов на основе удвоения/половинного деления
  - линейный – за счет дополнительной памяти

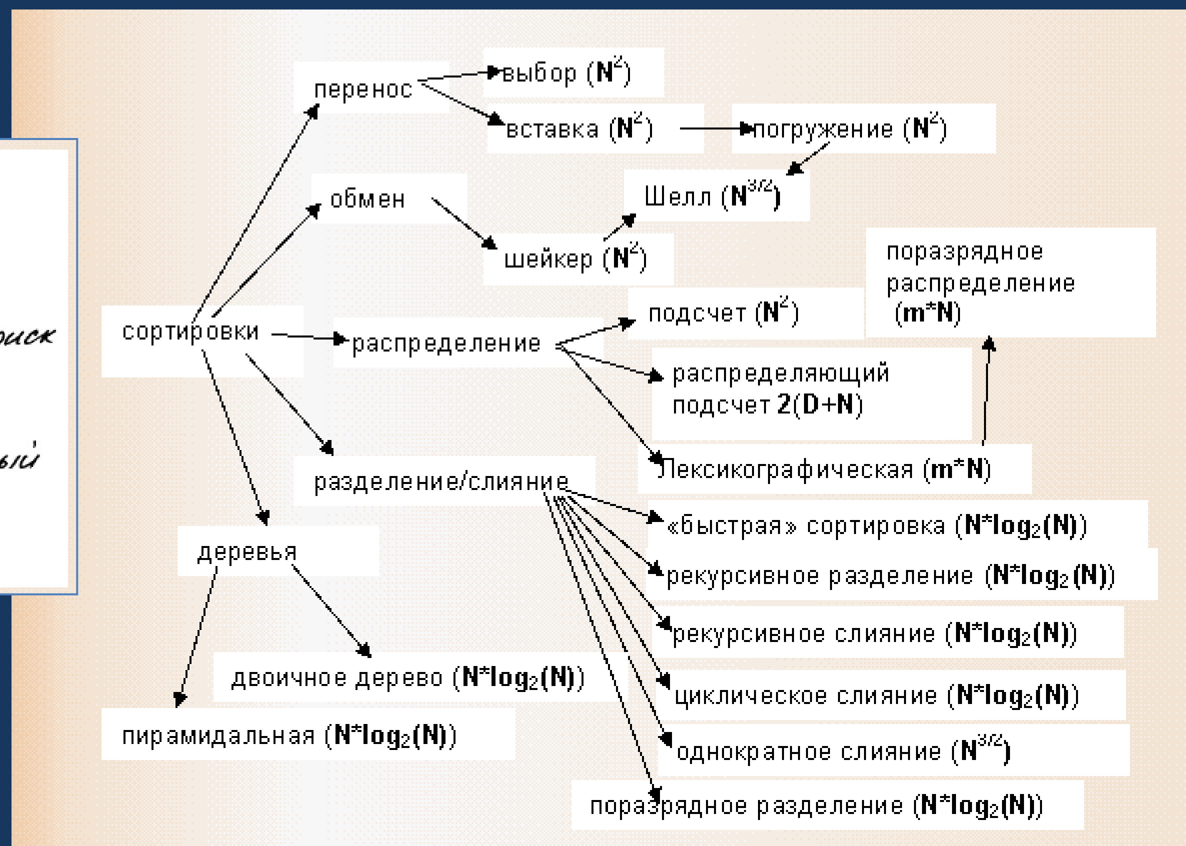




Сортировка

- многообразие способов (идей) решения задачи
- трудоемкость
  - квадратичная O(N^2) – «тупые сортировки», каждый с каждым
  - линейно-логарифмическая - O($N*\log_2 N$) «хитрые» - для каждого элемента удается применить $\log_2 N$ шагов на основе удвоения/половинного деления
  - линейный – за счет дополнительной памяти





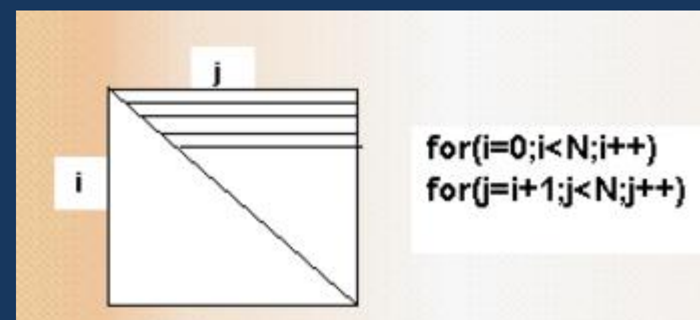
«Тупые» сортировки по N^2

Перенос – за 1 шаг один элемент переносится из неупорядоченной в упорядоченную часть:

- Выбор: минимальный из неупорядоченной в конец упорядоченной
- Вставка: любой (очередной) из неупорядоченной – вставка в упорядоченную с сохранением порядка
  - Вставка погружением
- Обмен: свойство упорядоченности - любая пара рядом стоящих упорядочена, алгоритм *по определению* «меняем пары до тех пор, пока…»

Выбор:

- Сигнатура – поиск min
- Нечувствительная к данным
- $T_{step,min} = T_{step,max} = T_{step,mid} = N^2/2$




```
void sort(int in[], int n){
for ( int i=0; i < n-1; i++){                // Для очередного i
        for ( int j=i+1, k=i; j<n; j++)      // k - индекс минимального
        if (in[j] < in[k]) k=j;                     // в диапазоне i..n-1
        int c=in[k]; in[k] = in[i]; in[i] = c;      // Три стакана для очередного
        }}                                          // и минимального
```

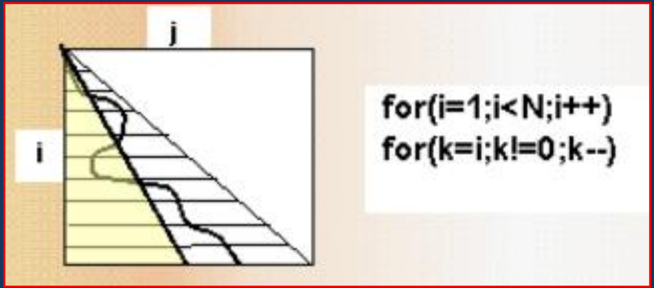
«Тупые» сортировки по N^2

Вставка:

- Сигнатуры нет
- Чувствительная к данным
- Погружение:
- $T_{step,min}$ = N-1
- $T_{step,max}$ = $N^2/2$
- $T_{step,mid}$= $N^2/4$

```
void sort(int in[], int n){
for ( int i=1; i < n; i++) {                   // Для очередного i
          int v=in[i];                         // сохранить очередной
          for (int k=0; k<i; k++)              // поиск места вставки
                    if(in[k]>v) break;         // перед первым, большим v
           for(int j=i-1; j>=k; j--)           // сдвиг на 1 вправо
                    in[j+1]=in[j];             // от очередного до найденного
          in[k]=v;                             // вставка очередного на место
          }}                                   // первого, большего него
```

```
//----------------------------------------------------------------
int A[20]={1,7,3,4,7,6,3,7,4,3}, n=10;
int k=2,vv=15,vv1;
for (int i=k;i<=n;i++){
          vv1=A[i];                  // Текущий «подбросить вверх»
          A[i]=vv;                   // На его место поместить новый
          vv=vv1; }                  // «Подброшенный» будет вставлят
n++;                               // позицию
```



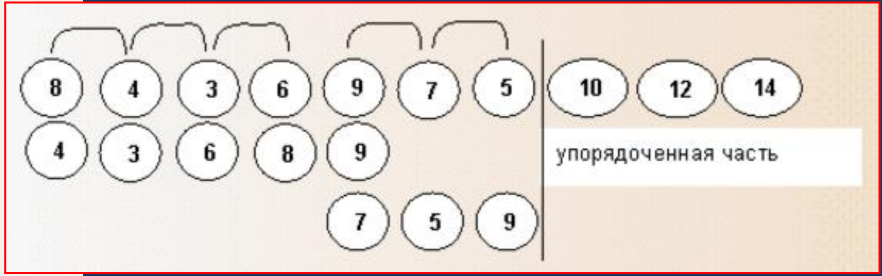

```
void sort(int in[],int n) {
for ( int i=1; i<n; i++)              // Пока не достигли " дна" или меньшего себя
          for ( int k=i; k !=0 && in[k] < in[k-1]; k--){
    int c=in[k]; in[k]=in[k-1]; in[k-1]=c;
    }}
```

«Тупые» сортировки по N^2

Обмен:

- Сигнатура – обмен соседей (или с шагом m)
- Чувствительная к данным (при оптимизации)
- Оптимизации:
  - Индикатор/счетчик обменов
  - Граница обмена – эффект «пузырька»
  - Шейкер
  - С переменным шагом – сортировка Шелла $T(N) = O(N^{3/2})$

```
void sort(int A[], int n){
int i,found;                          // Количество обменов
    do {                              // Повторять просмотр...
    found =0;
    for (i=0; i<n-1; i++)
            if (A[i] > A[i+1]) {      // Сравнить соседей
        int cc = A[i]; A[i]=A[i+1]; A[i+1]=cc;
        found++;                      // Переставить соседей
        }
    } while(found !=0); }             //...пока есть перестановки
```




```
void sort(int A[], int n){
int i,b,b1;                                  // b граница отсортированной части
for (b=n-1; b!=0; b=b1) {                    // Пока граница не сместится к правому краю
        b1=0;                                // b1 место последней перестановки
        for (i=0; i<b; i++)                  // Просмотр массива
                if (A[i] > A[i+1]) {         // Перестановка с запоминанием места
                int cc = A[i]; A[i]=A[i+1]; A[i+1]=cc;
                b1=i;
        }}}
```

«Тупые» сортировки по N^2

С переменным шагом – сортировка Шелла $T(N) = O(N^{3/2})$

```
//------ Сортировка Шелла с шагом по степеням 2 (погружение)
void shell(int A[], int n ){
        for (int m=1; m<n; m*=2);               // Определение последней степени 2
        for (m/=2; m!=0; m/=2)                  // Цикл с переменным шагом m=32,16,8..1
           for (int k=0; k<m; k++)              // Цикл по группам k=0..m-1
              for (int i=k+m; i<n; i+=m)        // Погружение с шагом с в группе k
                 for (int j=i; j>=m && A[j]<A[j-m]; j-=m){
                    int cc = A[j]; A[j]=A[j-m]; A[j-m]=cc;
                 }
}
```

```
// Сортировка Шелла. Формула  шага h=3h+1
// Погружение с уменьшающимся шагом
void sort(int A[], int n){
        int i,j,h;
        for (h=1; h<n/9; h=h*3+1);          // Определить максимальный шаг
        for (;h>0;h=h/3)                    // Одновременно просматриваются все группы
           for (i=h;i<n;i++)                // Погружение с шагом h
              for (j=i;j>=h && A[j]<A[j-h];j-=h)
                 { int c=A[j]; A[j]=A[j-h]; A[j-h]=c; }
        }
```

«Парадокс»: 3 вложенных цикла, трудоемкость меньше N^2 - $O(N^{3/2})$

Распределение – от $O(N^2)$ до $O(N)$

Сортировка подсчетом:

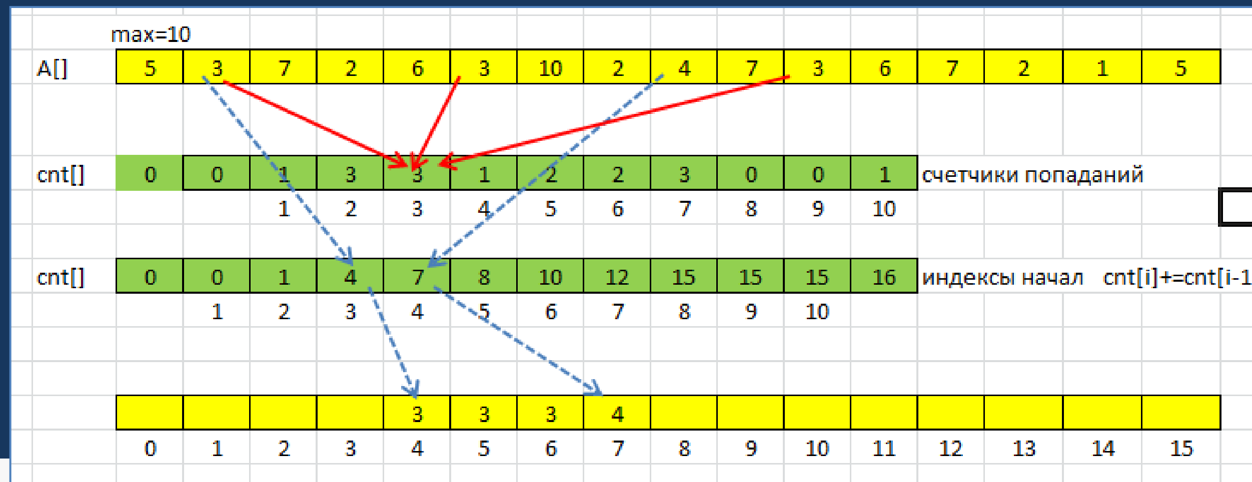
- Количество меньших очередного – место (индекс) в выходном
- $T_{step,min} = T_{step,max} = T_{step,mid} = N^2$
- Нечувствительная к данным
- Вывод: хуже не бывает

```
void sort(int in[],int n){
int i,j,cnt;
int *out=new int[n];                         // выходной массив
    for (i=0; i<n; i++){
        for ( cnt=0,j=0; j<n; j++)           // для in[i] подсчет
          if (in[j] < in[i]) cnt++;          // меньших его
        else                                 // а также равных ему
           if (in[j]==in[i] && j>i) cnt++;   // и стоящих слева
        out[cnt]=in[i];                      // место в выходном
        }                                    // определяется счетчиком
for (i=0; i<n; i++) in[i]=out[i];
delete []out;}
```

Распределение - O(N)

Распределяющий подсчет:

- Использование дополнительной памяти – счетчики повторов для каждого значения. Размерность массива – диапазон сортируемых значений (max)
- T_{step} = 2N + 2max
- Нечувствительная к данным
- Вывод: лучше не бывает




```
void sort(int A[], int n){
        int i,j,max;
        for (i=0,max=0; i<n; i++) if (A[i]>max) max=A[i];
        int *cnt=new int[max+2];                    // массив счетчиков по всему
        int *out=new int[n];                        // диапазону значений
        for (i=0; i<=max+1; i++) cnt[i]=0;
        for (i=0; i<n; i++) cnt[A[i]+1]++;          // накопление счетчиков значений
        for (i=1; i<=max;i++)                       // добавление к каждому счетчику
                cnt[i]+=cnt[i-1];                   // суммы предыдущих
        for (i=0;i<n;i++)                           // перенос в выходную позицию
                out[cnt[A[i]]++]=A[i];              // в соответствии со счетчиком предыдущих
        //-----------------------------------------------------------------
        for (i=0; i<n; i++) A[i]=out[i];            // Возвратить данные
        delete cnt;
        delete out;
}
```

Распределение - O(N)

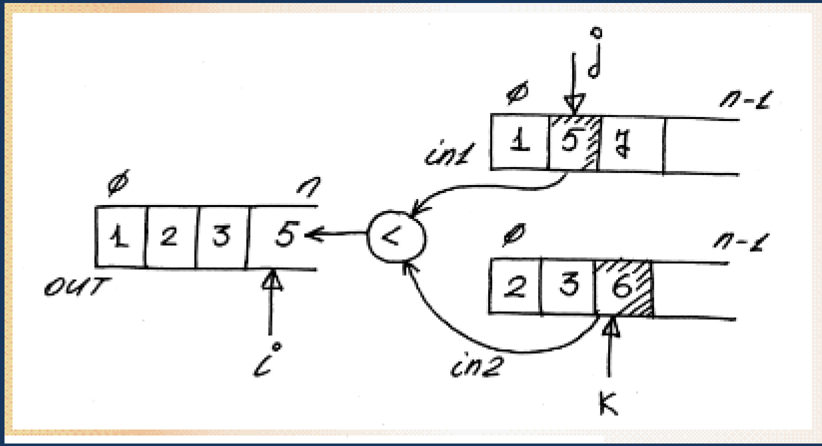
Лексикографическая сортировка:

- Распределение по значениям «цифр» или «символов» ключа по карманам
- Соединение карманов
- Повторение по количеству цифр в ключе (m), начиная с младшей
- T_{step} = mN

234	654	375	458	23	865	345	45	674	76	453	176	356	5	564	54
0															
1															
2							по мл.цифре								
3	23	453													
4	234	654	674	564	54										
5	375	865	345	45	5										
6	76	176	356												
7															
8	458														
9															
23	453	234	654	674	564	54	375	865	345	45	5	76	176	356	458
0	5														
1															
2	23	234				по средней цифре									
3															
4	345	45													
5	453	654	54	356	458										
6	564	865													
7	674	375	76	176											
8															
9															
5	23	234	345	45	453	654	54	356	458	564	865	674	375	76	176
0	5	23	45	54	76										
1	176														
2	234				по старшей цифре										
3	345	356	375												
4	453	458													
5	564														
6	654	674													
7															
8	865														
9															
5	23	45	54	76	176	234	345	356	375	453	458	564	654	674	865

Разделение/слияние

Слияние – соединение упорядоченных последовательностей (1 цикл)
Вопрос – откуда их взять???




```
//------------------------- Слияние упорядоченных последовательностей
 void sleave(int out[], int in1[], int in2[], int n){
 int i,j,k;                                   // Каждой последовательности - по индексу
 for (i=j=k=0; i<2*n; i++){
         if (k==n) out[i]=in1[j++];             // Вторая кончилась - сливать первую
         else
          if (j==n) out[i]=in2[k++];            // Первая кончилась - сливать вторую
         else                                  // Сливать меньший из очередных
         if (in1[j] < in2[k]) out[i]=in1[j++];
         else out[i]=in2[k++]; }}
```

Разделение/слияние

Однократное слияние – разделение на части, сортировка частей, многопутевое слияние частей

						B						
						6	6	11	12	14	x	
A				A[i]	B[k][0]	8	9	9	16	19	21	
1	2	2	4	5		5	16	17	18	22	23	x
						7	8	9	12	16	x	
					B[j][0]	6	8	11	12	x	x	
						7	8	x	x	x	x	

```
//-----------------------------------------------------------------------46
//------- Простое однократное слияние
void sort(int a[], int n);              // любая сортировка одномерного массива

void big_sort(int A[], int N){
int max=A[0],i,j,n=sqrt(N)+1;
int **B=new int*[n];
        for (i=0; i<n; i++) B[i]=new int[n];
        for (i=0; i<N; i++) {
                B[i/n][i%n]=A[i];
                if (A[i]>max) max=A[i];                 // Распределение
                }
        for (j=n*n-N; j<n;j++)                          // Заполнение "хвоста" последнего
                B[n-1][j]=max+1;
        for (i=0; i<n; i++) sort(B[i],n);               // Сортировка частей
        for (i=0; i<N; i++){                            // Слияние
        for ( int k=0, j=0; j<n; j++)                   // Индекс строки с минимальным
                if (B[j][0] < B[k][0]) k=j;             // начальным B[k][0]
        A[i] = B[k][0];                                 // Перенос элемента
        for (j=1; j<n; j++)
                B[k][j-1]=B[k][j];                      // Сдвиг сливаемой строки
        B[k][n-1]=max+1;                                // Запись ограничителя
        }
for (i=0; i<n; i++) delete []B[i];
delete []B;}
```

$T_0 = N^2/2$

$n=\sqrt{N}$

$T_{i\ sort}=n^2/2$

$T_{sort}=n*n^2/2 = n^3/2 = N^{3/2}/2$

$T_{merge}= N*n = N^{3/2}$

$T_{full} = 1.5N^{3/2} = O(N^{3/2})$

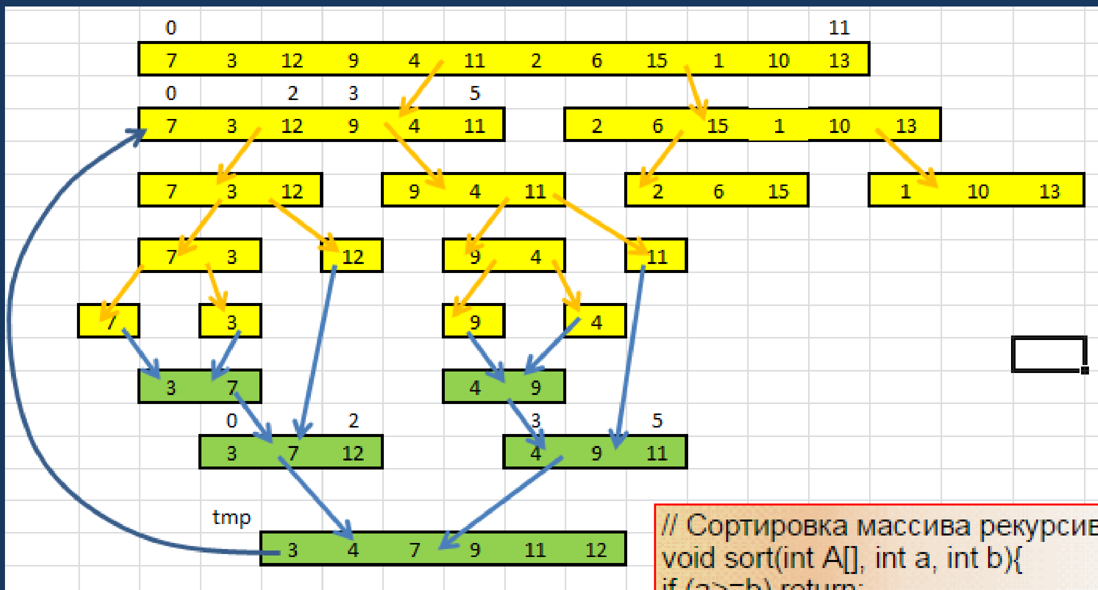
$T_0 / T_{full} = \sqrt{N}/3$

при $T_0= N*\log_2 N$

эффект обратный

Разделение/слияние

Рекурсивное слияние – (cprog 7.2) рекурсивное разделение на 2 части, пока не станут <=1, обратное попарное слияние



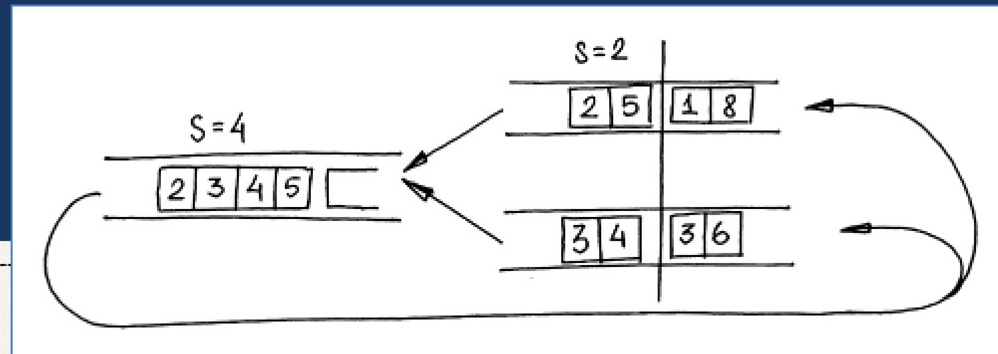
$T(N) = O(N*log_2N)$ – log_2N раз слияние из N элементов, нечувствительна к данным, требуется памяти для слияния

```
// Сортировка массива рекурсивным слиянием
void sort(int A[], int a, int b){
if (a>=b) return;                          // Разделение закончилось
int m=(a+b+1)/2,i,j,k;                     // Нет - взять середину интервала
sort(A,a,m-1);                             // Рекурсивный вызов для частей
sort(A,m,b);
int *tmp=new int[b-a+1];                   // Слияние частей во временный массив
for (i=a,j=m,k=0; k<=b-a; k++)
        if (i==m || j!=b+1 && A[j]<A[i])
                tmp[k]=A[j++];             // слить из второй части, если
        else                               // первая кончилась или во второй меньше
                tmp[k]=A[i++];             // слить из первой части
for (i=a,j=0; i<=b; i++,j++)
        A[i]=tmp[j];                       // вернуть слитые части обратно в A
delete tmp;      }                         // удалить временный массив
```

Разделение/слияние

Циклическое слияние – слияние из двух последовательностей группами по s элементов (s на каждом шаге удваивается)

T(N) = O(N*log$_2$N) – log$_2$N раз слияние из N элементов, нечувствительна к данным, требуется 2N памяти




```
//---------------------------------------------------------------------------
//------ Циклическое двухпутевое слияние
void sort(int A[], int n){
int i,i1,i2,s,k;
for (s=1; 1; s*=2){                                // Размер группы кратен степени 2
        int nn=n/s;                                // Количество групп по s элементов
        if (n%s!=0) nn++;                          // Остаток – есть неполная группа
        int n1=nn/2*s;                             // Деление ближе к середине,
        int n2=n-n1;                               // но кратно размеру группы
        if (n1<=0 || n2<=0) return;                // Часть больше целого - выход
        int *B1=new int[n1],*B2=new int[n2];
        for (i=0; i<n1; i++) B1[i]=A[i];           // Разделение на части
        for (i=0; i<n2; i++) B2[i]=A[i+n1];
        i1=i2=0;
        for (i=0,k=0; i<n; i++){                   // Слияние с переходом «скачком»
                if (i1==s && i2==s)                // при достижении границ обеих
                        k+=s,i1=0,i2=0;            // групп
                if (i1==s || k+i1==n1) A[i]=B2[k+i2++];  // Достигла границы группы или
                else                               // массива
                if (i2==s || k+i2==n2) A[i]=B1[k+i1++];
                else                               // Если нет – минимальный из пары
                if (B1[k+i1 ] < B2[k+i2 ]) A[i]=B1[k+i1++];
                else A[i]=B2[k+i2++];
                }
        delete []B1; delete []B2;
        }}
```

k – начало групп
i1,i2 – текущий в группе
s – длина группы

работает при любой размерности массива – деление на две части, первая кратна размеру группы

Рекурсивное разделение

Быстрая сортировка

```
void sort(int in[], int a, int b){
int i,j,mode;
if (a>=b) return;
for (i=a, j=b, mode=1; i < j; mode >0 ? j-- : i++)
                        if (in[i] > in[j]){
            int c = in[i]; in[i] = in[j]; in[j]=c;
            mode = -mode;
            }
sort(in,a,i-1); sort(in,i+1,b);}
```

a				b	mode до	
7	4	9	2	6	1	обмен
6	4	9	2	7	-1	i++
6	4	9	2	7	-1	i++
2	4	9	6	7	-1	обмен
2	4	7	6	9	1	j--
2	4	7	6	9	1	обмен
2	4	6	7	9	-1	i++
			a,b			
2	4	6	7	9	Рекурсия	

медина – самое левое значение
количество шагов = длина диапазона -1
разделение обменом на две части, больших и меньших медианы
медиана – между частями
рекурсивный вызов для частей

Рекурсивное разделение

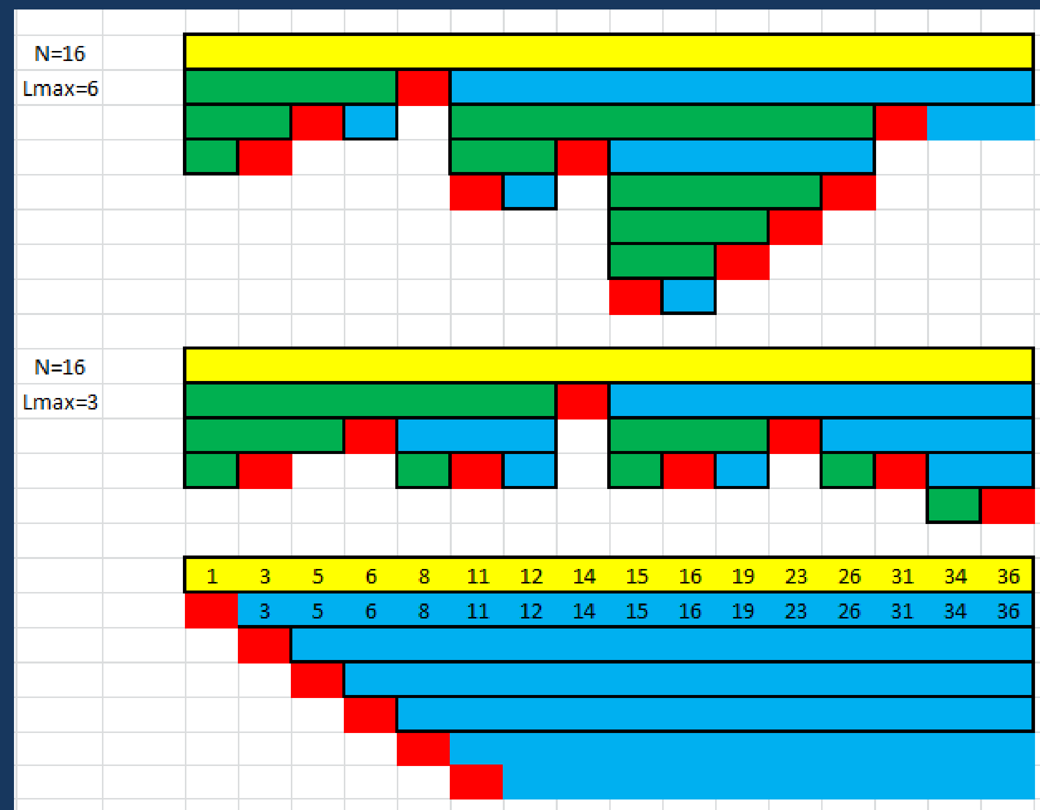
Трудоемкость

$T_{step,min}$ = $N\log_2(N)$ – идеальное разделение на равные части (без учета медианы), сумма шагов на уровне = N, уровней разделения $\log_2(N)$

$T_{step,mid}$ = …

$T_{step,max}$ = $N^2/2$ – вырожденный случай, медиана слева, сортировка упорядоченного массива

$T_{swap,min}$ = 0 – обменов нет



Рекурсивное разделение

Разделение:

- среднее арифметическое – медиана
- два движка, оставляют слева и справа элементы меньшие и большие медианы
- «упираются» в пару элементов, расположенных **наоборот**
- меняют их местами
- if (i==a) return – для случая «все одинаковые»

```
void sort(int in[], int a, int b){
int i,j,mode;
double sr=0;
if (a>=b) return;
for (i=a; i<=b; i++) sr+=in[i];
sr=sr/(b-a+1);
for (i=a, j=b; i <= j;)
        {
        if (in[i]< sr) { i++; continue; }
        if (in[j]>=sr) { j--; continue; }
        int c = in[i]; in[i] = in[j]; in[j]=c;
        i++,j--;
        }
if (i==a) return;
sort(in,a,j); sort(in,i,b);}
```

Древовидные структуры (cprog 8.4,8,5)

- Дерево, упорядоченное по вертикали – включение с вытеснением, слияние с замещением.
- Пирамидальная сортировка - дерево в массиве n -> 2n, 2n+1 (пирамида)
- Двоичное дерево – слева меньшие, справа большие
- Трудоемкость от линейно-логарифмической до квадратичной (вырождение)