Сортировки, продолжение

Быстрые сортировки — quicksort, mergesort, heapsort.

```
void QuickSort (int n, double *a)

 int QuickSortSplitting (int n, double *a)
 {
     int left = -1, right = n;
     double v = 0.5*(a[0] + a[n-1]);
     while (true) {
        while (a[++left] < v);
        while (a[--right] > v);
        // утв: a[left] >= v  && a[right] <= v
        if (left >= right) {
            return (right < n-1) ? right+1 : rignt;
        } else {
            swap(a+left, a+right);
        }
     }
 }

 void QuickSort(int n, double *a)
 {
     int k;
     if (n<2) return;
     k = QuickSortSplitting(n,a);
     QuickSort(k,a);
     QuickSort(n-k,a+k);
 }

  void QuickSort(int n, double *a)
  {
     int k;
     while ( n>1 )
     {
        k = QuickSortSplitting(n,a);
        if ( k < n-k ) {
            QuickSort(k, a);
            a = a+k;
            n = n-k;
        } else {
            QuickSort(n-k, a+k);
            n = k;
        }
     }
  }
```

Сортировка слиянием.

```
void MergeSort(int n, double *a, double *b)
{
    int k = n/2
    if (n<2) return;
    MergeSort(k, a);
    MergeSort(n-k, a+k);
    Merge(k, a, n-k, a+k, b);
    Copy(n, a, b);    // a <= b
}

void MergeParts(int n, double *a, int partlen, double *b)
{  // слияние упорядоченных фрагментов
   int k;
   int npairs = n / (2*partlen);
   int lastlen = n % (2*partlen);

   if (partlen > n) return;
   for (k = 0; k < nPairs; k++) {
       Merge(partlen, a + k*partlen,
             partlen, a + (k+1)*partlen, b + k*partlen);
   }
   if (lastlen > partlen) Merge(partlen, a + k*partlen,
                                lastlen, a + (k+1)*partlen, b + k*partlen);
   Copy(n, a, b);
}

void MergeSort(int n, double *a, double *b)
{
    int k;
    for (k=1; k<n; k*=2) {
        MergeParts(n, a, k, b);
    }
}
```

Слияние без дополнительной памяти — возможно ли?

Да, за $O(n \log n)$ т.е. трудоемкость сортировки будет $O(n \log^2 n)$

Идея такого слияния состоит в следующем. Пусть A исходный массив длины k, в котором есть два упорядоченных участка $A1, A2$ с длинами $k1, k2$, $k1 + k2 = k$. Требуется объединить эти две упорядоченные части, чтобы весь массив стал упорядоченным. Пусть для определенности $k1 \geq k2$ (противоположное неравенство обрабатывается симметричным алгоритмом). Рассмотрим "средний" элемент в части $A1$, пусть это элемент `a[i]`. Так как он "средний" в $A1$, то $i = k1/2 \geq k/4$. Теперь найдем в части $A1$ позицию, в которую мог бы быть вставлен этот элемент `a[i]`. Пусть это будет позиция `j`. Исходный массив разбился на 4 части $A11, A12, A21, A22$ по позициям `i, k1, j`. При этом по построению выполненны неравенства $x \leq a_i$ для $x \in A11$ или $x \in A21$ и $x \geq a_i$ для $x \in A12$ или $x \in A22$. Выполним циклический сдвиг (обмен) двух центральных частей, получим массив $A11, A21, A12, A22$. Теперь все элементы из $A11, A21$ не превосходят `a[i]`, а все элементы $A12, A22$ не меньше `a[i]`. Поскольку

части $A11, A21, A12, A22$ являются упорядоченными, мы можем применить рекурсию для слияния как $A11, A21$, так и $A12, A22$. В результате получится полностью упорядоченный массив. Так как длины $A11$ и $A12$ есть величины порядка $k/4$, то глубина рекурсии будет не хуже, чем $\log_{4/3}$.

Да, за $O(n)$, но способ очень извращенный и практического смысла не имеет
Но не забываем про рекурсию, которая требует $O(\log n)$ памяти.

Сортировка кучей.

Числа, записанные в массиве образуют пирамиду (или кучу — heap), если для любого элемента a[i] выполнены условия a[i] >= a[2*i + 1] и a[i] >= a[2*i + 2] для все индексов, не выходящих за границы массива.

Иллюстрация. "Геометрический" смысл пирамиды.

```
void SiftUp(double *a, int k)
{  // пополнение пирамиды с конца
   int j;
   while(k>0) {
      j = (k-1)/2;
      if (a[k] > a[j]) { swap(a+k, a+j); }
      else { break; }
      k = j;
   }
}

void SiftDown(double *a, int n)
{   // восстановление пирамиды при изменении a[0]
    int i, i1, i2;
    if (n<2) return;
    for (i=0; i<n; ) {
       i1 = 2*i + 1;
       i2 = i1 + 1;
       if (i1 >= n) break;
       if (i2 < n && a[i1] < a[i2]) {
           i1 = i2;
       }
       if (a[i] < a[i1]) {
           swap(a+i, a+i1);
           i = i1;
       } else {
           break;
       }
    }
}
 void HeapSort(int n, double *a)
{
    int k;
    for (k=1; k<n; k++)
    {
```

```
            SiftUp (a, k);
        }
        for (k=n-1; k>0; k--)
        {
            swap(a, a+k);
            SiftDown(a, k);
        }
    }
```

Сортировка подсчетом

Сортировка подсчетом (вариант с "генерацией" + вариант с перестановками).

```
   a  - исходный массив длины na
   b  - массив количеств или позиций длины nb=p
   c  - массив результата

void CountSort (int *a, int *b, int na, int nb)
{   int i,j=0;
    for (i=0; i<nb; i++) b[i] = 0;
    for (i=0; i<na; i++) b[a[i]]++;
    for (i=0; i<nb; i++) for ( ; b[i]!=0; b[i]--) a[j++] = i;
}

void CountSort2 (int *a, int *b, int *c, int na, int nb)
{   int i,s=0, s1;
    for (i=0; i<nb; i++) b[i] = 0;
    for (i=0; i<na; i++) b[a[i]]++;
    for (i=0; i<nb; i++) { s1 = s; s+=b[i]; b[i] = s1; }
    for (i=0; i<na; i++) c[b[a[i]]++] = a[i];
}
```

Трудоемкость $O(na + nb)$, память $nb + na$

Напрямую редко применяется, но как составная часть других сортировок может быть весьма эффективна.