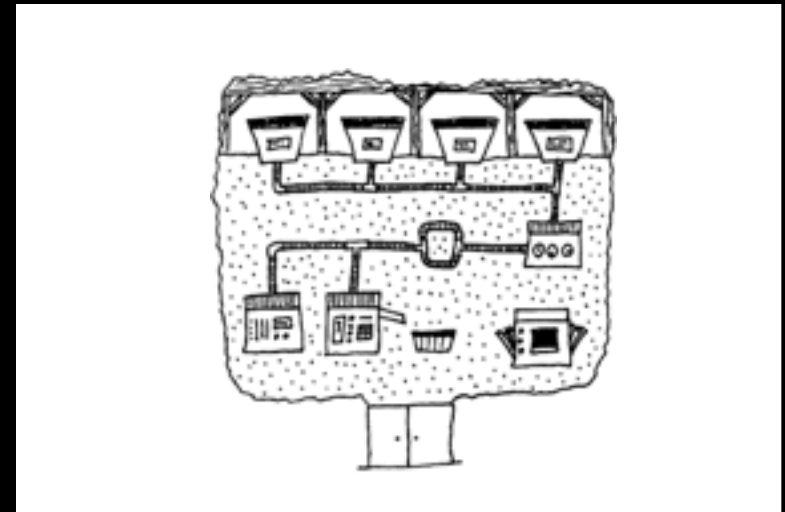




Алгоритмы и структуры данных



Сортировки: часть 2



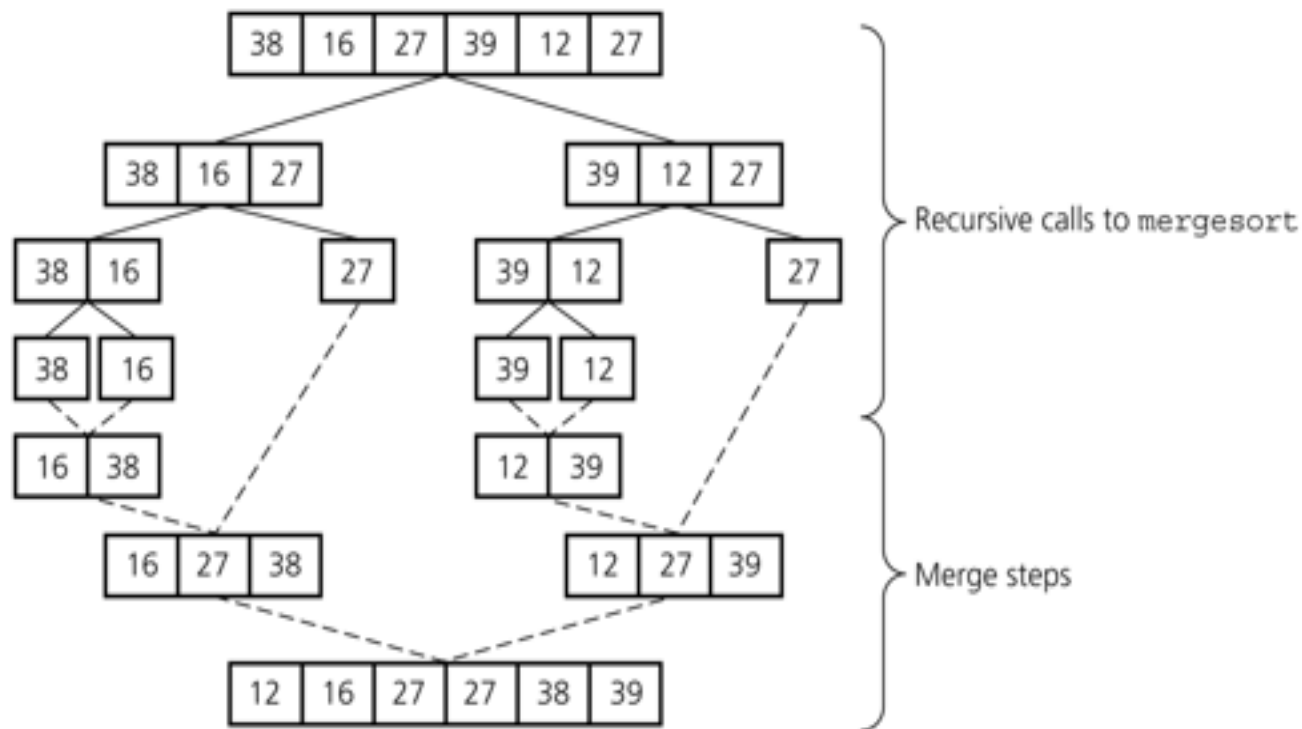
1. Слияние
2. Сравнение
3. Порязрядная

Сортировка слиянием



- Разбить массив на 2^k частей размером не больше m .
- Отсортировать каждую часть другим алгоритмом
- Слить 1 и 2, 3 и 4, ... $n-1$ и n части.
- Повторить шаг 3, пока не останется одна часть

Сортировка слиянием



Слияние 2х упорядоченных массивов



- Выберем массив, крайний элемент которого меньше
- Извлечём этот элемент в массив-результат
- Продолжаем, пока один из массивов не опустеет
- Копируем остаток второго массива в конец массива-результата

Слияние 2х упорядоченных массивов



```
void merge(int *a, int a_len, int *b, b_len,
int *c) {
    int i=0; int j=0;
    for (;i < a_len and j < b_len;) {
        if (a[i] < b[j]) {
            c[i+j] = a[i];
            ++i;
        } else {
            c[i+j] = b[j];
            ++j;
        }
    }
    if (i==a_len) {
        for (;j < b_len;++j) { c[i+j] = b[j]; }
    } else {
        for (;i < a_len;++i) { c[i+j] = a[i]; }
    }
}
```

Слияние 2х упорядоченных массивов



- Сложность $O(n+m)$
- Количество сравнений
 - Лучшее: $\min(n,m)$
 - Худшее $n+m$

- Дано k упорядоченных массивов суммарным размером n : A_1, A_2, \dots, A_k
- Построим сбалансированное бинарное дерево с массивами $A_1..A_k$ в листьях
- В узловых вершинах будем хранить указатель на минимальный элемент поддерева
- Изъятие элемента из узловой вершины – изъятие из минимального из дочерних узлов
- При изъятии элемента из списка, его размер уменьшается на 1
- Если список пуст – его сосед перемещается место родительской узловой вершины
- При изъятии происходит одно сравнение на каждом уровне дерева
- Высота дерева $\log(k)$
- Итого $O(n \cdot \log(k))$



К-путевое слияние



- Экономия на операциях копирования: n
- Количество сравнений такое же как и при $\log(k)$ 2х путевых слияниях

К-путевое слияние



- Построить кучу из первых элементов k массивов $O(k)$
- Скопировать минимум из кучи в результат $O(1)$
- Заменить минимальный элемент в куче следующим из того же массива $O(1)$
- Если массив пуст – извлечь элемент из кучи
- Восстановить порядок элементов в куче $O(\log(k))$
- Повторить пока куча не пуста
- Итого $O(k + n \cdot \log(k))$



Сравнение сортировок

Name	Best	Average	Worst	Memory	Stable
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$ on average, worst case is n	typical in-place sort is not stable; stable versions exist
Merge sort	$n \log n$	$n \log n$	$n \log n$	Depends ^[1] <i>further explanation needed</i> , worst case is n	Yes
In-place merge sort	—	—	$n (\log n)^2$	1	Yes
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No
Insertion sort	n	n^2	n^2	1	Yes
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No
Selection sort	n^2	n^2	n^2	1	No
Timsort	n	$n \log n$	$n \log n$	n	Yes
Shell sort	n	$n(\log n)^2$ or $n^{3/2}$	Depends on gap sequence; best known is $n(\log n)^2$	1	No
Bubble sort	n	n^2	n^2	1	Yes
Binary tree sort	n	$n \log n$	$n \log n$	n	Yes
Cycle sort	—	n^2	n^2	1	No
Library sort	—	$n \log n$	n^2	n	Yes
Patience sorting	—	—	$n \log n$	n	No
Smoothsort	n	$n \log n$	$n \log n$	1	No
Strand sort	n	n^2	n^2	n	Yes
Tournament sort	—	$n \log n$	$n \log n$	$n^{[4]}$	
Cocktail sort	n	n^2	n^2	1	Yes
Comb sort	n	$n \log n$	n^2	1	No
Gnome sort	n	n^2	n^2	1	Yes
Franceschini's method ^[5]	—	$n \log n$	$n \log n$	1	Yes

Как сортировать без сравнений



Сортировка подсчётом



Algorithm 3.10: CountingSort(R)

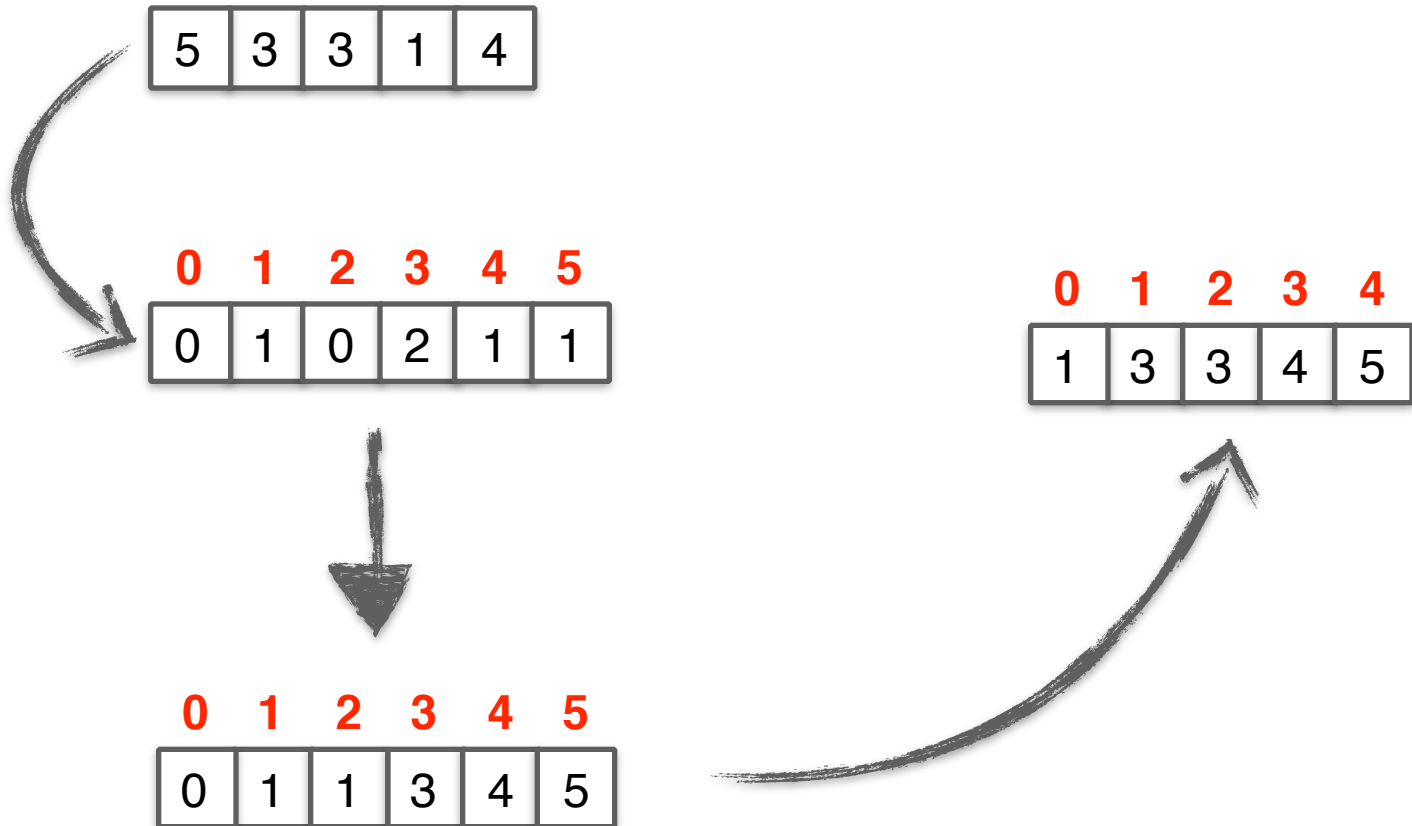
Input: (Multi)set $R = \{k_1, k_2, \dots, k_n\}$ of integers from the range $[0..\sigma)$.

Output: R in nondecreasing order in array $J[0..n)$.

```
(1) for  $i \leftarrow 0$  to  $\sigma - 1$  do  $C[i] \leftarrow 0$ 
(2) for  $i \leftarrow 1$  to  $n$  do  $C[k_i] \leftarrow C[k_i] + 1$ 
(3)  $sum \leftarrow 0$ 
(4) for  $i \leftarrow 0$  to  $\sigma - 1$  do           // cumulative sums
(5)      $tmp \leftarrow C[i]$ ;  $C[i] \leftarrow sum$ ;  $sum \leftarrow sum + tmp$ 
(6) for  $i \leftarrow 1$  to  $n$  do           // distribute
(7)      $J[C[k_i]] \leftarrow k_i$ ;  $C[k_i] \leftarrow C[k_i] + 1$ 
(8) return  $J$ 
```



Сортировка подсчётом



Сортировка подсчётом



```
1 void jsw_countsort ( int a[], int m, int n )
2 {
3     int *aux = malloc ( n * sizeof *aux );
4     int *count = calloc ( m + 1, sizeof *count );
5     int i;
6
7     for ( i = 0; i < n; i++ )
8         ++count[a[i] + 1];
9
10    for ( i = 1; i < m; i++ )
11        count[i] += count[i - 1];
12
13    for ( i = 0; i < n; i++ )
14        aux[count[a[i]]++] = a[i];
15
16    for ( i = 0; i < n; i++ )
17        a[i] = aux[i];
18
19    free ( count );
20    free ( aux );
21 }
```



Сортировка подсчётом



- + $O(n)$ - linear time**
- + stable**
- + нет сравнений**
- требует $O(n)$ памяти**
- $O(n)$ перемещений**



LSD raddix sort



Algorithm 3.11: LSDRadixSort(\mathcal{R})

Input: Set $\mathcal{R} = \{S_1, S_2, \dots, S_n\}$ of strings of length m over the alphabet $[0..\sigma)$.

Output: \mathcal{R} in increasing lexicographical order.

- (1) for $\ell \leftarrow m - 1$ to 0 do CountingSort(\mathcal{R}, ℓ)
- (2) return \mathcal{R}



LSD raddix sort

A	00001	R	10010	T	10100	X	11000	P	10000	A	00001	A	00001
S	10011	T	10100	X	11000	P	10000	A	00001	A	00001	E	00101
O	01111	N	01110	P	10000	L	01100	I	01001	R	10010	E	00101
R	10010	X	11000	A	00001	I	01001	A	00001	S	10011	G	00111
T	10100	P	10000	E	00101	R	10010	T	10100	T	10100	I	01001
I	01001	L	01100	A	00001	S	10011	L	01100	E	00101	L	01100
N	01110	A	00001	M	01101	E	00101	M	01101	G	00111	M	01101
G	00111	S	10011	R	10010	N	01110	N	01110	X	11000	N	01110
E	00101	O	01111	S	10011	O	01111	O	01111	I	01001	O	01111
X	11000	I	01001	G	00111	G	00111	G	00111	L	01100	P	10000
A	00001	G	00111							M	01101	R	10010
M	01101	E	00101							N	01110	S	10011
P	10000	A	00001							O	01111	T	10100
L	01100	M	01101							O	01111	X	11000
E	00101	E	00101										



LSD raddix sort



```
3
4 void jsw_radix_pass ( int a[], int aux[], int n, int radix )
5 {
6     int i;
7     int count[RANGE] = {0};
8
9     for ( i = 0; i < n; i++ )
10         ++count[ digit ( a[i], radix ) + 1 ];
11
12     for ( i = 1; i < RANGE; i++ )
13         count[i] += count[i - 1];
14
15     for ( i = 0; i < n; i++ )
16         aux[ count[ digit ( a[i], radix ) ]++ ] = a[i];
17
18     for ( i = 0; i < n; i++ )
19         a[i] = aux[i];
20 }
21
```



LSD raddix sort



- + $O(n \cdot \text{key len})$ время работы**
- + $O(n)$ время работы**
- + stable**
- can't use unstable sort for buckets**



MSD raddix sort



Algorithm 3.15: MSDRadixSort(\mathcal{R}, ℓ)

Input: Set $\mathcal{R} = \{S_1, S_2, \dots, S_n\}$ of strings over the alphabet $[0..\sigma)$ and the length ℓ of their common prefix.

Output: \mathcal{R} in increasing lexicographical order.

- (1) if $|\mathcal{R}| < \sigma$ then return StringQuicksort(\mathcal{R}, ℓ)
- (2) $\mathcal{R}_\perp \leftarrow \{S \in \mathcal{R} \mid |S| = \ell\}$; $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{R}_\perp$
- (3) $(\mathcal{R}_0, \mathcal{R}_1, \dots, \mathcal{R}_{\sigma-1}) \leftarrow \text{CountingSort}(\mathcal{R}, \ell)$
- (4) for $i \leftarrow 0$ to $\sigma - 1$ do $\mathcal{R}_i \leftarrow \text{MSDRadixSort}(\mathcal{R}_i, \ell + 1)$
- (5) return $\mathcal{R}_\perp \cdot \mathcal{R}_0 \cdot \mathcal{R}_1 \cdots \mathcal{R}_{\sigma-1}$



MSD raddix sort

0	1	0	0	0
1	0	0	0	0
0	1	1	0	0
0	0	1	1	1
0	1	1	1	0
1	0	1	0	1
1	0	0	1	0
1	0	0	0	0
0	0	1	0	1
0	1	1	1	0
1	1	0	1	1
1	1	1	0	1
0	1	1	0	1
1	0	1	1	1
0	0	0	0	1
1	0	1	0	1

0	1	0	0	0
0	0	0	0	1
0	1	1	0	0
0	0	1	1	1
0	1	1	1	0
0	1	1	0	1
0	1	1	1	0
0	0	1	0	1
1	0	0	0	0
1	0	0	1	0
1	1	0	1	1
1	1	1	0	1
1	0	1	0	1
1	0	1	1	1
1	0	0	0	0
1	0	1	0	1

0	0	1	0	1
0	0	0	0	1
0	0	1	1	1
0	1	1	0	0
0	1	1	1	0
0	1	1	0	1
0	1	1	1	0
0	1	0	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	0	1
1	0	1	1	1
1	1	1	0	1
1	1	0	1	1

0	0	0	0	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
0	1	1	0	1
0	1	1	1	0
0	1	1	0	0
1	0	0	0	0
1	0	0	1	0
1	0	0	0	0
1	0	1	0	1
1	0	1	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1

0	0	0	0	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	0
1	0	0	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1

0	0	0	0	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	0
1	0	0	0	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1



MSD raddix sort



- + $O(n \cdot \text{key_len})$ время работы*
- + $O(n)$ памяти
- + нет сортировки частей размером 1
- + could be unstable
- рекурсивная реализация



MSD vs LSD



362	291	207	207
436	362	436	253
291	253	253	291
487	436	362	362
207	487	487	397
253	207	291	436
397	397	397	487

LSD Radix Sorting:
Sort by the last digit, then
by the middle and the first one

237	237	216	211
318	216	211	216
216	211	237	237
462	268	268	268
211	318	318	318
268	462	462	460
460	460	460	462

MSD Radix Sorting:
Sort by the first digit, then sort
each of the groups by the next digit



Как сортировать ключи разной длины?



- * расширить алфавит пустым символом**
- сложность LSD ~ максимальной длине**
- + MSD сортирует только непустые суффиксы**



Как сортировать длинные строки?



- дорогое сравнение если строки отличаются только на конце
- много итераций LSD на длинных ключах



MSD sort vs QuickSort



первый бит == 1 как опорный элемент



Binary QuickSort



- **разделим массив на две части**
 - с ведущим разрядом 0
 - с ведущим разрядом 1ы
- **рекурсивно отсортируем обе части**



Binary quicksort



```
quicksortB(int a[], int l, int r, int w)
{ int i = l, j = r;
  if (r <= l || w > bitsword) return;
  while (j != i)
  {
    while (digit(a[i], w) == 0 && (i < j)) i++;
    while (digit(a[j], w) == 1 && (j > i)) j--;
    exch(a[i], a[j]);
  }
  if (digit(a[r], w) == 0) j++;
  quicksortB(a, l, j-1, w+1);
  quicksortB(a, j, r, w+1);
}
```



3-way split



actinian	coenobite	actinian
jeffrey	conelrad	bracteal
coenobite	actinian	coenobite
conelrad	bracteal	conelrad
secureness	secureness	cumin
cumin	dilatedly	chariness
chariness	inkblot	centesimal
bracteal	jeffrey	cankorous
displease	displease	circumflex
millwright	millwright	millwright
repertoire	repertoire	repertoire
dourness	courness	dourness
centesimal	southeast	southeast
fondler	fondler	fondler
interval	interval	interval
reversionary	reversionary	reversionary
dilatedly	cumin	secureness
inkblot	chariness	dilatedly
southeast	centesimal	inkblot
cankorous	cankorous	jeffrey
circumflex	circumflex	displease



3-way raddix quick sort



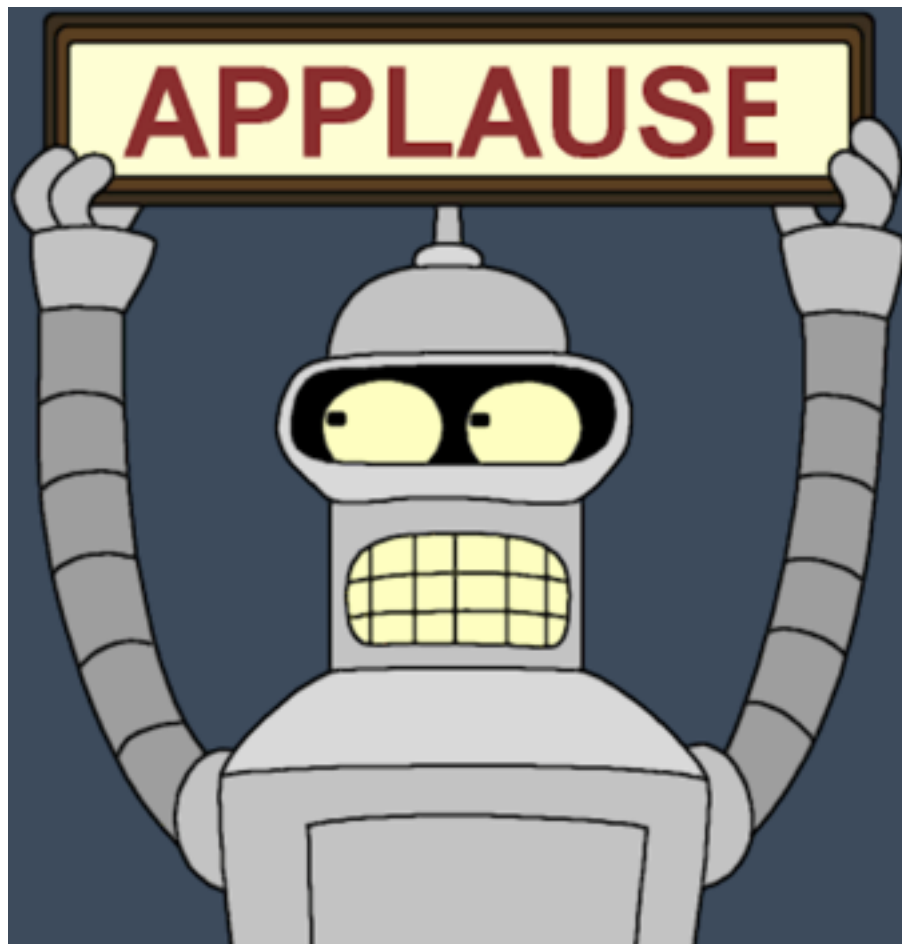
```
#define ch(A) digit(A, D)
void quicksortX(Item a[], int l, int r, int D)
{
    int i, j, k, p, q; int v;
    if (r-l <= M) { insertion(a, l, r); return; }
    v = ch(a[r]); i = l-1; j = r; p = l-1; q = r;
    while (i < j)
    {
        while (ch(a[++i]) < v) ;
        while (v < ch(a[--j])) if (j == l) break;
        if (i > j) break;
        exch(a[i], a[j]);
        if (ch(a[i]) == v) { p++; exch(a[p], a[i]); }
        if (v == ch(a[j])) { q--; exch(a[j], a[q]); }
    }
    if (p == q)
        { if (v != '\0') quicksortX(a, l, r, D+1);
          return; }
    if (ch(a[i]) < v) i++;
    for (k = l; k <= p; k++, j--) exch(a[k], a[j]);
    for (k = r; k >= q; k--, i++) exch(a[k], a[i]);
    quicksortX(a, l, j, D);
    if ((i == r) && (ch(a[i]) == v)) i++;
    if (v != '\0') quicksortX(a, j+1, i-1, D+1);
    quicksortX(a, i, r, D);
}
```



3-way raddix quick sort

now	gig	ace	ago	a	go
for	for	bet	bet	a	ce
tip	dug	dug	and	a	nd
ilk	ilk	cab	ace	<u>b</u>	<u>et</u>
dim	dim	dim	<u>c</u>	<u>ab</u>	
tag	ago	ago	<u>c</u>	<u>aw</u>	
jot	and	and	<u>c</u>	<u>ue</u>	
sob	fee	egg	<u>e</u>	<u>gg</u>	
nob	cue	cue	dug		
sky	caw	caw	dim		
hut	hut	<u>f</u>	<u>ee</u>		
ace	ace	<u>f</u>	<u>or</u>		
bet	bet	<u>f</u>	<u>ew</u>		
men	cab	<u>ilk</u>			
egg	egg	gig			
few	few	hut			
jay	<u>j</u>	<u>ay</u>	<u>ja</u>	<u>m</u>	
owl	<u>j</u>	<u>ot</u>	<u>ja</u>	<u>y</u>	
joy	<u>j</u>	<u>oy</u>	<u>jo</u>	<u>y</u>	
rap	<u>j</u>	<u>am</u>	<u>jo</u>	<u>t</u>	
gig	owl	owl	<u>m</u>	<u>en</u>	
wee	wee	now	owl		
was	was	nob	nob		
cab	men	men	now		
wad	wad	<u>r</u>	<u>ap</u>		
caw	sky	sky	sky	sky	
cue	nob	was	tip	sob	
fee	sob	sob	sob	<u>t</u>	<u>ip</u>
tap	tap	tap	tap	<u>t</u>	<u>ap</u>
ago	tag	tag	tag	<u>t</u>	<u>ag</u>
tar	tar	tar	tar	<u>t</u>	<u>ar</u>
dug	tip	tip	<u>w</u>	<u>as</u>	
and	now	wee	<u>w</u>	<u>ee</u>	
jam	rap	wad	<u>w</u>	<u>ad</u>	







Спасибо за внимание!

Георгий Иванов

Поиск@mail.ru