TAA: folha entregue na aula teórica de 2009/04/17

Algoritmo de ordenação "radix-sort", começando pela ordenação das posições menos significativas.

```
    Dados: m palavras x1, x2,..., xm
         tendo cada palavra n letras, xi = xi[n], xi[n-1],..., xi[1]
    Resultado: a ordenação estável das m palavras

    for i=1 to n:  // começando nos símbolos menos significativos
[*]   ordena estavelmente as palavras x1, x2,..., xm
      usando x1[i], x2[i],..., xm[i], como campo de ordenação
```

Exemplo, vector `[5211, 2213, 2122, 2223]`, $n = 4$, $m = 4$.

```
          (1)           (2)          (3)          (4)
 5 2 2 2      5 2 2 2     5 2 2 2      2 5 2 2 --> 2 2 2 5
 2 2 1 2      2 1 2 2     2 2 1 2 --> 1 2 2 2      1 2 2 2
 1 1 2 2      1 2 1 2 --> 1 1 2 2      2 1 1 2     2 1 2 2
 1 3 2 3  --> 1 2 3 4     1 3 2 3      2 1 3 3     2 3 3 1
```

Teorema O algoritmo está correcto e é estável.
Dem. Por indução em n, número de letras (dígitos) de cada palavra (número).

– Caso base, $n = 1$, trivial, a ordenação global coincide com a ordenação a um só nível ($n = 1$) e esta, por hipótese, está correcta e é estável.

– Passo indutivo.
Queremos provar que: se ordena correcta e estavelmente para palavras de comprimento n, ordena correcta e estavelmente para palavras de comprimento $n + 1$.

Distinguimos a ordenação de um nível (linha `[*]`) e a ordenação global (radix-sort). Usam-se as seguintes hipóteses: Hipótese

(I) A ordenação de nível $n + 1$ é correcta e estável (por construção do algoritmo).

(II) A ordenação global $1 \to n$ (resultante das ordenações dos níveis 1, 2,..., n, por esta ordem) é correcta e estável. (hipótese indutiva).

Sejam 2 valores $x = x_{n+1}x_n \ldots x_2x_1$ e $y = y_{n+1}y_n \ldots y_2y_1$.

1. Se $x_{n+1} < y_{n+1}$, como a última ordenação ($n + 1$) funciona correctamente (propriedade (I)), temos correcção global, pois x fica antes de y.
2. Se $x_{n+1} > y_{n+1}$: análogo.
3. Sejam 2 valores $x = ax_n \ldots x_2x_1$ e $y = ay_n \ldots y_2y_1$.
   (a) Se $x = y$ e x está inicialmente antes de y, também fica no fim x antes de y; usamos (I) e (II).
   (b) Se $x < y$, como o método funciona para o comprimento n (hipótese indutiva, (II)), x fica antes de y, pois a ordenação de nível $n + 1$ é estável por construção do algoritmo (propriedade (I)).

Eficiência
Parâmetros: m – número de palavras a ordenar, n – comprimento de cada palavra, u – tamanho do universo dos caracteres (por exemplo, 10 para inteiros na base 10 e 256 para caracteres no código ASCII).
Hipótese: nas ordenações por níveis, os elementos são colocados em u listas lineares com tempo de inserção e de extracção $O(1)$. Se c é o caracter em questão, o elemento é colocado na lista c.

– Inicialização das u listas: $O(u)$.

– Por cada ordenação "de nível" `[*]`: $O(m)$.

– Total dos m níveis: $O(nm + u)$

– Total: $O(mn + u)$.

Algoritmos para a selecção e a mediana

`select(v,i)`, `mediana(v)` ($i = \lfloor n/2 \rfloor$)
Semelhante ao quick-sort mas só com uma chamada recursiva

```
Função para calcular o elemento de ordem i no vector v
SEL(i,v[a..b])
// na  chamada inicial: a=1, b=n e admite-se que a <= i <= b
   1) Escolhe-se x=v[a]  // outras escolhas para pivot são possíveis!
   2) split(x); Seja k o número de elementos na parte esquerda
   3) se i=k+1: return v[i]
      se i<k+1: SEL(i,      v[a..a+k-1])
      se i>k+1: SEL(i-(k+1),v[a+k+1..b])
```

k	1	$b-a-k$
$a,\ a+1 \ldots\ldots\ldots\ldots\ldots, a+k-1$	$a+k$	$a+k+1,\ a+k+2, \ldots\ldots\ldots\ldots\ldots, b$

Nota.
Análise do algoritmo, pior caso.
Análise do algoritmo, caso médio. As duas partes podem ter, com igual probabilidade, os tamanhos

$$(n-1, 0),\ (n-2, 1),\ \ldots,\ (1, n-2),\ (0, n-1)$$

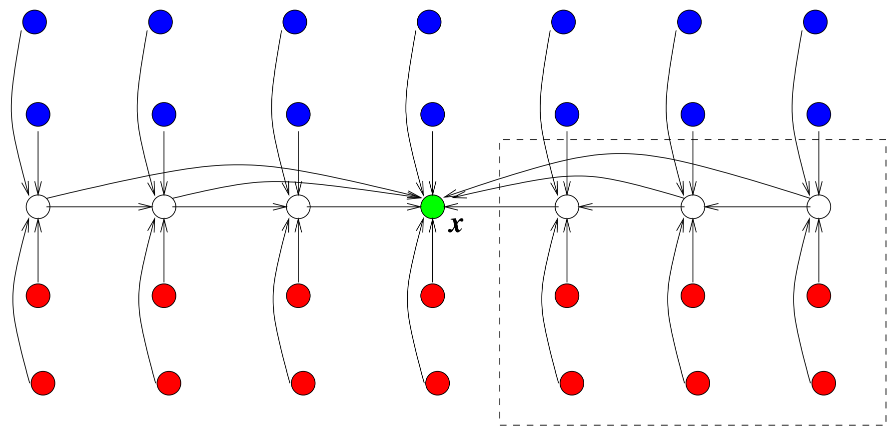
Majorante do caso médio: tomamos o maior dos lados em cada uma das n divisões. Vamos mostrar por indução em n que $E(t(n)) \le 4n$. Por simplicidade vamos supor que n é par.

$$\begin{aligned}
E(t(n)) &\le (n-1) + \frac{2}{n}\sum_{i=n/2}^{n-1} E(t(i)) && (1)\\
&= (n-1) + E[E(t(n/2)) + E(t(n/2+1)) + \ldots + E(t(n-1))] && (2)\\
&\le (n-1) + E(4(n/2) + 4(n/2+1) + \ldots + 4(n-1)) && (3)\\
&\le (n-1) + 4 \times \frac{3n}{4} && (4)\\
&\le 4n && (5)
\end{aligned}$$

É possível $O(1)$ para a mediana, mesmo no pior caso?

```
Algoritmo para seleccionar o elemento de ordem i no vector v
SEL(i,v[a..b])   // na  chamada inicial: a=1, b=n
   1) Dividem-se os n elementos do vector em n/5 grupos de 5
      elementos.
   2) Determina-se a mediana de cada um desses grupos de 5
      elementos.
   3) Chama-se recursivamente SEL para determinar a mediana x das
      n/5 medianas
   4) Faz-se um "split" de v, usando x como pivot
      Seja k o número de elementos no lado esquerdo (<x)
      e n-k no lado direito (>x)
   5) se i=k+1: return v[i]
      se i<k+1: SEL(i,      v[a..a+k-1])
      se i>k+1: SEL(i-(k+1),v[a+k+1..b])
```

A figura seguinte representa o estado do conhecimento sobre a relação de ordem entre os elementos do vector após o "split" (instrução 4); Uma seta de um valor a para um valor b significa que é forcosamente $a < b$.



Análise do algoritmo O número de elementos do vector que excedem x, a mediana das medianas, é pelo menos (no figura: elementos dentro do rectângulo tracejado): número de grupos de 5 elementos à direita de x (há 3 desses grupos na figura) vezes 3 (estamos a desprezar 2 elementos do grupo de 5 a que x pertence), ou seja[1]

$$\frac{n/5-1}{2} \times 3 \quad \in \Omega(n)$$

(cerca de $3n/10$ elementos). No exemplo da figura teríamos o valor $(6/2) * 3 = 9$, o número de elementos vermelhos ou brancos à direita (mas não por baixo) de x. Assim, na chamada recursiva da instrução 5, o tamanho do sub-vector é, no máximo,

$$n - 3 \times \frac{n/5-1}{2} = n - \frac{3n}{10} + \frac{3}{2} = \frac{7n}{10} + \frac{3}{2}$$

A constante $3/2$ pode ser ignorada, incorporando-a na constante c do termo geral da recorrência (6).

Recorrência que permite obter um majorante para o tempo do algoritmo. Seja $t(n,i)$ o tempo, no pior caso, para determinar o elemento de ordem i de um vector com n elementos e seja $t(n) = \max_i\{t(n,i)\}$. Temos as seguintes contribuições para o majorante de $t(n)$:

- Passos 1 e 2: $k_1 n$ (note-se que a mediana de um conjunto de 5 elementos se pode obter em tempo constante.
- Passo 3: $t(n/5)$
- Passo 4: $k_2 n$
- Passo 5: $t(7n/10)$

O termo geral da recorrência é

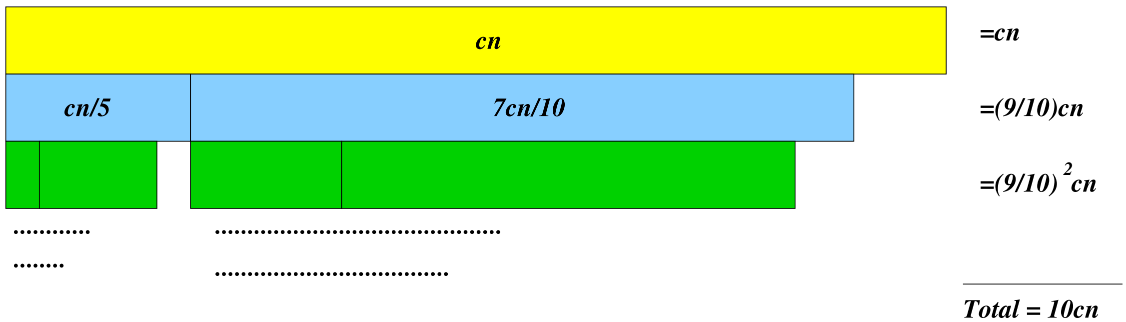
$$t(n) \leq cn + t(n/5) + t(7n/10) \tag{6}$$

onde $c = k_1 + k_2$. Para resolver esta recorrência, ou melhor, para obter um majorante da sua solução, vamos "usá-la" repetidamente

$$\begin{aligned}
t(n) &\leq cn + t(n/5) + t(7n/10) && (7)\\
&\leq cn + c(n/5) + c(7n/10) + [t(n/25) + t(7n/50)] + [t(7n/50) + t(49n/100)] && (8)\\
&= cn(1 + (9/10)) + [t(n/25) + t(7n/50)] + [t(7n/50) + t(49n/100)] && (9)\\
&\ldots \quad \ldots && (10)\\
&\leq cn(1 + (9/10) + (9/10)^2 + (9/10)^3 + \ldots) && (11)\\
&= 10cn && (12)
\end{aligned}$$

Assim, $t(n)$ é de ordem $O(n)$. O raciocínio usado neste desenvolvimento está ilustrado na figura seguinte

[1]Uma análise geral, onde não se supõe que as divisões são exactas, permite obter o minorante $\frac{3n}{10} - 6$ para do número de elementos à direita (ou à esquerda de x). Isto quer dizer que, na chamada recursiva da instrução 5, o tamanho do sub-vector é, no máximo, $n - [(3n/10) - 6] = (7n/10) + 6$.



Resumindo,

Teorema 1 *Para qualquer $i \in \{1, 2, \ldots, n\}$ o algoritmo tem tempo no pior caso de ordem $O(n)$.*

Outro método de resolução de recorrências:

Teorema 2 **(Recorrências com solução $O(n)$)** *Seja a equação geral de uma recorrência*

$$f(n) = f(k_1 n) + f(k_2 n) + \ldots + f(k_p n) + cn$$

onde $c \geq 0$ e k_1, k_2,..., k_p são constantes positivas com $k_1 + k_2 + \ldots + k_k < 1$. Então $f(n)$ é de ordem $O(n)$.