

Estrutura de Dados - MC202 A

1º Semestre de 2018

Tiago de Paula Alves - 187679

Lista 2 - Exercício 1

a) Para um vetor V com n inteiros entre 0 e k , isso é, $q = k + 1$ valores possíveis, o algoritmo mais apropriado seria o **Radix Sort**. Isso é porque esse algoritmo analisa cada elemento como dígitos de base r e ordena o vetor dígito a dígito de maneira estável, começando pelo menos significativo. Com isso, a complexidade da ordenação para cada dígito é no mínimo $r + n$, porque tem que analisar o vetor com n elementos e analisar cada um dos r valores possíveis da base, sendo $\log_r q$ dígitos, o que resulta em uma complexidade final de $O(\log_r q (r + n)) = O\left(\frac{\lg q}{\lg r} (r + n)\right)$.

b) Para que $O(n)$, precisamos escolher a base r com complexidade seja igual ou menor a $O(n)$, para que $O(r + n) = O(n)$, mas que a complexidade de $\log r$ acompanhe a de $\log q$ para que $O\left(\frac{\lg q}{\lg r}\right) = O(1)$, sendo q a quantidade de valores possíveis diferentes em V . A solução mais simples é com q e r constantes, então o algoritmo fica $O\left(\frac{A}{B}(C + n)\right) = O(n)$. Depois temos também $q = n^a$ e $r = n$, que resulta em $O\left(\frac{a \lg n}{\lg n} (n + n)\right) = O(n)$. Além disso, tem as funções logarítmicas $q = (\lg n)^s$ e $r = (\lg n)^t$, que dão $O\left(\frac{s \lg(\lg n)}{t \lg(\lg n)} ((\lg n)^t + n)\right) = O((\lg n)^t + n)$, mas:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n}{(\lg n)^A} &= \lim_{n \rightarrow \infty} \frac{1}{A (\lg n)^{A-1} (1/n \ln 2)} \\ &= \lim_{n \rightarrow \infty} \frac{\ln 2}{A} \frac{n}{(\ln n)^{A-1}} \\ &\quad \dots \\ &= \lim_{n \rightarrow \infty} \frac{(\ln 2)^A}{A!} \frac{n}{\ln n} \\ &= \frac{(\ln 2)^A}{A!} \lim_{n \rightarrow \infty} \frac{1}{1/n} \\ &= \frac{(\ln 2)^A}{A!} \lim_{n \rightarrow \infty} n = +\infty \end{aligned}$$

Portanto, $O(n)$ tem um crescimento assintoticamente maior que $O((\lg n)^t)$ e então $O((\lg n)^t + n) = O(n)$. Esses três grupos de funções são as soluções que resolvem o algoritmo com complexidade linear, sendo que $k = q - 1$.

c) Para $k = n^l - 1$, podemos usar o **Radix Sort** com $r = n$ e, claro, $q = k + 1 = n^l$, então a complexidade fica $O\left(\frac{\lg n^l}{\lg n} (n + n)\right) = O(l \cdot 2n) = O(l \times n)$. Nesse caso, para representar $q = n^l$ valores diferentes com a base $r = n$ precisaremos de $\log_r n^l = \log_n n^l = l$ dígitos. Considerando isso e o operador "/" como a divisão de inteiros, então o algoritmo fica:

Algoritmo RadixSort:

Entradas: Vetor V de tamanho n

Saída: Permutação ordenada S de V

```
-- base (radix)
 $r \leftarrow n$ 

-- cada dígito
para  $D$  de 0 até  $l - 1$ :

    -- conta as repetições do dígito
     $C[r] \leftarrow [0 \dots 0]$ 
    para  $i$  de 0 até  $n - 1$ :
         $d \leftarrow (n / \text{pow}(r, D)) \bmod r$ 
         $C[d] \leftarrow C[d] + 1$ 

    -- acumula o vetor de contagem
    para  $i$  de 1 até  $r$ :
         $C[i] \leftarrow C[i] + C[i - 1]$ 

    -- monta a saída de forma ordenada
    para  $i$  de 0 até  $n - 1$ :
         $d \leftarrow (n / \text{pow}(r, D)) \bmod r$ 
         $S[C[d]] \leftarrow V[i]$ 
         $C[d] \leftarrow C[d] + 1$ 
```
