PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Universidade Federal de Uberlândia – UFU
Faculdade de Computação – FACOM

Método *Divide and Conquer* para o problema de contar duplicatas

Alunos: Laura Ferreira Marquez
Angelo Travizan Neto
Leonard Vieira

Introdução

- Problema

  Dada uma sequência n de números inteiros, desenvolver um algoritmo que conte a quantidade de valores duplicados.

- Solução Proposta
  - Método Divide and Conquer
  - Método Iterativo

Método *Divide and Conquer*

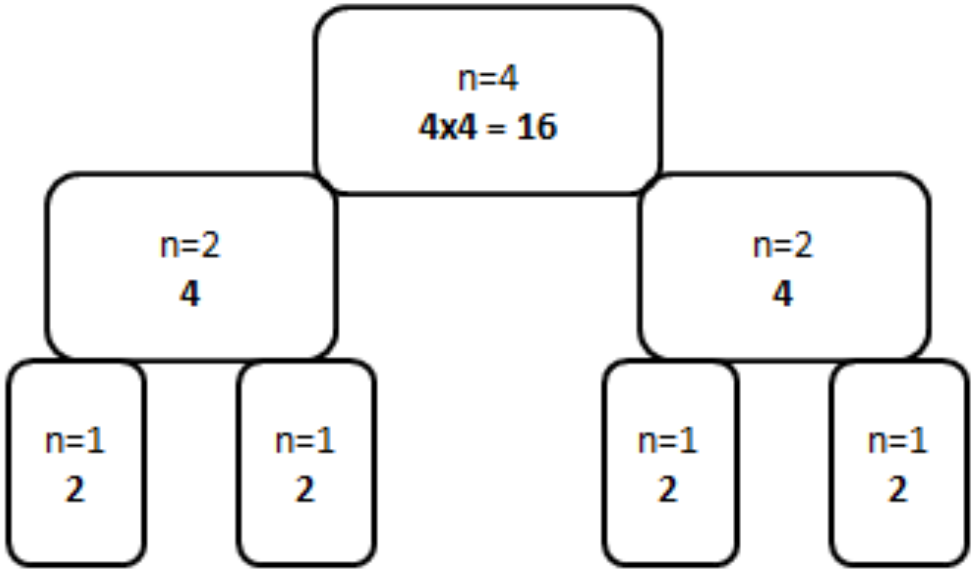


Figura 1: Exemplo da divisão de problemas

Método *Divide and Conquer*

```
1 Conta_duplicatas_Divide_and_Conquer(lista,n):
2   resposta = []
3   lista = mergeSort(lista,n)
4   se n>1:
5       q=1
6       ant = lista[0]
7       i = 1
8       enquanto i < n:
9           se ant == lista[i]:
10              q = q+1
11          senão se q != 1:
12              resposta.add([ant,q])
13          senão: q = 1
14          ant = vetor[i]
15          i++
```

Método *Divide and Conquer*

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$T(n) = 2\left(2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$T(n) = 2\left(2^2T\left(\frac{n}{8}\right) + 2\frac{n}{4} + \frac{n}{2}\right) + n$$

$$T(n) = 2^3T\left(\frac{n}{8}\right) + 2^2\frac{n}{4} + 2\frac{n}{2} + n$$

$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 2^2\frac{n}{2^2} + 2^1\frac{n}{2^1} + 2^0\frac{n}{2^0}$$

$$T(n) = 2^i + \left(\frac{n}{2^i}\right) + 2^{i-1}\frac{n}{2^{i-1}} + 2^{i-2}\frac{n}{2^{i-2}} + \cdots + 2^0\frac{n}{2^0}$$

$$T(1) = 1, \frac{n}{2^i} = 1, T\left(\frac{n}{2^i}\right) = 1$$

$$Então, i = \log n$$

$$Daí, T(n) = 2^{\log_2 n} * 1 + \sum_{j=0}^{\log n - 1} n$$

$$T(n) = n + n\log n$$

$$T(n) = \theta(n\log n)$$

Tempo do algoritmo pelo método *divide and conquer*

$f(n) = O(O(\text{mergeSort}(\text{lista},n)) + n)$

Tempo do algoritmo pelo método *divide and conquer*

$f(n) = O(O(\text{mergeSort}(\text{lista},n)) + n)$

$f(n) = O(n\log n + n)$

Tempo do algoritmo pelo método *divide and conquer*

$f(n) = O(O(\text{mergeSort}(\text{lista},n)) + n)$

$f(n) = O(n\log n + n)$

$f(n) = O(n\log n)$

Método Iterativo

```
1 Conta_duplicata_iterativo(lista,n):
2     resposta = []
3     rep = 1
4     para i de 0 até n-1:
5         atual = lista[i]
6         para j de i+1 até n:
7             se lista[i] == lista[j]:
8                 rep = rep + 1
9         se rep != 1 e naoTaNaLista(resposta,atual):
10            resposta.add([atual,rep])
11        rep = 1
```

Método Iterativo

$$fn = (n-1) + (n-2) + \cdots + \big(n-(n-1)\big) + R$$

$$fn = \sum_{k=1}^{n-1} (n-k) + R$$

$$fn = \sum_{k=1}^{n-1} n - \sum_{k=1}^{n-1} k + R$$

$$fn = n(n-1) - \frac{(1+n-1)n}{2} + R$$

$$fn = n^2 - n - \frac{n^2}{2} = \frac{n^2}{2} - n + R$$

$$f(n) = O(n^2)$$

- Tempo de Execução: $O(n^2)$
- Custo Quadrático!!!

Método Iterativo

$$R = 1(q_1 - 1) + 2(q_2 - 1) + \cdots + r(q_r - 1)$$

$$R = \sum_{B=1}^{r} B(q - 1)$$

$$R = \sum_{B=1}^{r} Bq_r - \sum_{B=1}^{r} B$$

$$R = (1q_1 + 2q_2 + \cdots + rq_r) - 1 - 2 - \cdots - r$$

$$R = (1 + 2 + 3 + \cdots + r)(q_1 + q_2 + \cdots + q_r - 1)$$

$$R = \frac{(1 + r)r}{2}k$$

$$R = \frac{r^2 - r}{2}k$$

$$0 \leq r \leq \frac{n}{2}$$

$$R \leq \frac{\frac{n^2}{2^2} - \frac{n}{2}}{2}k$$

$$R \leq \frac{\frac{n^2}{4} - \frac{n}{2}}{2}k$$

$$R \leq \frac{\frac{n^2 - 2n}{4}}{2}k$$

$$R \leq \frac{n^2 - 2n}{8}k$$

$$R \leq (\frac{n^2}{8} - \frac{n}{4})k$$

$$R \leq \frac{k}{8}n^2$$

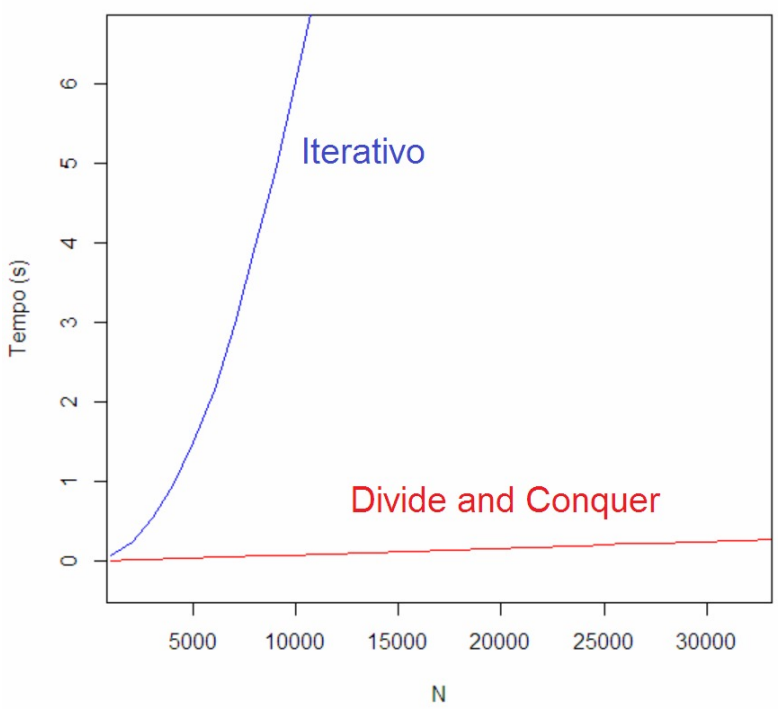
$$\frac{k}{8} > 0$$

$$k > 0$$

$$R = O(n^2)$$

Resultados e comparações

1000	0.06250143051147461
2000	0.24420523643493652
3000	0.5468628406524658
4000	0.976203441619873
5000	1.5141065120697021
6000	2.1813793182373047
7000	2.952444553375244
8000	4.07818603515625
9000	4.91309666633606
10000	6.052159070968628
11000	7.367504596710205
12000	8.738299369812012
13000	10.365773916244507
14000	11.906111717224121
15000	13.709340333938599
16000	15.584218978881836
17000	17.66521120071411
18000	20.103060245513916
19000	21.881428718566895
20000	25.5100359916687
21000	27.03130316734314
22000	29.54704236984253
23000	32.344545125961304
24000	35.294190883636475
25000	40.06362009048462
26000	41.48461580276489
27000	44.54652714729309
28000	48.30391001701355
29000	51.60410165786743
30000	55.59222769737244
31000	59.06983804702759
32000	63.32230353355408
33000	67.54547142982483
34000	70.75657391548157
35000	74.44159817695618
36000	78.52759861946106
37000	82.75893878936768
38000	87.03187823295593
39000	91.6657702922821
40000	96.06881046295166



100000	0.9266571998596191
200000	1.963392972946167
300000	3.1422119140625
400000	4.270026445388794
500000	5.473882436752319
600000	6.694745779037476
700000	7.645276069641113
800000	8.390705347061157
900000	9.625737428665161
1000000	10.756816625595093
1100000	12.015940427780151
1200000	13.203811407089233
1300000	14.43575143814087
1400000	15.493358373641968
1500000	16.907795190811157
1600000	17.75702691078186
1700000	19.154666423797607
1800000	20.503317832946777
1900000	21.578224658966064
2000000	23.09976601600647
2100000	24.364455223083496
2200000	25.852617979049683
2300000	26.83482551574707
2400000	27.671900272369385
2500000	28.968537092208862
2600000	29.974494695663452
2700000	31.552165508270264
2800000	32.77173924446106
2900000	33.859269857406616
3000000	35.227020025253296
3100000	37.2609760761261
3200000	37.353254318237305
3300000	38.686928510665894
3400000	40.55401945114136
3500000	42.127925634384155
3600000	42.38993692398071
3700000	44.04754614830017
3800000	44.965760707855225
3900000	46.129509687423706
4000000	47.64083409309387

Considerações Finais

- A falta de proporcionalidade nos tempos de execução dos dois métodos dificultou a geração de estatísticas

- O algoritmo DC foi mais fácil de implementar, e resultou em tempo de execução menor do que o algoritmo iterativo.

Obrigado