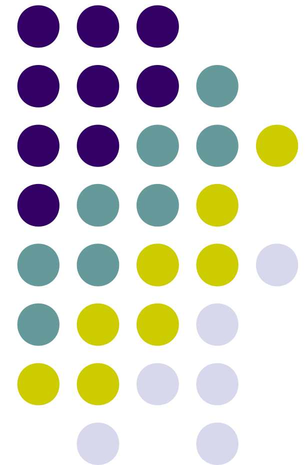
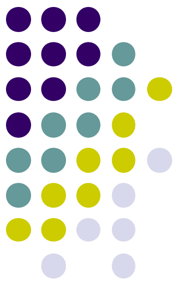
Mergesort

ICC2

Thiago A. S. Pardo

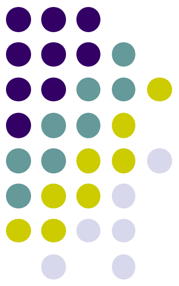


Idéia básica



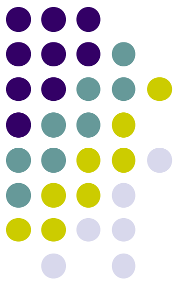
- Dividir para conquistar
  - Um vetor v é dividido em duas partes, recursivamente
  - Cada metade é ordenada e ambas são intercaladas formando o vetor ordenado
  - Usa um vetor auxiliar para intercalar

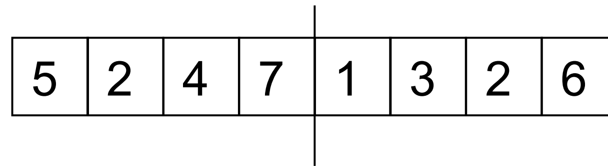
Idéia básica



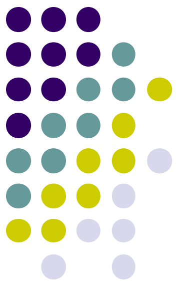
- Algoritmo de ordenação de arranjos por intercalação
  - Passo 1: divide-se um arranjo não ordenado em dois subarranjos
  - Passo 2: se os subarranjos não são unitários, cada subarranjo é submetido ao passo 1 anterior; caso contrário, eles são ordenados por intercalação dos elementos e isso é propagado para os subarranjos anteriores

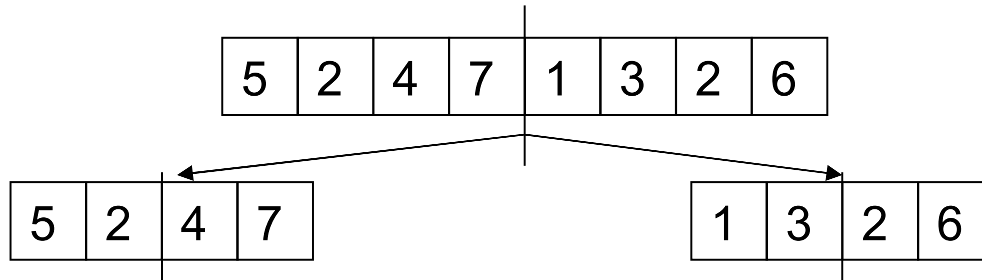
Ordenação por Intercalação



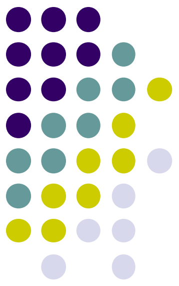


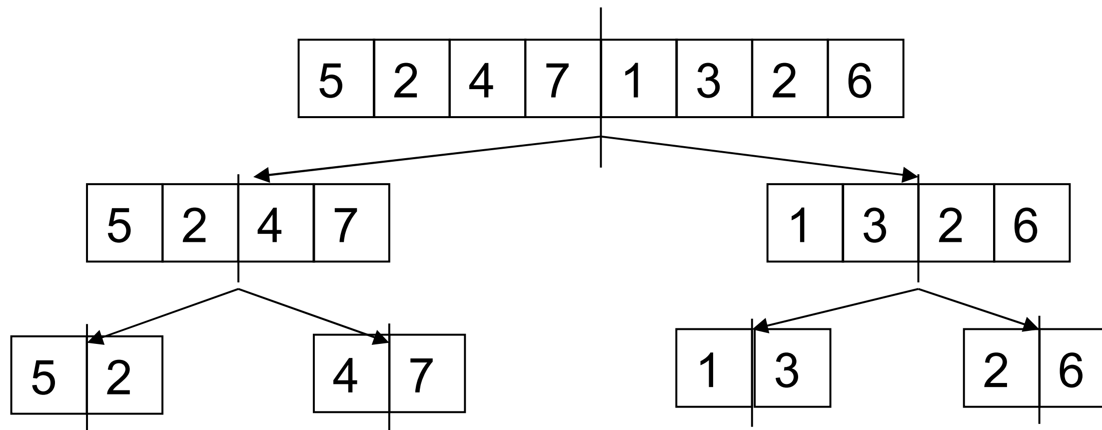
Ordenação por Intercalação

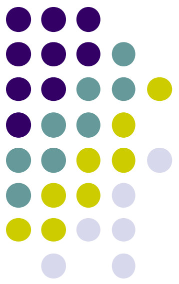




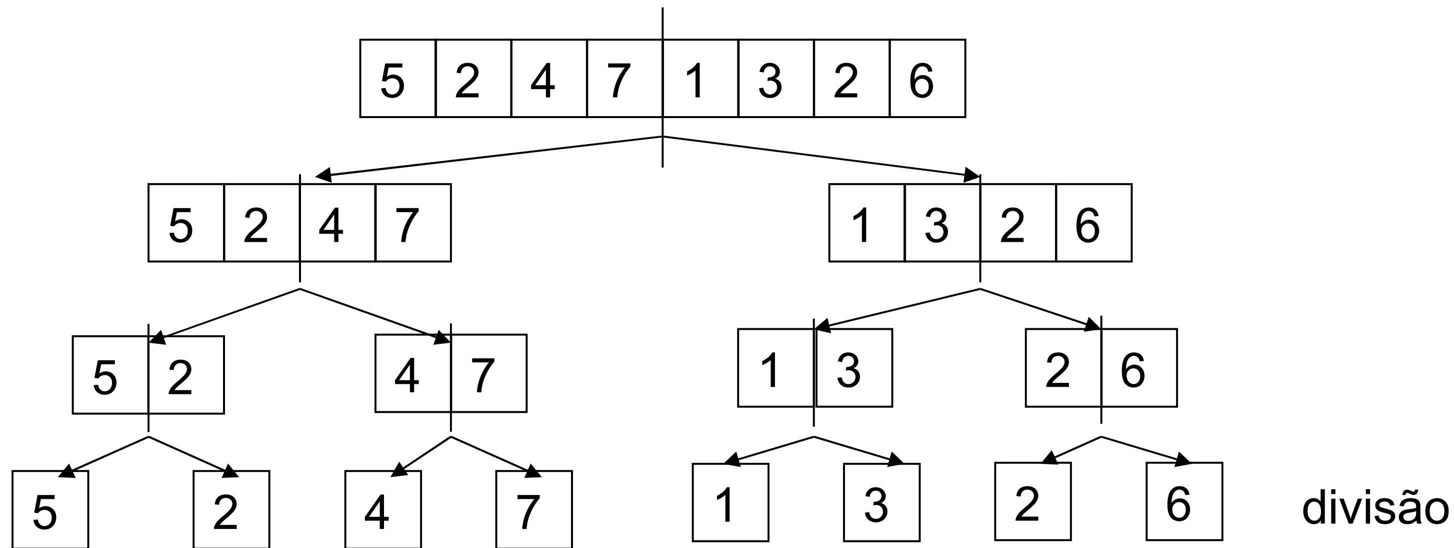
Ordenação por Intercalação



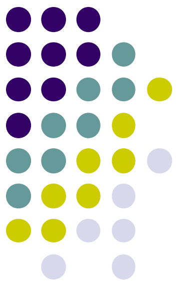




Ordenação por Intercalação



Ordenação por Intercalação



5 2 4 7 1 3 2 6

5 2 4 7 | 1 3 2 6

5 2 | 4 7 | 1 3 | 2 6

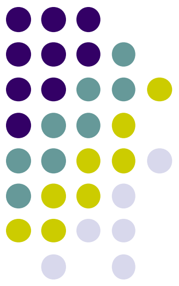
5 | 2 | 4 | 7 | 1 | 3 | 2 | 6

divisão

intercalação

2 5 | 4 7 | 1 3 | 2 6

Ordenação por Intercalação



5	2	4	7	1	3	2	6

5	2	4	7

1	3	2	6

5	2

4	7

1	3

2	6

5 2 4 7 1 3 2 6

divisão

intercalação

2	5

4	7

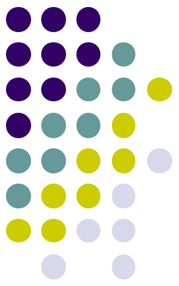
1	3

2	6

2	4	5	7

1	2	3	6

Ordenação por Intercalação



5	2	4	7	1	3	2	6

5 2 4 7   1 3 2 6

5 2   4 7   1 3   2 6

5   2   4   7   1   3   2   6

divisão

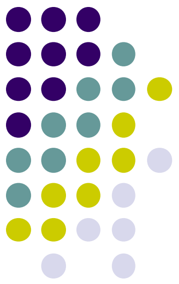
intercalação

2 5   4 7   1 3   2 6

2 4 5 7   1 2 3 6

1	2	2	3	4	5	6	7

Ordenação por intercalação



- Exemplo com arranjo de 4 elementos

Arranjo inicial

3	1	4	2

Divisão em arranjos menores

3	1

4	2

3 1 4 2

Intercalação em ordem

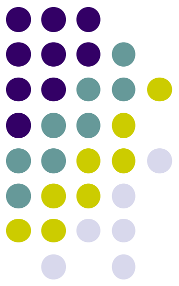
1	3

2	4

1	2	3	4

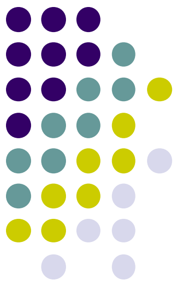
Arranjo final ordenado

Ordenação por intercalação



- Em duplas
  - Implemente a sub-rotina de intercalação de 2 subvetores ordenados

Ordenação por intercalação



- Em duplas

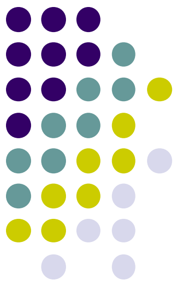
  - Implemente a sub-rotina do mergesort
    - Usando a sub-rotina de intercalação anterior

Implementação

```
void mergesort(int v[], int ini, int fim) {
    int meio;
    if (ini<fim) {
        meio=(ini+fim)/2;
        mergesort(v,ini,meio);
        mergesort(v,meio+1,fim);
        intercala(v,ini,meio,fim);
    }
}
```

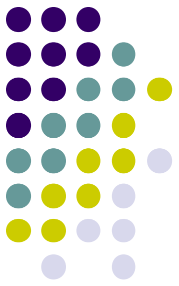
```
void intercala(int v[], int ini, int meio, int fim)
{
    int i, j, k, n1, n2;
    n1=meio-ini+1;
    n2=fim-meio;
    int L[n1+1], R[n2+1];
    for (i=0;i<n1;i++)
        L[i]=v[ini+i];
    L[n1]=9999;
    for (j=0;j<n2;j++)
        R[j]=v[meio+j+1];
    R[n2]=9999;
    i=j=0;
    for (k=ini;k<=fim;k++)
        if (L[i]<=R[j]) {
            v[k]=L[i];
            i++;
        }
        else {
            v[k]=R[j];
            j++;
        }
}
```

Ordenação por intercalação



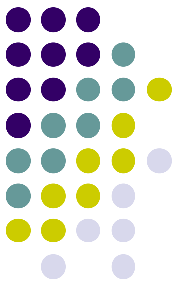
- Faça a análise do algoritmo

Ordenação por intercalação



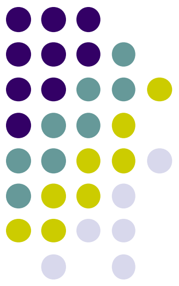
- Rotina principal: mergesort
  - Se n=1 elemento no arranjo, ordenação não é necessária: ?
  - Se n>1
    - O problema é inicialmente dividido em subproblemas: ?
    - Os subproblemas são processados: ?
    - As soluções são combinadas: complexidade da rotina auxiliar de intercalação

Ordenação por intercalação



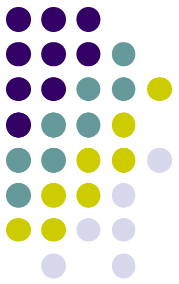
- Rotina principal: mergesort
  - Se n=1 elemento no arranjo, ordenação não é necessária: 1 operação é realizada, tempo constante O(c)
  - Se n>1
    - O problema é inicialmente dividido em subproblemas: 3 operações, tempo constante O(c)
    - Os subproblemas são processados: 2 subproblemas, sendo que cada um tem metade do tamanho original = 2T(n/2)
    - As soluções são combinadas: O(n)

Ordenação por intercalação



- Equações de complexidade do algoritmo
  - ???

Ordenação por intercalação



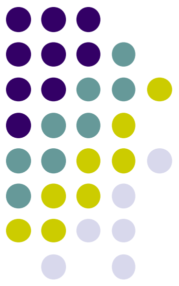
- Equações de complexidade do algoritmo

$T(n)=O(c)=1$, se $n=1$

$T(n)=2T(n/2) + \underbrace{O(c) + O(n)}$, se $n>1$

$O(n)$, já que $c<n$ em geral

Ordenação por intercalação

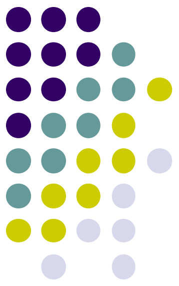


- Equações de complexidade do algoritmo

  $T(n)=1$, se $n=1$

  $T(n)=2T(n/2) + O(n)$, se $n>1$

Ordenação por intercalação

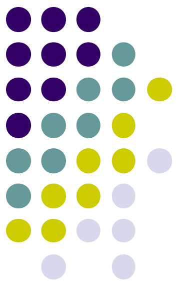


- Equações de complexidade do algoritmo

  $T(n)=1$, se $n=1$

  $T(n)=2T(n/2) + n$, se $n>1$

Ordenação por intercalação



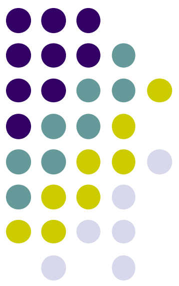
- Equações de complexidade do algoritmo

  $T(n)=1$, se $n=1$

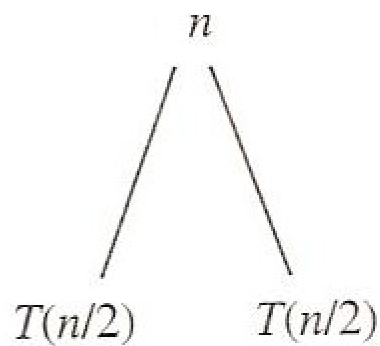
  $T(n)=2T(n/2) + n$, se $n>1$

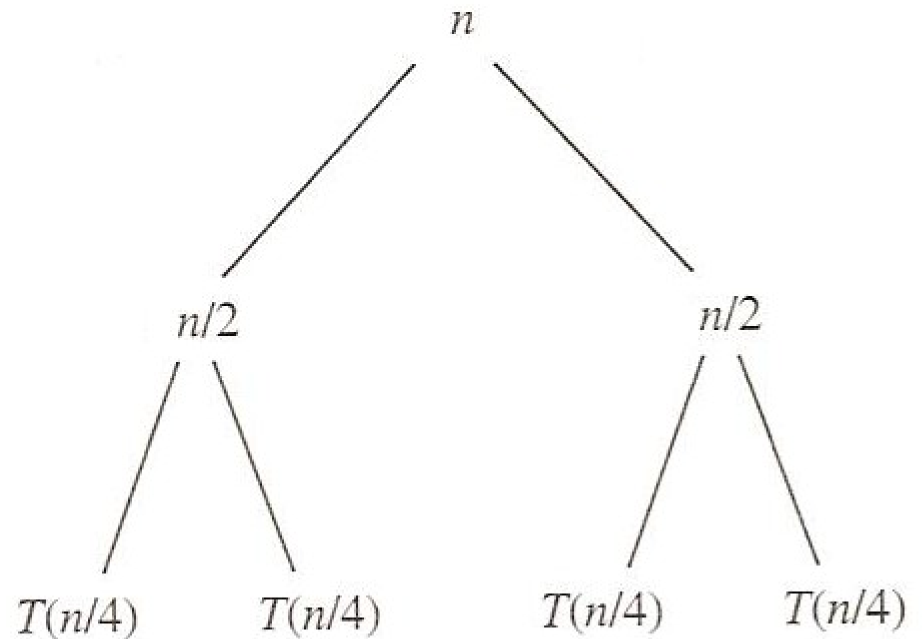
EQUAÇÃO DE RECORRÊNCIA, podendo ser resolvida via árvore de recorrência

Resolução de recorrências

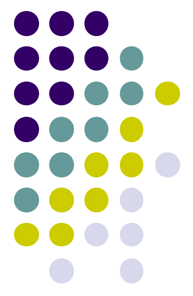


$T(n)$





(a) (b) (c)



n → n

$n/2$ $n/2$ → n

$n/4$ $n/4$ $n/4$ $n/4$ → n

$\vdots$

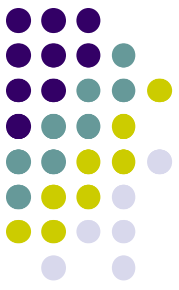
1 1 1 1 1 … 1 1 → n

$\log n + 1$

Total: $n \log n + n$

(d)

Resolução de recorrências



- Tem-se que:
  - Na parte (a), há T(n) ainda não expandido
  - Na parte (b), T(n) foi dividido em árvores equivalentes representando a recorrência com custos divididos (T(n/2) cada uma), sendo n o custo no nível superior da recursão (fora da recursão e, portanto, associado ao nó raiz)
  - ...
  - No fim, nota-se que a altura da árvore corresponde a (log n)+1, o qual multiplica os valores obtidos em cada nível da árvore, os quais, nesse caso, são iguais
    - Como resultado, tem-se n logn + n, ou seja, O(n log n)