

* Linguagem
Java *

/	/	/				
D	E	T	Q	Q	S	S

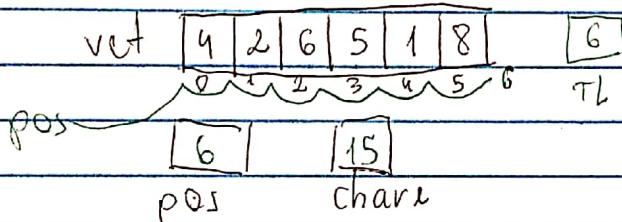
Pesquisa e Ordenação

Métodos de Busca (Searching)

- Buscas em estruturas desordenadas
- Busca Exaustiva (vetor, lista dinâmica, arquivo)
- Busca com Sentinela (vetor, lista dinâmica, arquivo)

Busca Exaustiva

- Critérios de Parada : 1 - Quando encontra a chave procurada
2 - Quando termina o vetor



public int busca_exaustiva (int chave)

```
{ int pos=0;
  while ( pos < TL && chave != vet [pos] )
    pos++;
  if (pos < TL)
    return pos;
  return -1;
}
```

B) Complexidade N

$N \neq 2, +1$

Melhor caso = 3 comparações
↓ ↓ And

B S T Q O S S

Busca Exclusiva com Sentinel

- Critérios de Parada: 1- Quando encontra a chave procurada sentinel

vet	[6 2 6 5 1 8 12]	[6]	[5]
	0 1 2 3 4 5	pos	chave

int i = vet[0] = new int [TL+1]

public int busca_sentinela (int chave)

{ int pos = 0;

vet [TL] = chave; //sentinela

while (chave != vet[pos])

pos++;

if (pos < TL)

return pos;

return -1;

}

Pior caso: 100% comparações + 1 atribuição

Melhor caso: 2 comparações

+ 1 atribuição

Estruturas ordenadas

- Busca Sequencial (vetor, lista dinâmica, arquivo)

- Busca Binária (vetor, arquivo)

Busca Sequencial

- Critérios de parada: 1- Quando encontra a chave procurada
2- Quando termina o vetor
3- Quando encontra o elemento maior que a chave

[] [] [] [] [] []	[6]	[] [5]	chave	pos
0 1 2 3 4 5	TL	pos	chave	6 2

5 | 2



public int busca_sequencial (int chave)

int pos = 0;

while (pos < TL && chave >= vet [pos])

pos++;

if (pos < TL && chave == vet [pos])

return pos;

return pos + TL;

}

→ Pior caso : 2002 Comparações +
2 and'

Busca Binária

→ Tempo de busca : $\log_2 N = 2^k = N$

$$2^k \geq N$$

$$2^{10} = 1024 \geq 1000$$

vet [2 | 4 | 6 | 10 | 12 | 14 | 16 | 18]

↑ 0 1 2 3 4 5 6 7 ↑
int meio fim

public int busca_binaria (int chave)

int ini = 0, fim = TL - 1, meio = fim / 2;

while (ini < fim && chave != vet [meio])

{ if (chave > vet [meio])

início = meio + 1;

else

fim = meio - 1;

meio = (início + fim) / 2;

}



B	S	T	Q	O	S	S
---	---	---	---	---	---	---

→ set

①

```

if(chave == vet[meio])
    return meio;
if(chave >
    return meio + 1;
return meio - 1;
}

```

Lista Duplamente Encadeada

→ Inserção

1. Quando inicio = NULL e fim = NULL

2. Quando já há elementos.

② Em Linguagem C:

```

void inserirNoInicio(Pont ** inicio, Pont ** fim, int info)
{
    Pont * nova = (Pont *) malloc (sizeof (Pont));
    nova->info = info;
    nova->prox = * inicio;
    if (* inicio == NULL)
        * inicio = * fim = nova;
    else
    {
        (* inicio)->ant = nova;
        * inicio = nova;
    }
}

```

③ Em Linguagem Java:

```

public void inserirNoInicio (int info)
{
    No nova = new No (null, inicio, info);
    if (inicio == null)
        inicio = fim = nova;
}

```



①

B	E	T	Q	S	S
---	---	---	---	---	---

①

```
else
{
    inicio.setAnt(nova);
    inicio = nova;
}
```

public void remover (int info)

```
    No aux = busca - exustiva (info);
```

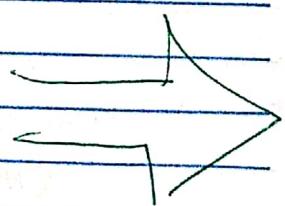
```
    if (aux != null)
```

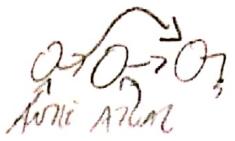
```
    {
        if (inicio == fim)
```

```
            inicializa();
```

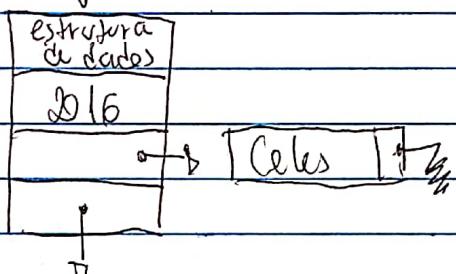
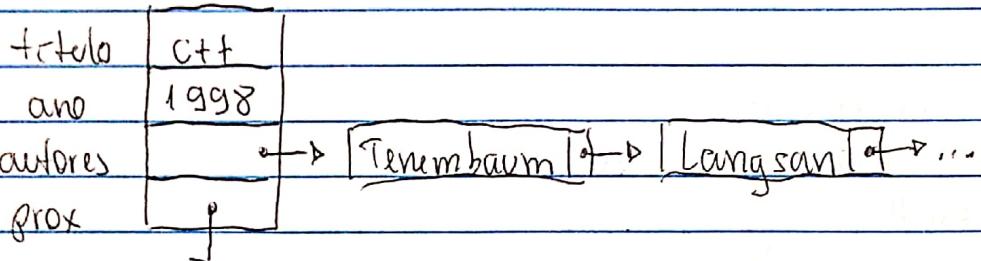
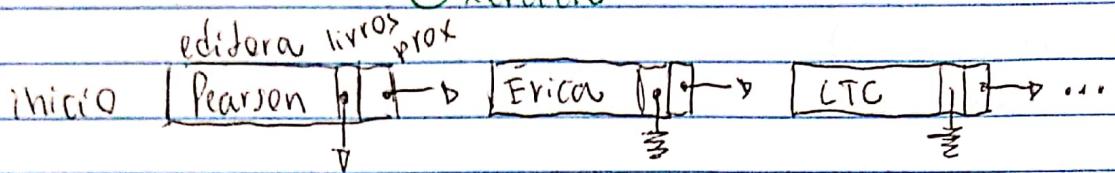
```
        else
```

Delete espaço
para voce
continuar



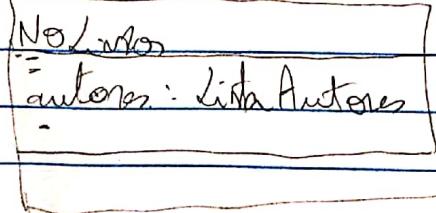
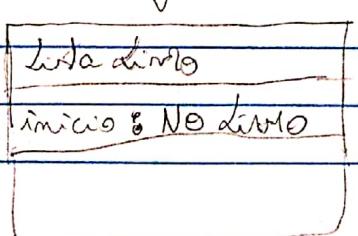
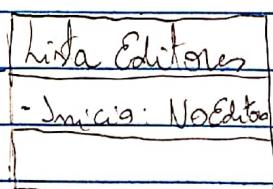
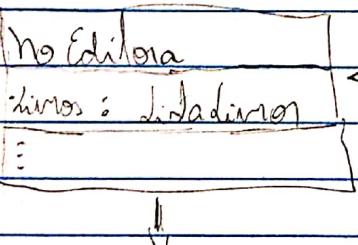


E Exercício



* Faça:

- Diagrama de classes
- Inserir nas listas os campos: editora, título, ano, autores
- Exibir todos os livros de um autor
- Exibir todos os livros de uma editora



método de ordenação

Entende -se a atividade de ordenação como sendo o processo de reanimação de um certo conjunto de objetos de acordo com um critério (ordem) específico. O objetivo da ordenação é facilitar a localização dos membros de um conjunto de dados. Exemplos: lista Telefônica, índices, dicionários, etc...

Os métodos utilizados são, em geral, classificados em 2 categorias:

Classificações interna (vetor / lista dinâmica)

Estruturas de dados armazenadas na memória "interna" dos computadores.

Classificações externas (arquivos)

Estruturas de dados armazenadas em memória "externa"

Algoritmos estudados

Por Inserção: Inserção Direta, Inserção Binária (para busca binária) e Shell.

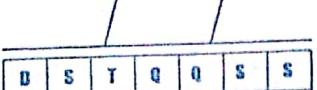
Por Seleção: Seleção Direta e Heap.

Por Troca: Bolha, Shaker, Quick.

Intercalação: Merge Sort ou fusão

Medidas de Análise de eficiência:

- A medida padrão de eficiência é o esforço do algoritmo, ou seja, o número médio de comparações entre os elementos do conjunto.



- Outro critério utilizado é o número de trocas efetuadas

Ordenação por Inserção Direta

Wirth pag. 53 - 55

Azenha pag. 26 - 20

Análise da Eficiência

	Comparações	Movimentações
mínimo	$n - 1$	$3 * (n - 1)$
médio	$(n^2 + n - 2)/4$	$(n^2 + 5n - 10)/4$
máximo	$(n^2 + n - 4)/4$	$(n^2 + 3n - 4)/2$

- Número mínimo ocorre se os elementos estiverem, inicialmente ordenados.

- Número máximo ocorre se os elementos estiverem, inicialmente em ordem reversa.

- Processo estável de ordenação, mantém inalterada a ordem dos elementos com chaves iguais.

Código Inserção Direta

```
public void insercao_direta()
```

```

    int aux, pos;
    for (int i = 1; i < Tl; i++)
    {
        aux = Vet[i];
        pos = i;
        while (pos > 0 && Vet[pos - 1] > aux)
        {
            Vet[pos] = Vet[pos - 1];
            pos--;
        }
        Vet[pos] = aux;
    }
}
```

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

① ①

while ($pos > 0$ & $8 \cdot aux < \text{vet}[pos - 1]$)

{

$\text{vet}[pos] = \text{vet}[pos - 1];$

$pos--;$

{

$\text{vet}[pos] = aux;$

{

{

Codificação mais simples direta em Lista

public void inserir_direto(List)

{

int aux

no pos, i;

i = inicio.getNo();

while (i != null)

{

aux = i.getNo();

pos = i;

while (pos != inicio && aux < pos.getAnt().getInfo().getInfo()),

{

pos.setInfo(pos.getAnt().getInfo());

pos = pos.getAnt();

{

pos.setInfo(aux);

i = i.getNo();

{

8

A ligação do chefe

Ordenação por Inserção Binária

With págs 55-56

Azevedo mjt 10-23

O algoritmo de inserção é "aperfeiçoado" utilizando um método mais rápido para determinar o ponto certo de inserção: busca binária.

Análise de eficiência

	Comparações	Movimentações
mínimo		$3 * (n - 1)$
médio	$n * (\log n - \log e + 0,5)$	$(n^2 + \frac{3}{4}n - 10) / 4$
máximo	$2,4828^n$	$(n^2 + 3n - 4) / 12$

- Movimentações continuam iguais

- Se int[] ordenado, leva mais tempo que a inserção direta

public void insercaoBinaria()

int aux, pos;

for (int i = 1; i < fL; i++)

{ aux = Vet[i]; }

pos = BuscaBinaria(aux, i);

for (int j = i; j > pos; j--)

 Vet[j] = Vet[j - 1];

 Vet[pos] = aux;

}

Método da Seleção direta

With págs 56-58

Azevedo págs 45-50

Esse método é feito no seguinte princípio:

- ~~Desmobrar~~ Percorre-se o menor elemento e troca-o com elemento da posição 0.
- Percorre-se o 2º menor elemento e troca com o elemento da posição 1.
- Prosegue até o final da estrutura.

Análise de Eficiência

O número C de comparações das chaves é independente da ordem inicial das chaves.
 $C = (n^2 - n)/2$

O número M de movimentações é de, no mínimo, $M_{\min} = 3 * (n - 1)$ no caso de as chaves estarem inicialmente ordenadas.

O número M de movimentações média, é $M_{mid} = n * (n(n) + G)$, onde $G = 0,577256\dots$

O número M de movimentações é de, no máximo, $M_{\max} = n^2/4 + 3 * (n - 1)$.

No caso de as chaves estarem em ordem reversa.

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

public void selecao_direta()

i = 0;
int i, j, pos, aux_menor, pos_menor;
While (i < L - 1)
{

j = i + 1;

menor = vet[i];

pos_menor = i;

While (j < L)

{

if (menor > vet[j])

{ menor = vet[j];

pos_menor = j;

}

j++;

vet[pos_menor] = vet[i];

vet[i] = menor;

i++;

}

public void selcas_direta_L ()

{

No i, j, pos;

int menor;

i = inicio;

while (i != prox || i == NULL)

{

pos = i;

menor = i.getInfo();

j = i.getProx();

while (j != NULL)

{

if (menor > j.getInfo())

{ menor = j.getInfo();

pos = j;

}

j = j.getProx();

j.setInfo(i.getInfo());

i.setInfo(menor);

i = i.getProx();

}

II S T M Q S S

Método da Bolha - Bubble Sort

~~Wirth~~ pg 58 - 61

Agenda ng 27-31

O método de Ordenação por permutação direta (bolha) é baseado na comparação e permutação de pares de elementos adjacentes até que todos eles estejam ordenados.

vet	8	3	20	7	2
jet	8	3	2	3	4
	8	3	10	7	2
	0	2	3	3	4
	8	3	20	7	2
	0	1	2	3	4
	8	3	7	20	2
	0	2	2	3	4
	7	9	7	2	2
	0	1	2	3	4

public void volha()

$\text{int } T_{L2} = T_L$, and;

whl (TUR)

[

```

for(i=0; i<TL2-1; i++)
    if(vet[i]>vet[i+5])
        {
            aux = vet[i];
            vet[i] = vet[i+5];
            vet[i+5] = aux;
        }

```

TL2--1

public void bolha - arg()

```
int TL2 = Fila.sig();
Registro regi = new Registro();
Registro regj = new Registro();
while (TL2 > 1)
{
    for (int i = 0; i < TL2 - 1; i++)
    {
        seekArg(i);
        regi.leDoArg(arquivo);
        regj.leDoArg(arquivo);
        if (regi.getCodigo() > regj.getCodigo())
        {
            seekArg(i);
            regi.gravaNoArg(arquivo);
            regj.gravaNoArg(arquivo);
        }
    }
    TL2--;
}
```

Método ShakeSort

Uma "Melhoria" para este método via obter a direção dos sucessivos passos de ordenação. O algoritmo resultante dessa prática se chama ShakeSort ("ordenação por agitação")

DISTOQS S

~~public void shakSort()~~

int inicio = 0, fim = TL - 3, aux;

while (inicio < fim)

{ for (int i = inicio; i < fim; i++)

{ if (vet[i] > vet[i + 1])

{ aux = vet[i];

vet[i] = vet[i + 1];

vet[i + 1] = aux;

fim =

for (int i = fim; i > inicio; i--)

{ if (vet[i] < vet[i - 1])

{ aux = vet[i];

vet[i] = vet[i - 1];

vet[i - 1] = aux;

inicio += 1;

Análise da eficiência

O número de comparações no algoritmo de permutações é $C = (n^2 - n)/2$ e os números mínimo, médio e máximo de movimentações (atribuição de valores aos elementos) são, respectivamente, $M_{\min} = 0$, $M_{\text{med}} = 3 * (n^2 - n)/2$, $M_{\max} = 3 * (n^2 - n)/4$.

Método de Ordenação Heap

Vinh página 63 - 68

Alfredo página 5 - 69

- Construir a árvore heap (filhos são menores que o pai)
- Trocar os ($1^{\text{a}}, 2^{\text{a}}, \dots, TL-1$) maiores elementos que estão nas posições 0 com os ($TL-2, TL-3, \dots, 1$) elementos, respectivamente.

pai \rightarrow posições $n \rightarrow$ filhos ($2n+1$ e $2n+2$)

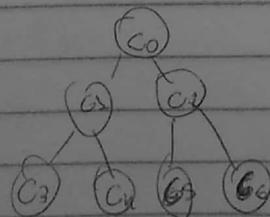
O método de ordenação Heapsort é dividido em duas partes: primeiro monta-se uma árvore binária chamada heap, para em seguida, classificar através de seleções (na árvore).

E' considerado um vetor de chaves C_0, C_1, \dots, C_{n-1} , quando sendo a representação de uma árvore binária, usando a seguinte representação das chaves:

C_i é a raiz da árvore:

$C_{2i+2} = \text{subárvore da esquerda de } C_i \quad \left. \begin{array}{l} \text{para } i = 0, \\ n \text{ div } 2 \end{array} \right\}$

$C_{2i+1} = \text{subárvore da direita de } C_i$



B S T Q Q S S

Representação da árvore no vetor C[0..6].

O algoritmo consiste em inserir os char's dentro do vetor, de tal forma que estes passem a formar uma árvore, na qual todas as raízes das subárvore sejam maiores que aquela ou seja maiores que qualquer uma das suas sucessoras.

(C[i] > C[i+1] e C[i] > C[i+2])

Quando todas as raízes das subárvore satisfizerem essas condições, a árvore forma um heap.

O teste se inicia pela ultima subárvore, cuja raiz está na posição $\lceil \frac{TL}{2} \rceil - 1$ do vetor de char's, ~~prox seguidos~~, a partir daí, passa as subárvore que antecedem esta, até testar a raiz da árvore.

public void Heap()

{
int pai, TL2, FE, FD, maiorF, aux;

TL2 = TL;

while (TL2 > 1)

{
pai = TL2 / 2 - 1;

while (pai >= 0)

{
FE = pai + pai + 2;

FD = FE + 2;

if (FD < TL2)

{
if (FE > FD)

{
maiorF = FE;

else

{
maiorF = FD;

D D D

① ② ③ ④

else

maiorF = FE;

if (vet[maiorF] > vet[par])

aux = Vet[maiorF];

Vet[maiorF] = Vet[par];

Vet[par] = aux;

par -- i

g

TL2 - j

g

aux = Vet[0];

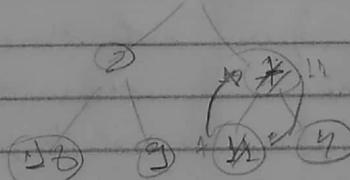
vet[0] = vet[TL2-j];

vet[TL2-j] = aux;

12

vet	5	2	7	13	9	25	4	
	0	1	0	3	4	5	6	

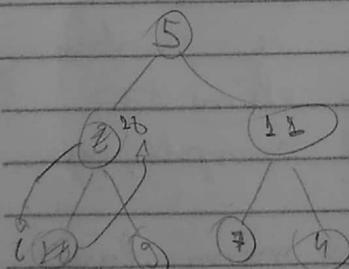
(5)



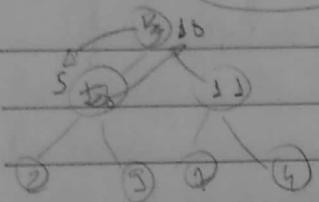
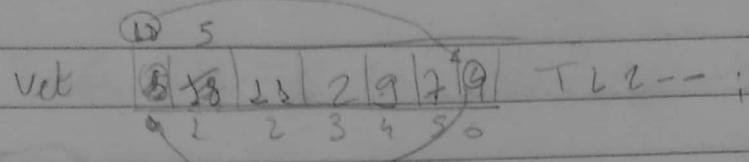
18

vet	5	2	21	13	9	25	4	
	0	1	2	3	4	5	6	

(5)



B S T Q Q S S



método de ordenação por incrementos
Descrescentes - Shell

Wirth - págs. 63-63

Azevedo - págs 22-26

O método ShellSort foi proposto por shell, D.L. como sendo um refinamento do método de ordenação por Inserções direta.

Primeiramente, todos os elementos que estiverem a intervalos de quatro posições entre si na sequência corrente são agrupados e ordenados separadamente. Este processo é denominado de ordenação de distância 4. Após este primeiro passo, os elementos são regroupados em grupos com elementos cujo intervalo é de duas posições, sendo então ordenados novamente. Este processo é denominado de ordenação de distância 2. Finalmente, em um terceiro passo, todos os elementos são ordenados através de uma ordenação simples, ou ordenação de distância 1.

O método possui a vantagem que a cada passo, faz classificações parciais da estrutura, o que favorece o desempenho dos passos seguintes.

B S T Q Q S S

vct	3	4	18	10	2	20	6	14	8	13	27	3	7
	0	1	2	3	4	5	6	7	8	9	10	11	12

public void Shell()

{

 int i, j, k, dist = 4, aux;

 while (dist > 0)

{

 for (i = 0; i < dist; i++)

 for (j = i; j + dist < 12; j += dist)

 if (vct[j] > vct[j + dist])

 aux = vct[j];

 vct[j] = vct[j + dist];

 vct[j + dist] = aux;

 }

 for (K = j; K - dist >= i && vct[K] < vct[K - dist]; K -= dist)

{

 aux = vct[K - dist];

 vct[K - dist] = vct[K];

 vct[K] = aux;

}

 dist = dist / 2;

}

G

Método de ordenação Quick Sort

Wirth pág. 68 - 73

Azevedo pág. 34-44

Criado por C.A.R. Hoare, o algoritmo Quick Sort ("ordenação rápida") é baseado no fato de que as permutações devem ser preferencialmente empregadas para pares de elementos que guardem entre si distâncias grandes, com a finalidade de se conseguir uma eficiência maior. É possível ordená-los com apenas $n/2$ permutações, tornando-se primeiramente os elementos das extremidades à direita e à esquerda convergindo gradualmente para o centro, pelos dois lados.

Quick com Pivô: O vetor é varrido da esquerda para a direita até que seja encontrado um elemento maior que o pivô; sendo então varrido da direita para a esquerda até que seja encontrado um elemento menor que o pivô.

Nesta ocasião os elementos não permudados, e este processo de varredura e permutações continua até que os 2 deslocamentos se encontrem, quando deve-se repetir o processo para os partes restantes.

Quick com pivô: para $i=0$ e $j=TL-1$, começando aumentando o i ; quando o elemento da posição i for maior que o elemento da posição j , permute-se os elementos e passar-se a dimensão $i \leftrightarrow j$ até que o elemento de j seja menor que o de i , assim sucessivamente até que $i \leftrightarrow j$ se encontrem. Separamos os partes e continuarmos até que todos os partes tivessem um só elemento.

B S T Q Q S S

public void QuickSP()

QuickSP(0, fin);

}

public void QuickSP(int ini, int fin)

{ int i = ini, j = fin, aux;

while (i < j)

{

 while (i < j && vet[i] <= vet[j])

 i++;

 aux = vet[i];

 vet[i] = vet[j];

 vet[j] = aux;

 while (i < j && vet[i] >= vet[j])

 j--;

 aux = vet[i];

 vet[i] = vet[j];

 vet[j] = aux;

 if (ini < i - 1)

 QuickSP(ini, i - 1);

 if (j + 1 < fin)

 QuickSP(j + 1, fin);

}

U	S	T	Q	Q	S	S
---	---	---	---	---	---	---

public void QuickSort (int ini, int fim)

{ int i = ini, j = fim, aux;

boolean flag = true;

while (i < j)

{ if (flag)

{ while (i < j && Vet[i] <= Vet[j])

{ i++;

else

while (i < j && Vet[j] >= Vet[i])

j--;

aux = Vet[i];

Vet[i] = Vet[j];

Vet[j] = aux;

flag = ! flags

if (ini < i - 1)

QuickSP(ini, i - 1);

if (j + 1 < fim)

QuickSP(j + 1, fim);

F

public void QuickCP();

QuickCP(0, t-1);

public void QuickCP(int ini, int fim)

int i = ini, j = fim, aux; pivo = Vet[(ini + fim)/2];
while (i < j)

{

 while (Vet[i] < pivo)

 i++;

 while (Vet[j] > pivo)

 j--;

 if (i <= j)

 { // Troca

 i++;

 j--;

}

 if (ini < j)

 QuickCP(ini, j);

 if (i < fim)

 QuickCP(i, fim);

}

D S T Q Q S S

Método de Ordenação - Intercalação Simples

Merge Sort ou fusão Direta

With page 76-82

Azenado maig 86 - 96

A classificação por intercalação consiste em dividir o roteiro em 2 ou mais segmentos, formando novos segmentos ordenados, os quais serão intercalados entre si.

Prémissa Implementação

vet [23 | 27 | 8 | 25 | 9 | 12 | 19 | ?]
 0 1 2 3 4 5 6 7

	vet 2	8	9	28	23	
Participación		0	1	2	3	
	vet 2	7	21	35	27	

public void Merge()

```

    int seq = 1;
    int vet1[] = New int [TL / 2];
    int vet2[] = New int [TL / 2];
    while (seq < TL)
        {
            particao (vet1, vet2);
            fusao (vet1, vet2, seq);
            seq = seq * 2;
        }
    
```

public void particao (int vet1[], int vet2[])

```

    int meio = TL / 2;
```

```

    for (int i = 0; i < meio; i++)
```

```

    {
```

```

        vet1[i] = vet[i];
```

```

        vet2[i] = vet[i + meio];
```

```

    }
```

```

}
```

public void fusao (int vet1[], int vet2[], int seq)

```

    int i = 0, j = 0, k = 0, aux_seq = seq;
```

```

    while (k < TL)
```

```

    {
```

```

        while (i < seq && j < seq)
```

```

        {
```

```

            if (vet1[i] < vet2[j])
```

```

                vet[k++] = vet1[i++];
```

```

            else
```

```

                vet[k++] = vet2[j++];
```

```

            }
```

① ①

D	S	T	Q	O	S	S
---	---	---	---	---	---	---

while ($i < \text{seq}$)

$\text{vet}[K+i] = \text{vet}[2 \cdot i + 1];$

while ($j < \text{seq}$)

$\text{vet}[K+j] = \text{vet}[2 \cdot j + 1];$

$\text{seq} = \text{seq} + \text{aux} - \text{seq};$

?

?

Segunda Implementação

aux

0 1 2 3 4 5

vet	23	17	8	25	9	12
	0	1	2	3	4	5

public void MergeSort ()

{ int aux[] = new int [TL];
merge (aux, 0, TL-1);

}

public void Merge (int aux[], int esq, int dir)

{ int meio;

if ($esq < dir$)

{

meio = ($esq + dir$) / 2;

merge (aux, esq, meio);

Merge (aux, meio+1, dir);

dividi (aux, esq, meio, meio+1, dir);

}

?

public void fusao(int aux[3], int lim1, int lim2,
 int ini1, int fin1)

{

int i = ini1, j = ini2, k = 0;

while (i <= lim1 && b. j <= lim2)

if (vet[i] <= vet[j])

aux[k++] = vet[i++];

else

aux[k++] = vet[j++];

while (i <= lim1)

aux[k++] = vet[i++];

while (j <= lim2)

aux[k++] = vet[j++];

for (i = 0; i < k; i++)

vet[i] = aux[i];

}

+1

B S T Q Q S S

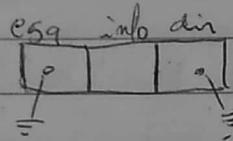
Árvore N-ária

- N é o numero de ligações

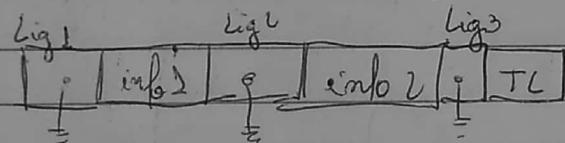
- Cada nóde possui N-1 informações

Exemplos:

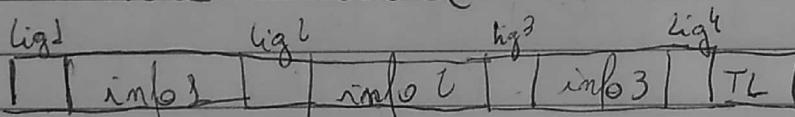
- Árvore Binária (2-ária)



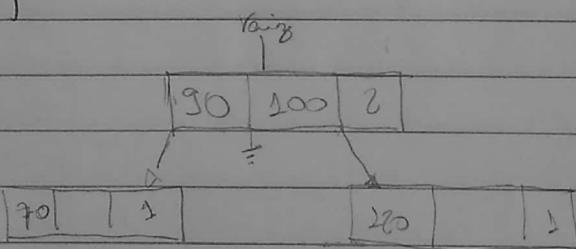
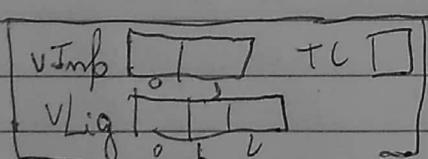
- Árvore Terária (3-ária)



- Árvore 4-aria



- Árvore N-ária ($N=3$)



public void inserir (int info)

{ NO aux, ant
int pos;

if (raiz == null)

{ raiz = new NO(info);

else

{

aux = raiz;

while (aux != null && !flag)

{ pos = aux.buscsequencial(info);
, if (aux.getTl() < N - 1)

{ aux.remover(pos);

aux.setV3info(pos, info);

aux.setTl(aux.getTl() + 1);

flag = true;

else

{ aux = aux.getVl();

aux.setVl(aux.getVl() + 1);

if (!flag)

ant.setVliz(pos, new NO(info));

}

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

In-Order N-Ara

- lig 0
- exile 0
- lig 1
- Exile 1
- lig 2 (T)

public void InOrder (NO raiz)

{ int i;

if (raiz != null)

for (i=0; i < raiz.getTL(); i++)

InOrder (raiz.getLig(i));

System.out.println (raiz.getVimbo(i));

InOrder (raiz.getRig(raiz.getTL()));

}

Exclusão da Árvore B

public void exclusao(int info)

{ int no, SubE, subD, infoE, infoD, pai

NO no

no = buscarInfo(info);

info = no.procurarFisicas(info);

if (no.getVig(0) != null)

{

SubE = buscarSubE(no, nos);

SubD = buscarSubD(no, nos+1);

if (SubE.getTL() >= SubD.getTL())

{

no.setVNlp(no, subE.getVLnp(SubE.getTL()-1));

no.setVPes(no, subE.getVPes(SubE.getTL()-2));

SubE.setTL(SubE.getTL()-1);

folha = subE;

}

else

{

no.setVSmpl(no, subD.getVLnp(0));

no.setVPes(no, subD.getVPes(1));

SubD.remanejarEx(0);

subD.setTL(SubD.getTL()-1);

folha = subD;

}

else

{

folha = no;

folha.remanejarEx(no);

folha.setTL(folha.getTL()-1);

}

B S T Q Q S S

T L D # / H / H

1

if (folha • get TL() < N)

 pai = localizarPai (folha, info);

 posP = pai • procurarPosicao (info);

 if (posP > 0)

 irmaE = pai.getVLig (posP - 1);

 if (posP < ypai • get TL (1))

 irmaD = pai • get VLig (posP + 1);

 if (irmaE != null && irmaD != null && get TL () > N)

 {

 folha • remanejar (0);

 folha • setVInfo (0, pai • get VInfo (posP - 1));

 folha • set VPos (0, pai • get VPos (posP - 1));

 folha • set TL (folha • get TL () + 2);

 pai • set VInfo (posP - 1, irmaE • get VInfo irmaE);

 get TL () - 1);

 pai • set VPos (posP - 1, irmaE • get VPos (irmaE • get TL () - 1));

 irmaE • set TL (irmaE • get TL () - 1);

 }

 if (irmaD != null && irmaD • get TL () > N)

 {

 folha • set VInfo (folha • get TL (), pai • get VInfo (posP));

 folha • set VPos (folha • get TL (), pai • get VPos (posP));

 folha • set TL (folha • get TL () + 2);

 pai • set VInfo (posP, irmaD • get VInfo (0));

 pai • set VPos (posP, irmaD • get VPos (0));

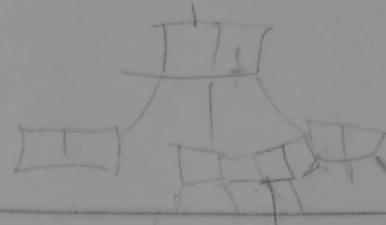
 irmaD • remanejar (& (0));

 irmaD • set TL (irmaD • get TL () - 1);

3

/	/					
B	S	T	Q	O	S	S

① ②



else
{

 2 funao(folha, pai, imatE, imad, pos);

}

}

public void ^{for}(no folha, no pai, no Imat, no Imad),
{ int Pos; }

 if(ImatE != NULL)

 {

 ImatE.setVInfo(ImatE.getTL(), Pai.getVInfo(Pos-1));

 ImatE.setVPos(ImatE.getTL(), Pai.getVPos(Pos-1));

 ImatE.setTL(ImatE.getTL() + 1);

 ImatE.setVInfo(ImatE.getTL(), folha.getVInfo(0));

 ImatE.setVPos(ImatE.getTL(), folha.getVPos(0));

 ImatE.setTL(ImatE.getTL() + 1);

 Pai.setRemanejavel(Pos-1);

 Pai.setTL(Pai.getTL() - 1);

 } ImatE.setVLig(imatE.getTL() - 1, folha.setVLig(0));

 ImatE.setVLig(imatE.getTL(), folha.setVLig(1));

 folha = ImatE;

 }

 {

 folha.setVInfo(folha.getTL(), pai.getVInfo(Pos));

 folha.setVPos(folha.getTL(), pai.getVPos(Pos));

 folha.setTL(folha.getTL() + 1);

 for(int i = 0; i < imad.getTL(); i++)

 folha.setVInfo(folha.getTL(), Imad.getVInfo(i));

 folha.setVPos(folha.getTL(), Imad.getVPos(i));

 folha.setVLig(folha.getTL(), Imad.getVLig(i));

 ① ③

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

(2) (b)

folha.setTL(folha.getTL() + 1);

folha.setVLig(folha.getTL(), irmad).getVLig(i);

pai.remanejarEx(Pes);

pai.setTL(pai.getTL() - 1);

5

if (pai.getTL() < N)

if (raiz == pai && pai.getTL() == 0)
raiz = folha

else

folha = pai; irmae = NULL; irmad = NULL;

pai.coligarPai(folha, folha.getVLig(0));

pes = pai.procuaPosicao(folha.getVLig(0));

if (pes > 0)

irmae = pai.getVLig(pes - 1);

if (pes < pai.getTL())

irmaad = pai.getVLig(pes);

pesao(folha, pai, irmae, irmad, pes);

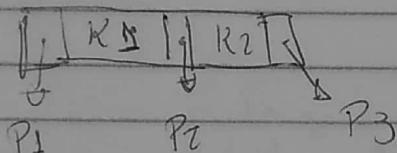
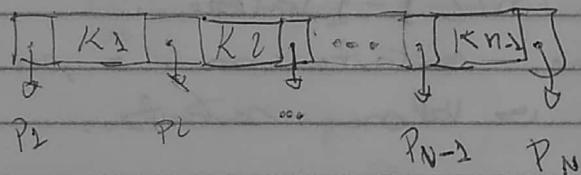
8

9

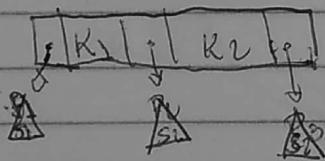
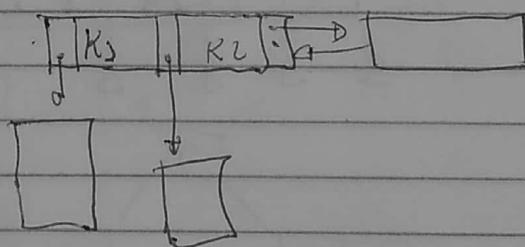
2
4

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

Arvore B+ (B+ tree)

 $N =$ Número de ligações $N-s =$ Número de informaçõescaso para $N = 3$  $\lfloor \cdot \rfloor =$ Truncamento $\lceil \cdot \rceil =$ arredondamentocaso geral para N 

Nodo não folha

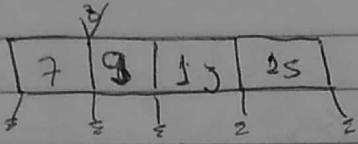
Nodo ~~não~~ folha

Nodo folha

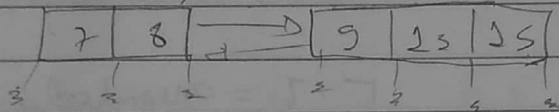
- O primeiro nodo contém $\lceil (n-s)/2 \rceil$ valores.
- O segundo nodo contém os valores restantes,
- Copia o menor valor do segundo nodo para o pai

B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

Exemplo $n = 5$



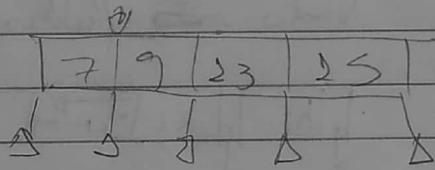
{9}



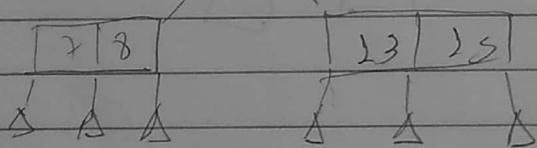
modo não folha

- O primeiro modo contém $\lceil n/2 \rceil - 1$ valores.
- Move o menor dos valores restantes para o pai.
- O segundo modo contém os valores restantes.

Exemplo $n = 5$



{9}



B	S	T	Q	Q	S	S
---	---	---	---	---	---	---

Exemplo B + Tree $n=4 \rightarrow 3$ interações
 $(1, 4, 7, 10, 27, 22, 31, 25, 39, 20, 28, 42)$

No folha

$$\Gamma_{(n-2)/2} = 2$$

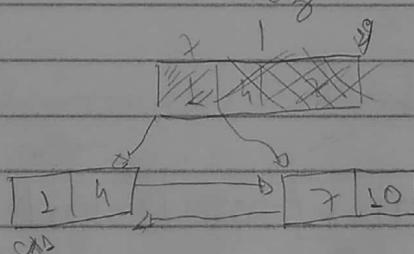
$$\Gamma_{(4-2)/2} = 2$$

não folha

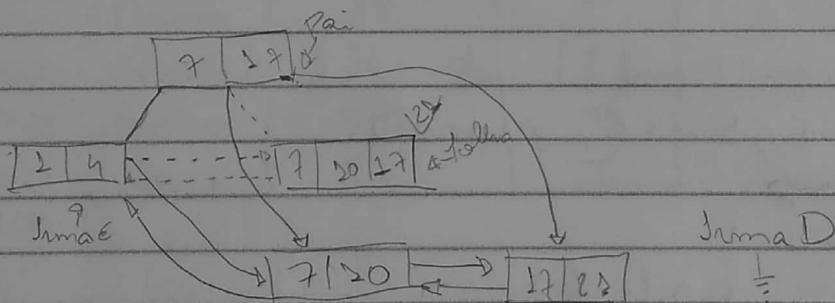
$$\Gamma_{n/2} = 2$$

$$\Gamma_{4/2} = 2 = 2$$

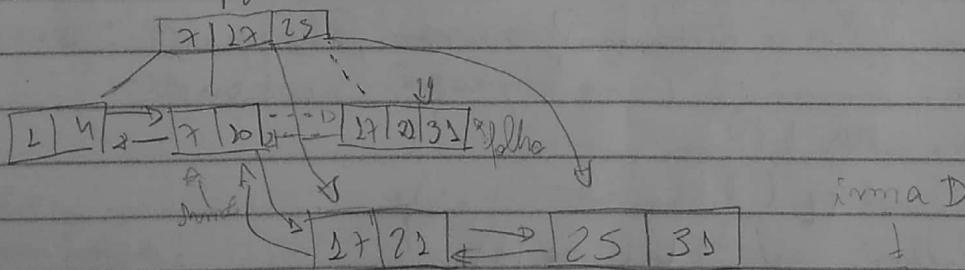
raiz



raiz



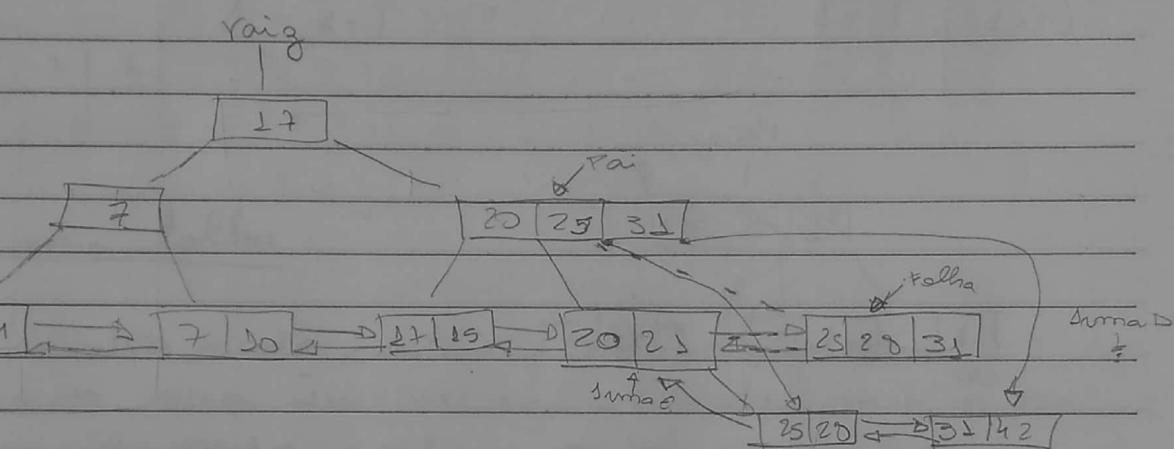
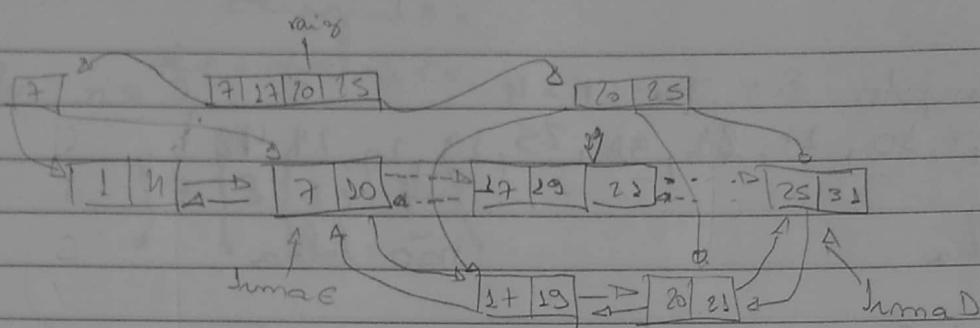
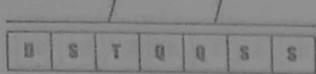
raiz & pai



irmão D

\downarrow

\uparrow



subir void inserirBinario (int info)

{ No folha, pai, IrmãoE, IrmãoD

int pos, posP

; } (raiz = -NULL)

raiz = New No (info);

folha = navegarArvorefolha (info);

pos = folha - procurarPosicas (info);

folha - remaneja (pos);

folha - setInfo (pos, info);

folha - setTL (folha - getTL () + 1);

if (folha - getTL () == N)

{ pai = localizaPai (folha, info);

prox = pai - procurarPosicas (info);

if (prox > 0)

irmaoE = pai - getVlrig (prox - 1);

0 0 0

0 0 0

f (pai, filho, pais, getTc())

irmao = pais . getVlrig (norp + 1),

split (folha, pais, , irmaoE, irmaoD);

3

3

3

Organizações de Arquivos

Organização Sequencial

Arquivo de Dados

	numero	nome	idade	status
0	1000	André	25	T
1	1050	Alvaro	34	T
2	1100	Antonio	22	T
3	1400	Pedro	34	T
4	1500	Mario	56	T

<EOF>

- Arquivo de dados ordenado pelo campo chave

- Acesso ; - Busca Binária

- Varredura Sequencial

- Inserção

- Exclusão Lógica

- Exclusão Física

- Edição

Organização Indexada

Arquivo de Índice

	chave	valor
0	3000	2
1	2050	4
2	1100	0
3	1400	2
4	2500	3

<EOF>

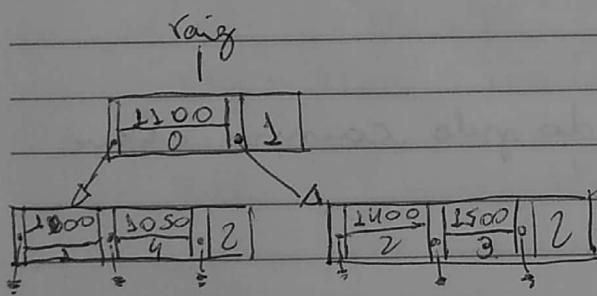
Arquivo de Dados

	numero	nome	idade	status
0	1200	Antonio	22	T
1	1000	Onofre	25	T
2	2400	Pedro	34	T
3	1500	Elasio	56	T
4	2050	Alonso	34	T

- Arquivo de índice ordenado pelo campo chave
- Insere sempre no final do arquivo de dados.
- = Caso: - Busca Binária no arquivo de índice.
- Variedade no arquivo de índice.

Organização Sequencial - Indexada

Árvore B



	numero	nome	idade	status
0	1200	Antonio	22	T
1	1000	Onofre	25	T
2	2400	Pedro	34	T
3	1500	Elasio	56	T
4	2050	Alonso	34	T

- Árvore B como índice para o arquivo de dados.
- Insere sempre no final do arquivo de dados
- = Caso: - Busca na B Tree
- In Ordem na árvore

Sequential

public void inserir(Registro reg)

Registro regAux = New Registro();

int pos = busca_binaria(regAux.getNumero());

if (pos >= fileSize())

{

pos = pos - fileSize();

for (int i = fileSize() - 1; i >= pos; i--)

{ seekAng(i);

regAux.setAng(angulo);

regAux.gravaNoAng(angulo);

}

seekAng(pos);

regAux.gravaNoAng(angulo);

{

else

System.out.println("numero ja existente!");

{

public void Reorganiza().

Registro reg = New registro();

int it = 0;

for (int i = 0; i < fileSize(); i++)

{ seekAng(i);

regAux.setAng(angulo);

if (regAux.get_Status() == 1)

{ if (it != i)

{ seekAng(it);

regAux.gravaNoAng(angulo);

it++

B S T Q Q S S

① truncate(it);
②