Métodos de Ordenação

Conceitos básicos sobre ordenação

- Ordenar corresponde ao processo de rearranjar um conjunto de objetos em uma ordem específica.
- **Objetivo** da ordenação:
  - **facilitar** a recuperação posterior de elementos do conjunto ordenado.
  - Os algoritmos trabalham sobre os registros de um arquivo.

Conceitos básicos sobre ordenação

- **Classificação dos métodos** de ordenação:
  - **Ordenação interna**: arquivo a ser ordenado cabe todo na memória principal.
  - **Ordenação externa**: arquivo a ser ordenado não cabe na memória principal.

- Diferenças entre os métodos:
  - Em um método de ordenação interna,
    - qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa,
    - os registros são acessados sequencialmente ou em grandes blocos.

Conceitos básicos sobre ordenação

- **Notação**
  - Sejam os itens: a1; a2; . . . ; an.
  - Ordenar consiste em permutar estes itens em uma ordem ak1;ak2; . . . ; akn tal que, dada uma função de ordenação **f**, tem-se a seguinte relação:
  - f(ak1) < f(ak2) < : : : < f(akn)
  - A função de ordenação é definida sobre o campo **chave**.
  - A relação < deve satisfazer as condições:
    - Apenas uma das três condições é verdadeira: a < b, a = b, a > b.
    - Se a < b e b < c então a < c.

Conceitos básicos sobre ordenação

- Qualquer tipo de chave sobre o qual exista uma regra de ordenação bem-definida pode ser utilizada.
- Um método de ordenação é **estável** se a ordem relativa dos itens com chaves iguais não se altera durante a ordenação.
- Na escolha de um algoritmo de ordenação interna **deve** ser considerado o tempo gasto pela ordenação.
- Sendo **n** o número registros no arquivo, as medidas de **complexidade** relevantes são:
  - Número de **comparações C(n)** entre chaves.
  - Número de **movimentações M(n)** de itens do arquivo.

Ordenação direta e indireta

- Ordenação **direta** ou por registros:
  - Dado um vetor para ordenação, percorre-se cada elemento inserindo-o na posição ordenada.
- Ordenação **indireta** ou por endereço:
  - Feita numa tabela de ponteiros, não há troca de registros, apenas de apontadores.
- Ordenação direta → Exemplo:

Vetor antes da ordenação

0	1	2	3
11	20	6	12

Vetor após a ordenação

0	1	2	3
6	11	12	20

Ordenação direta e indireta

- Ordenação indireta → Exemplo:

Vetor antes da ordenação

1	200	11
2	202	20
3	204	6
4	206	12

Vetor após a ordenação

1	204	11
2	200	20
3	206	6
4	202	12

Ordenação Interna

- Os métodos de ordenação interna são classificados em dois tipos:
  - **Métodos Simples**:
    - mais recomendados para **conjuntos pequenos** de dados.
    - Usam **mais** comparações,
    - produzem códigos menores e mais simples;
  - **Métodos Eficientes ou Sofisticados**:
    - adequados para **conjuntos maiores** de dados.
    - Usam **menos** comparações,
    - produzem códigos mais complexos e com muitos detalhes.

Ordenação Interna

- Ordenação por Seleção (Selection Sort)
- Ordenação por Inserção (Insertion Sort)
- Ordenação por Seleção e Troca (Bubble Sort)
- Ordenação por Inserção através de incrementos decrescentes (ShellSort)
- Ordenação por Particionamento (QuickSort)
- Ordenação por Árvores (HeapSort)

Métodos Simples

Métodos Eficientes

Selection Sort

- **Ideia** do Selection Sort é:
  - a cada passagem pelo vetor, selecionar o menor elemento e colocar este elemento o mais a esquerda possível.
- Um algoritmo de ordenação bastante **simples**.
- Recomendado para **conjuntos pequenos**.
- Etapas:
  - Procurar menor elemento e trocar com o elemento na 1ª posição;
  - Procurar o 2° menor elemento e trocar com o elemento na 2 ª posição;
  - Proceder assim até ordenação estar completa.

Selection Sort

- Exemplo

	0	1	2	3	4	5
Chaves iniciais	44	12	55	42	94	18
i = 0	12	44	55	42	94	18
i = 1	12	18	55	42	94	44
i = 2	12	18	42	55	94	44
i = 3	12	18	42	44	94	55
i = 4	12	18	42	44	55	94

Selection Sort

- **Vantagens**
  - Custo linear no tamanho da entrada para o número de movimentos de registros.
  - É o algoritmo a ser utilizado para arquivos com registros muito grandes.
  - É muito interessante para arquivos pequenos.
- **Desvantagens**
  - O fato de o conjunto já estar ordenado não ajuda em **nada** (o número de comparações continua o mesmo)
  - O algoritmo não é estável, isto é, os registros com chaves iguais **nem sempre** irão manter a **mesma posição** relativa de antes do início da ordenação

Insertion Sort

- **Ideia do insertion Sort**: considerar os elementos um a um e inseri-los em seu lugar entre os elementos já tratados (mantendo essa ordenação).
  - Ex: ordenar cartas de jogar.
- **Inserção** implica em arranjar novo espaço.
  - Mover um número elevado de elementos uma posição para a direita.
  - Elementos à esquerda do índice corrente estão ordenados mas não necessariamente na sua posição final:
    - podem ainda sofrer deslocamento à direita para dar lugar a elementos menores encontrados posteriormente

Insertion Sort

- Etapas:
  - Em cada passo a partir de **i=2** faça:
    - Selecione o i-ésimo item da sequência fonte.
    - Coloque-o no lugar apropriado na sequência destino de acordo com o critério de ordenação.

Insertion Sort

- Exemplo

	0	*1*	*2*	*3*	*4*	*5*
Chaves iniciais	44	12	55	42	94	18
i = 1	12	44	55	42	94	18
i = 2	12	44	55	42	94	18
i = 3	12	42	44	55	94	18
i = 4	12	42	44	55	94	18
i = 5	12	18	42	44	55	94

Insertion Sort

- O **número mínimo de** comparações e movimentos ocorre quando os itens estão originalmente em ordem.
- O **número máximo** ocorre quando os itens estão originalmente na ordem reversa.
- É o método a ser utilizado quando o arquivo está "**quase**" ordenado.
- É um bom método quando se deseja **adicionar** uns poucos itens a um **arquivo ordenado**.
- O algoritmo de ordenação por inserção é **estável** pois deixa os registros com chaves iguais na mesma posição relativa.

Insertion x Selection

- Arquivos já ordenados:
  - **Insertion**: algoritmo descobre imediatamente que cada item já está no seu lugar (custo linear)
  - **Selection**: ordem no arquivo não ajuda (custo quadrático)
- Adicionar alguns itens a um arquivo já ordenado:
  - Insertion sort é o método a ser usado em arquivos "quase ordenados"
- Comparações:
  - Insertion tem um número médio de comparações que é aproximadamente a **metade** do Selection
- Movimentações:
  - Selection tem um número médio de comparações que **cresce linearmente com n**, enquanto que a média de movimentações no Insertion **cresce com o quadrado de n**.

Bubble Sort

- **Idéia**: fazer múltiplas passagens pelos dados trocando de cada vez dois elementos adjacentes que estejam fora de ordem, até não haver mais trocas.
  - supostamente muito fácil de implementar.
  - **usualmente mais lento** que os dois métodos anteriores.
- Talvez o algoritmo mais utilizado por ser o mais conhecido.
- Etapas
  - Percorra o vetor inteiro comparando elementos adjacentes (**dois a dois**).
  - Troque as posições dos elementos se eles estiverem fora de ordem.
  - Repita os dois passos anteriores até que não haja mais trocas.

Bubble Sort

- Exemplo

	0	1	2	3	4	5
Chaves iniciais	44	12	55	42	94	18
1ª passagem	12	44	55	42	94	18
	12	44	42	55	94	18
	12	42	44	55	18	94
2ª passagem	12	42	44	18	55	94
3ª passagem	12	42	18	44	55	94
4ª passagem	12	18	42	44	55	94

Bubble Sort

- Método **muito simples**, mas de **custo elevado**.
- Adequado apenas para pequenos arquivos.
- Ruim se os registros são muito grandes.
- Método extremamente lento:
  - só faz comparações entre posições adjacentes
- É o **método mais ineficiente** entre os métodos simples
- Melhor caso: vetor já ordenado
- Pior caso: vetor de entrada em ordem reversa

Shellsort

- Criado por Donald Shell em **1959**, o Shellsort é o mais eficiente algoritmo de classificação dentre os de complexidade **quadrática**.
- É uma extensão do algoritmo Insertion sort.
- Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em menores.
  - Nos grupos menores é aplicado o método Insertion sort.

Shellsort

- O Shellsort explora o fato de que o **Insertion** sort apresenta desempenho aceitável quando o número de chaves é pequeno e/ou estas já possuem uma ordenação parcial.
- Difere do Insertion sort pelo fato de que considera apenas **um segmento classificado** e inserindo ordenadamente todos os demais elementos.
- Vários segmentos são considerados inicialmente, sendo cada elemento inserido seletivamente em um deles.

Shellsort

- Os itens separados de h posições são rearranjados.
- Todo h-ésimo item leva a uma sequência ordenada.
- Tal seqüência é dita estar h -ordenada.
- Algoritmo
  - Para um **h** inicial, os segmentos assim formados são então classificados pelo Insertion Sort.
  - O incremento h é diminuído (a metade do valor anterior), dando origem a novos segmentos, os quais também serão classificados pelo Insertion Sort.
  - Este processo se repete até que h seja igual a 1.
  - **Quando h = 1 Shellsort corresponde ao algoritmo Insertion Sort.**

Shellsort

- Exemplo:

Original	32	95	16	82	24	66	35	19	75	54	40	43	93	68
After 5-sort	32	35	16	68	24	40	43	19	75	54	66	95	93	82
After 3-sort	32	19	16	43	24	40	54	35	75	68	66	95	93	82
After 1-sort	16	19	24	32	35	40	43	54	66	68	75	82	93	95

Shellsort

- A sequência ótima não foi **ainda descoberta**(se é que existe)
- A análise do algoritmo é desconhecida
- Ninguém encontrou a fórmula que define a complexidade
- Referência: Nívio Ziviani

Shellsort

- Vantagens
  - rápido/eficiente
  - código reduzido
  - boa opção para arquivos pequenos e médios (±10.000 itens)
  - aceitável para elevados volumes de dados
  - por ser um método eficiente, pode ser utilizado em sistemas que não dispõe de muitos recursos de memória
  - O seu tempo de execução é sensível à ordem inicial do programa, o que lhe garante um bom uso em seqüências já ordenadas (herdado do método Insertion sort).

Quicksort

- **Idéia do Quicksort** : dividir o problema de ordenar um conjunto com **n** itens em dois problemas menores.
  - Os problemas menores são ordenados independentemente.
  - Os resultados são combinados para produzir a solução final.
- É o algoritmo de **ordenação interna mais rápido** que se conhece para uma ampla variedade de situações.

Quicksort

- A parte mais delicada do método é relativa à divisão da partição.
  - O vetor **v[esq..dir]** é rearranjado por meio da escolha arbitrária de um pivô **x**.
  - O vetor **v** é particionado em duas partes:
    - A parte esquerda com chaves menores ou iguais a **x**.
    - A parte direita com chaves maiores ou iguais a **x**.

Quicksort

- Algoritmo
  - Escolher um item do vetor e colocar este valor em x (**pivô x**)
  - Percorrer o vetor a partir da esquerda até que **A[i]** ≥ x seja encontrado
  - Percorrer o vetor a partir da direita até que **A[j] ≤ x** seja encontrado
  - Trocar v[i] com v[j].
  - Continuar este processo até os apontadores i e j se cruzarem.
  - Ao final, o vetor A[esq..dir] está particionado de tal forma que:
    - Os itens em A[Esq], A[Esq+1],..., A[j] são menores ou iguais a x
    - Os itens em A[i], A[i+1],..., A[Dir] são maiores ou iguais a x

Quicksort

- Pseudocódigo

```
procedimento QuickSort(X[ ], IniVet, FimVet)
var i, j, pivo
início
   i <- IniVet
   j <- FimVet
   pivo <- X[(IniVet + FimVet) div 2]
   repita
      enquanto (X[i] < pivo) faça
      início
            i <- i + 1
      fim
      enquanto (X[j] > pivo) faça
      início
            j <- j - 1
      fim
      se (i <= j) então
      início
        swap(X[i], X[j])
        i <- i + 1
        j <- j - 1
      fim
  até_que (i > j)
  se (j > IniVet) então QuickSort(X, IniVet, j)
  se (i < FimVet) então QuickSort(X, i, FimVet)
fim
```

Quicksort

- O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.
- Isto faz com que o procedimento Ordena seja chamado recursivamente **n** vezes, eliminando apenas um item em cada chamada.
- O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
  - Para isso basta escolher três itens quaisquer do vetor e usar a mediana dos três como pivô.

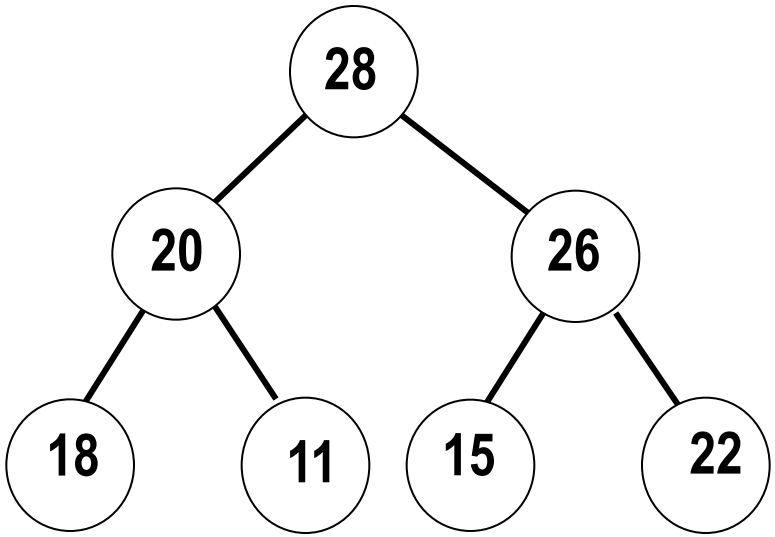
Heapsort

- Utiliza uma heap para organizar informação durante a execução do algoritmo.
- O custo da ordenação é reduzido através da construção da heap.
- Possui o mesmo princípio de funcionamento da ordenação por seleção.
- Idéia do HeapSort:
  - Construir a heap.
  - Trocar o item na posição 1 do vetor (raiz da heap) com o item da posição n.
  - Reconstituir a heap para os itens v[1], v[2], . . . , v[n − 1].
  - Repetir os passos 2 e 3 com os n − 1 itens restantes, depois com os n − 2, até que reste apenas um item.

Heapsort

0	1	2	3	4	5	6
18	20	15	28	11	22	26

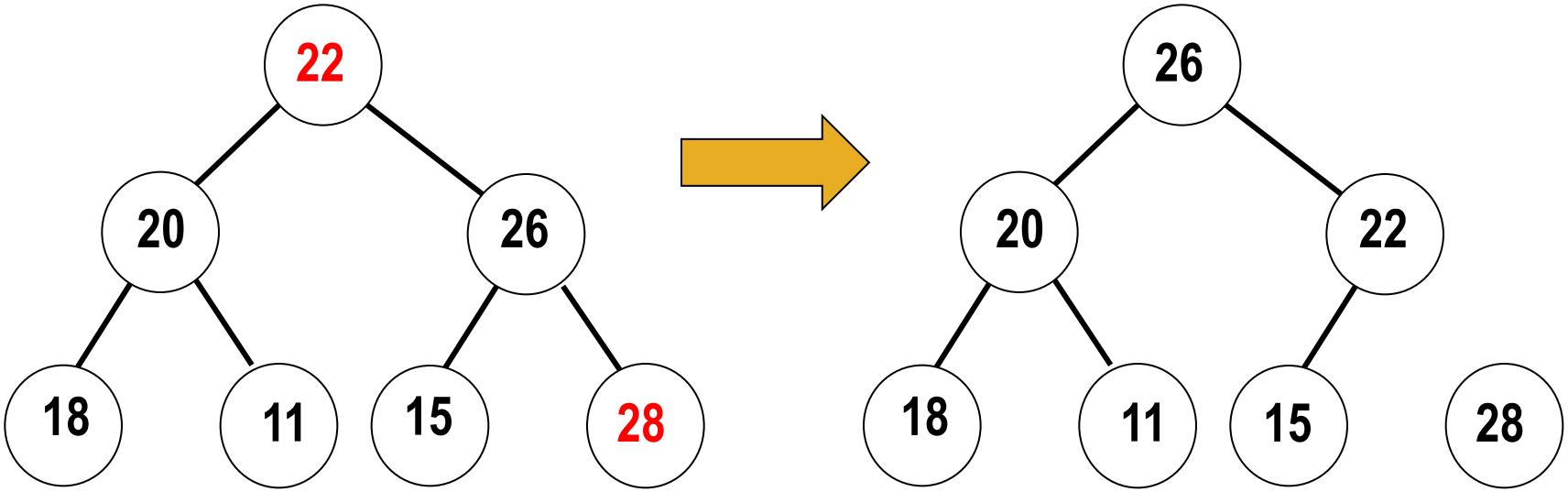
- Construção da *Heap:*



0	1	2	3	4	5	6

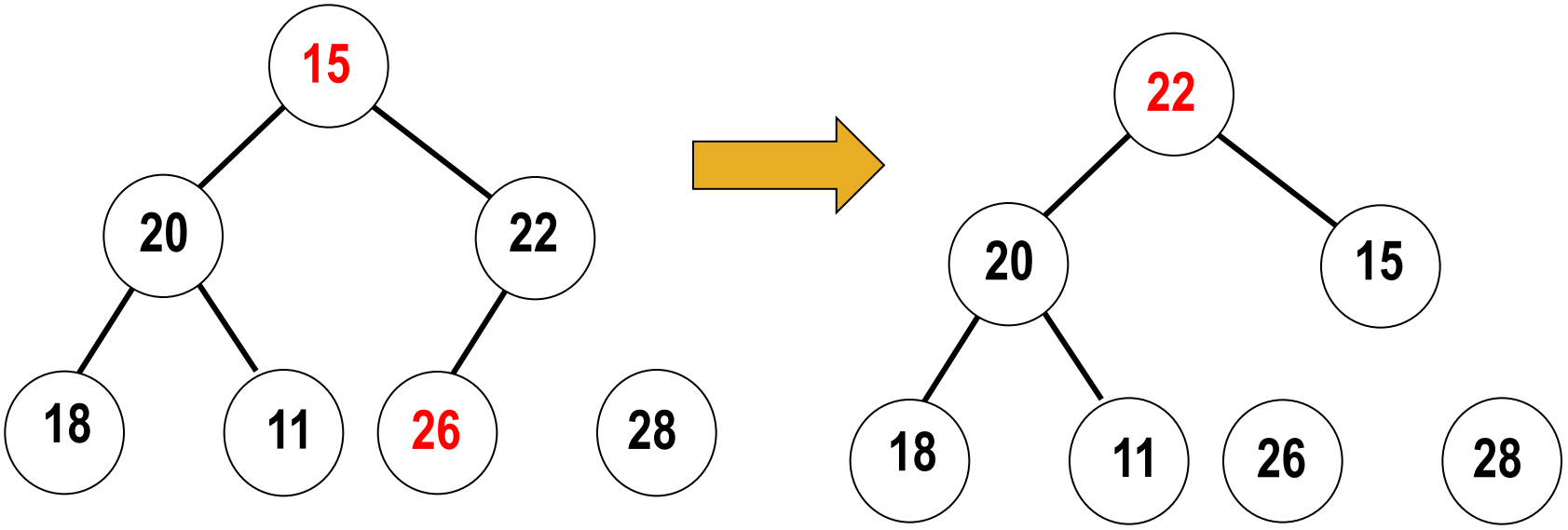
Heapsort

1. Trocar o elemento da raiz (28) da *heap* e colocá-lo na posição do elemento n (22).
2. Reorganizar a heap excluindo o elemento 28.



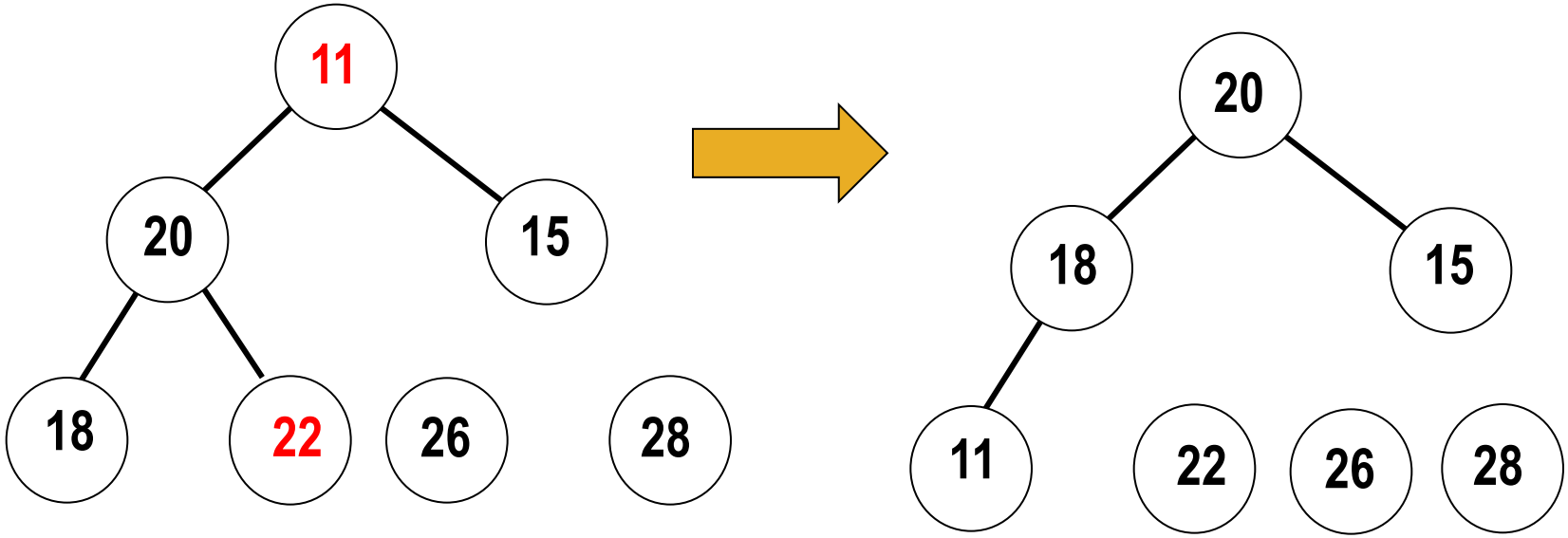
Heapsort

1. Trocar o elemento da raiz (26) da *heap* e colocá-lo na posição do elemento $n-1$ (15).
2. Reorganizar a heap excluindo os elementos 26 e 28.



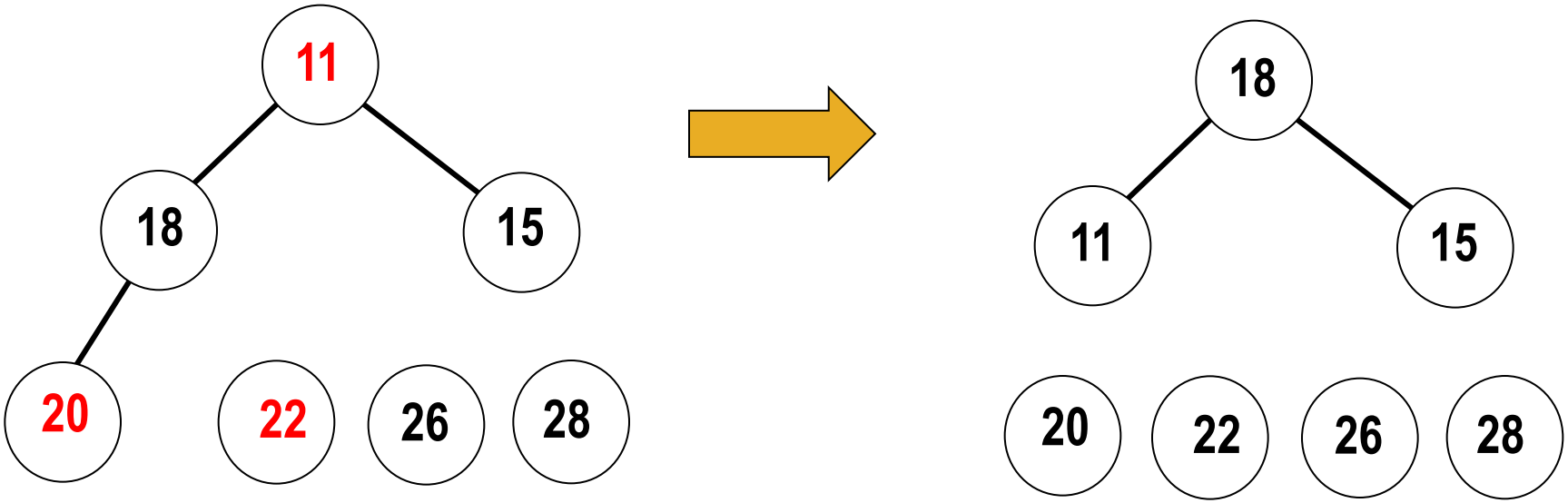
Heapsort

1. Trocar o elemento da raiz (22) da *heap* e colocá-lo na posição do elemento $n-2$ (11).
2. Reorganizar a heap excluindo os elementos 22,26 e 28.



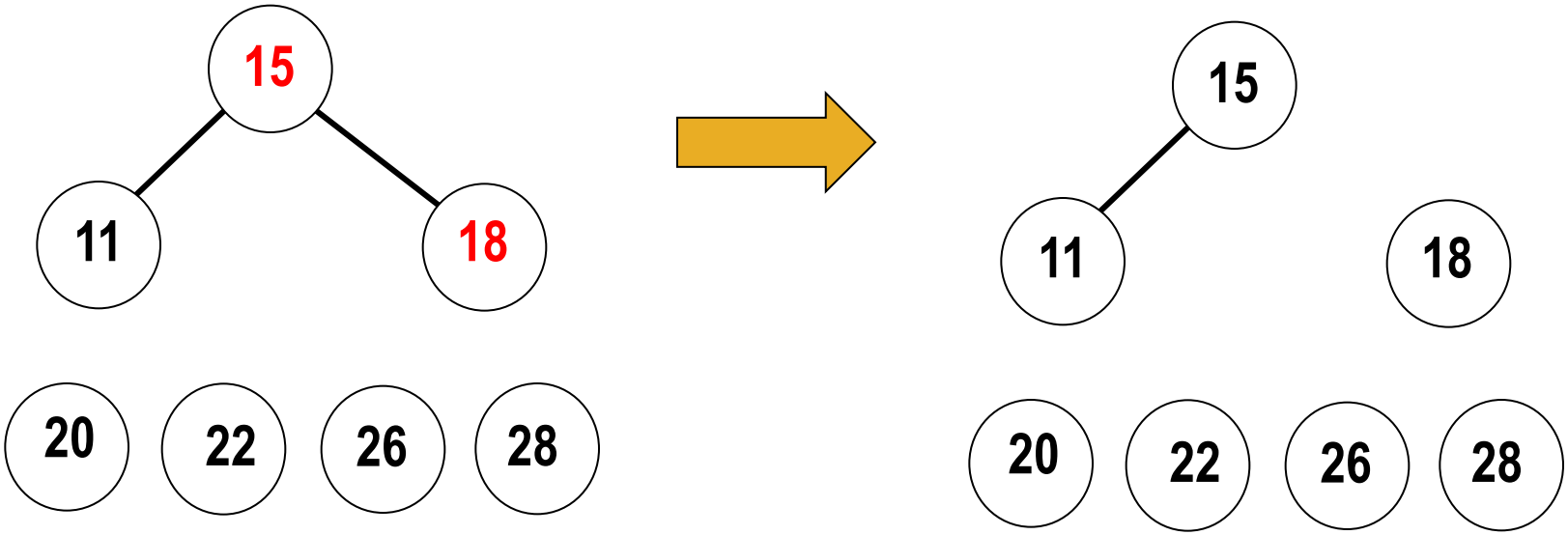
Heapsort

1. Trocar o elemento da raiz (20) da *heap* e colocá-lo na posição do elemento $n-3$ (11).
2. Reorganizar a heap excluindo os elementos 20, 22, 26 e 28.



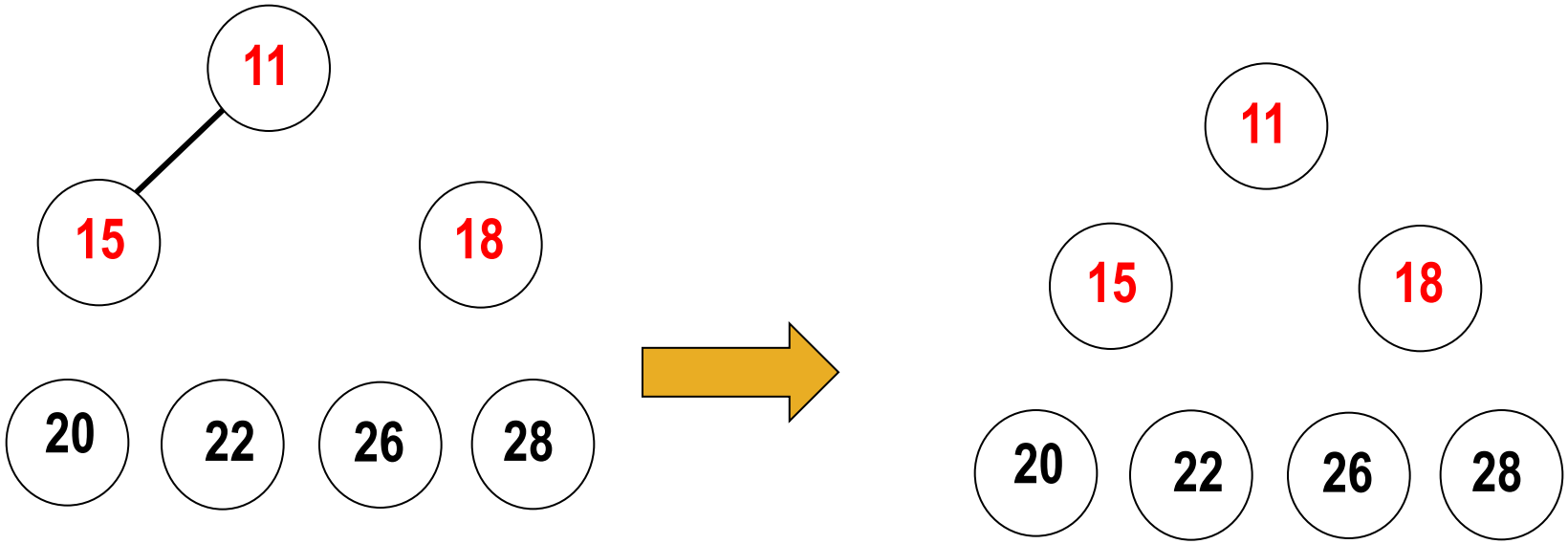
Heapsort

1. Trocar o elemento da raiz (18) da *heap* e colocá-lo na posição do elemento $n-5$ (15).
2. Reorganizar a heap excluindo os elementos 18, 20, 22, 26 e 28.



Heapsort

1. Trocar o elemento da raiz (15) da *heap* e colocá-lo na posição do elemento $n-6$ (11).



0	1	2	3	4	5	6
11	15	18	20	22	26	28

Considerações

- O Heapsort **não é recomendado** para arquivos com poucos registros porque:
  - O tempo necessário para construir **a heap é alto**
  - O anel interno do algoritmo é bastante complexo, se comparado com o anel interno do Quicksort
- O **Quicksort** é, em média, cerca de duas vezes mais rápido que o Heapsort.
- O método não é estável.
- A diferença do comportamento do algoritmo entre o melhor caso e o pior caso é pequena.

Comparação entre os Métodos de Ordenação

- O Quicksort é o mais rápido para todos os tamanhos aleatórios experimentados.
- A relação Heapsort/Quicksort se mantém constante para todos os tamanhos, sendo o Heapsort mais lento.
- A relação Shellsort/Quicksort aumenta à medida que o número de elementos aumenta;
- O Insertion sort é o mais rápido para qualquer tamanho se os elementos estão ordenados;
  - Ele é o mais lento para qualquer tamanho se os elementos estão em ordem descendente;

Comparação entre os Métodos de Ordenação

- O Shellsort é bastante sensível à ordenação ascendente ou descendente da entrada;
- O Quicksort é sensível à ordenação ascendente ou descendente da entrada.
- **O Heapsort praticamente não é sensível à ordenação da entrada.**

Comparação entre os Métodos de Ordenação

- **Selection Sort**.
  - É vantajoso quanto ao número de movimentos de registros
  - Deve ser usado para arquivos com registros muito grandes, desde que o tamanho do arquivo não exceda 1.000 elementos.
- **Shellsort**
  - É o método a ser escolhido para a maioria das aplicações por ser muito eficiente para arquivos de tamanho moderado.
  - Mesmo para arquivos grandes, o método é cerca de apenas duas vezes mais lento do que o Quicksort .
  - Sua implementação é simples e geralmente resulta em um programa pequeno.
  - Quando encontra um arquivo parcialmente ordenado trabalha menos.

Comparação entre os Métodos de Ordenação

- Quicksort
  - É o algoritmo mais eficiente que existe para uma grande variedade de situações.
  - O algoritmo é recursivo, o que demanda memória adicional.
  - O principal cuidado a ser tomado é com relação à escolha do pivô.
  - A escolha do elemento do meio do arranjo melhora muito o desempenho quando o arquivo está total ou parcialmente ordenado.

Comparação entre os Métodos de Ordenação

- Heapsort
  - Apesar de ser cerca de duas vezes mais lento do que o Quicksort , não necessita de nenhuma memória adicional.
  - Aplicações que não podem tolerar eventuais variações no tempo esperado de execução devem usar o Heapsort .

Exercícios

- Implementar os algoritmos abaixo para ordenar uma lista encadeada.
  - Selection Sort
  - Insertion-Sort
  - Bubble-Sort.
  - Shell-Sort
  - Quick-Sort.