Estrutura de Dados

Prof. Rafael Nunes

Algoritmos de Busca e Ordenação

Search and Sorting Algorithms

Parte 1

Ordenação

O que significa Ordenar?

Ordenar

- Ordenar:
- Processo utilizado para *rearranjar um conjunto de dados/objetos* em uma certa ordem, ascendente ou descendente…

- … para facilitar e *aumentar a eficiência das operações* de pesquisa sobre esse conjunto de dados/objetos

Vejamos um *exemplo* …

Ordenar

- Exemplo:
- Imagine procurar o telefone de uma determinada pessoa em uma lista telefônica
  - Esta simples tarefa de consulta poderia ser *extremamente exaustiva* se a lista não estivesse classificada por ordem alfabética dos nomes dos assinantes.

Antes de apresentarmos os algoritmos

Devemos ficar atentos a ...

Importante

- Geralmente devemos considerar como seqüência de entrada para os algorimos um *vetor com n elementos*...

- ... mas também existe a possibilidade de utilizarmos outras estruturas como entrada, por exemplo:
  - uma *lista encadeada*
  - uma *árvore binária*

Importante

- Devemos também considerar cada elemento a ser ordenado como um *conjunto de dados* denomidado *Registro*...

- ... e não como apenas um elemento de forma isolada.

O que é um Registro?

Você se lembra?

Importante

- Registro:
- Um registro geralmente contém uma *chave*, que é o valor a ser ordenado, e os demais valores (ou campos), que acompanham essa chave.
- Ou seja, quando realizarmos o processo de ordenação, se a chave necessitar ser trocada de posição, também necessitaremos alterar *a posição de todos os elementos* contidos no registro.

Como podemos otimizar essa tarefa de ordenação quando os *registros forem muito grandes*?

Otimizando!

- Resp:

- Podemos realizar o processo de ordenação *através de um vetor (ou lista) de ponteiros*.

Vejamos algumas possibilidades ...

Contigüidade física x Vetor Indireto x Encadeamento

Contigüidade Física

Os elementos são movidos juntos com a chave na ordenação

Contigüidade Física

	Chave	*Demais Campos*		
1	4	PP	PP	PP
2	7	FF	FF	FF
3	2	GG	GG	GG
4	1	DD	DD	DD
5	3	SS	SS	SS
6	5	ZZ	ZZ	ZZ

Tabela ***não-ordenada***

	Chave	*Demais Campos*		
1	1	DD	DD	DD
2	2	GG	GG	GG
3	3	SS	SS	SS
4	4	PP	PP	PP
5	5	ZZ	ZZ	ZZ
6	7	FF	FF	FF

Tabela ***ordenada***

Vetor Indireto

Um *vetor auxiliar* é criado na ordenação. Ele mantém *apontadores* para os registros reais, na ordem desejada. *Os elementos não se movem*.

Vetor Indireto

	Chave	*Demais Campos*		
1	4	PP	PP	PP
2	7	FF	FF	FF
3	2	GG	GG	GG
4	1	DD	DD	DD
5	3	SS	SS	SS
6	5	ZZ	ZZ	ZZ

Tabela ***não-ordenada***

	Índice da Tabela Original
1	***4***
2	***3***
3	***5***
4	***1***
5	***6***
6	***2***

Vetor de ordenação

Encadeamento

A ordem é feita através de uma *lista encadeada*. Adiciona-se um campo a mais na tabela original para identificar o próximo registro. Também é necessário utilizar um ponteiro para identificar a primeira posição. Os elementos também não se movem.

Encadeamento

	Chave	Demais Campos			Endereço Próximo Registro
1	4	PP	PP	PP	6
2	7	FF	FF	FF	-
3	2	GG	GG	GG	5
4	1	DD	DD	DD	3
5	3	SS	SS	SS	1
6	5	ZZ	ZZ	ZZ	2

Tabela ***não-ordenada***

Tipos de Ordenação

Ordenação Interna

x

Ordenação Externa

Tipos de Ordenação

- Ordenação Interna:

- Onde o conjunto de dados a serem ordenados *cabe inteiramente* (ou quase) *na memória principal*

Tipos de Ordenação

- Ordenação Externa:

- Onde o conjuntos de dados a serem ordenados *não cabem inteiramente na memória principal* e precisam ser armazenados em memória auxiliar como:
  - Discos
  - Fitas

Qual é a principal diferença entre esses dois métodos?

Sugestões!

Tipos de Ordenação

- Resp:
- No método de ordenação interna qualquer registro pode ser acessado imediatamente ....
- ... enquanto que no método de ordenação externa os registros *são acessados seqüencialmente* ou *em blocos*.

- Isso permite que os registros caibam na memória principal.
  - O arquivo é lido uma vez inteiramente e depois divido em blocos para caber na memória principal

Iremos estudar os principais métodos de ordenação interna

Veja a seguir

Algoritmos de *Ordenação Interna*

Algoritmos de Ordenação Interna

- Ordenação por ***Troca***
  - Neste processo faz-se a varredura do vetor como um todo.
  - Realiza comparações sucessivas de pares de elementos, trocando-os de posição quando estiverem fora de ordem
- Ordenação por ***Inserção***
  - Obtêm-se a ordenação inserindo os elementos na sua posição correta, levando em consideração os elementos já ordenados
- Ordenação por ***Seleção***
  - Obtêm-se a ordenação através da seleção sucessiva do maior (ou menor) valor contido no vetor.
  - Este valor é colocado posteriormente na sua posição correta

Algoritmos de Ordenação Interna

- Ordenação por ***Troca***
  - Bubble Sort – Método da Bolha
  - Quick Sort – Método da Troca e Partição
- Ordenação por ***Inserção***
  - Insertion Sort – Inserção direta
  - Shell Sort – Incrementos Decrescentes
- Ordenação por ***Seleção***
  - Selection Sort – Seleção direta
  - Heap Sort – Seleção em Árvore
- Ordenação por ***Intercalação***
  - Merge Sort

Podemos também classificar tais algoritmos por sua *eficiência!*

Veja a seguir ...

Algoritmos de Ordenação Interna

- Complexidade ***O(n^2)***
  - Bubble Sort – Método da Bolha
  - Insertion Sort – Inserção direta
  - Selection Sort – Seleção direta
  - Shell Sort – Incrementos Decrescentes
- Complexidade ***O(n log n)***
  - Merge Sort – Metodo da Intercalação
  - Quick Sort – Método da Troca e Partição
  - Heap Sort – Seleção em Árvore

Algoritmos de *Ordenação Interna*

Complexidade $O(n^2)$

Bubble Sort

Método da bolha

Bubble Sort

- Caracteristicas:
  - Este processo é chamado de bolha devido ao seu método de funcionamento, que indiretamente, faz com que *os maiores valores* de uma seqüência desordenada *"borbulhem" para o final* da mesma.
  - Igualmente às bolhas de gás em um copo de cerveja, ou refrigerante (pra quem não bebe), que sobem do fundo para a superfície do copo onde elas estão.

Bubble Sort

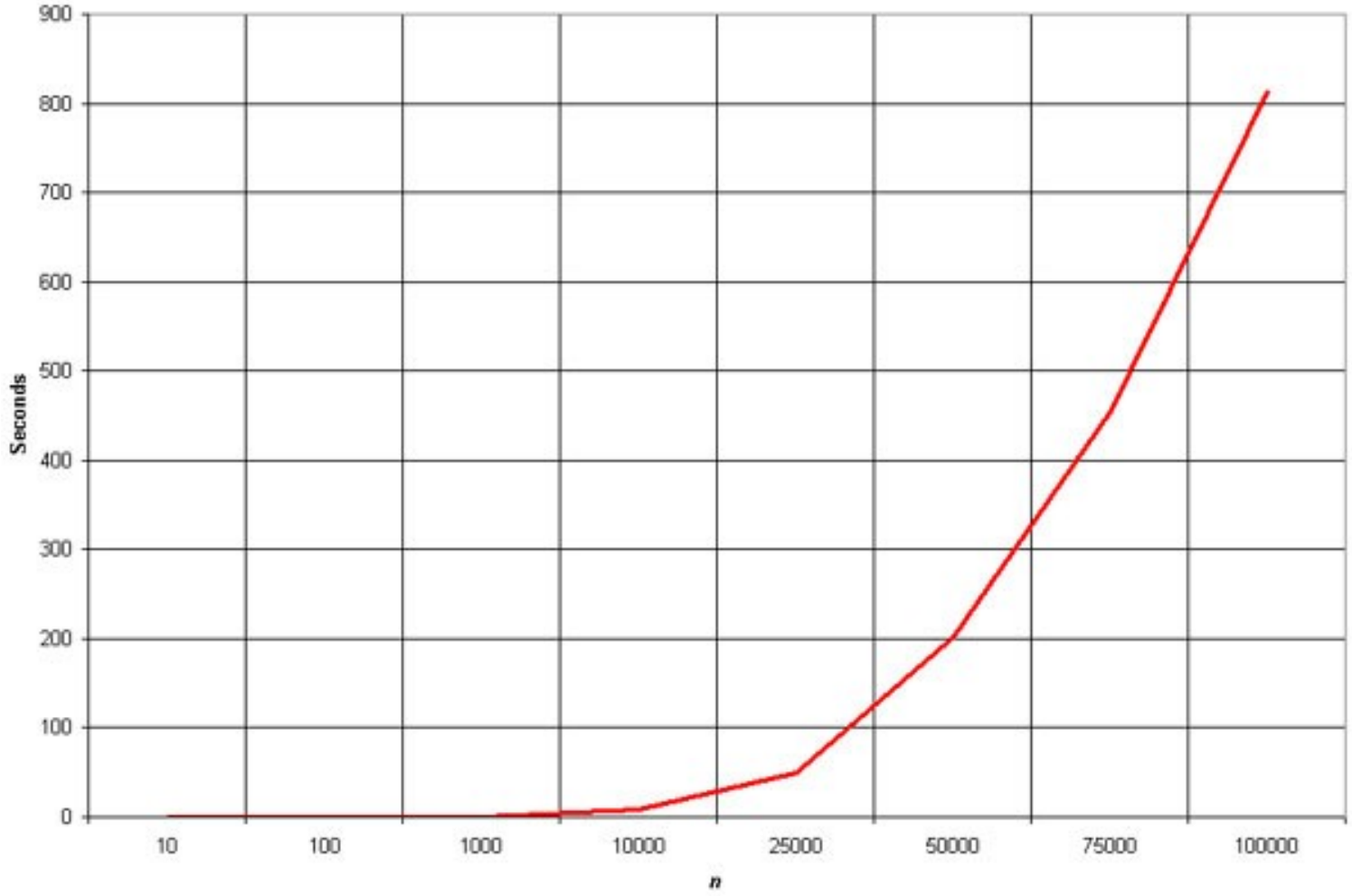
- Pros
  - Fácil entendimento
  - Fácil de implementar
- Contra
  - Pior eficiência dentre todos os algoritmos

Bubble Sort – $O(n^2)$

- Eficiência:
  - É considerado como o algoritmo de menor eficiência dentre os algoritmos de ordenação.
  - Em condições de melhor caso (quando o vetor ou a lista já estão ordenados), o algoritmo bubble sort tem complexidade de nível constante...
  - ... ou seja, ordem de complexidade igual ao numero de entradas – O(n).
  - Na maioria das vezes ele ultrapassa $O(n^2)$, o que é péssimo quando o numero de dados a serem ordenados é muito grande.

Bubble Sort – $O(n^2)$

- Eficiência:



Bubble Sort

- Execução (6 Iterações)

Bubble Sort para vetor de 4 posições (0-3)

0	1	2	3	--
8	**4**	10	2	8 > 4 (troca)
4	**8**	**10**	2	8 > 10 (não-troca)
4	8	**10**	**2**	10 > 2 (troca)
4	**8**	2	10	4 > 8 (não-troca)
4	**8**	**2**	10	8 > 2 (troca)
4	**2**	8	10	4 > 2 (troca)
2	4	8	10	Fim - Ordenado!

Insertion Sort

Inserção direta

Insertion Sort

- Como o próprio nome já diz, o algoritmo de ordenação – insertion sort – ordena um conjunto de dados por *meio de inserções*.

- Ele insere cada elemento, no vetor ou na lista final, já na sua posição correta...

- ... deslocando os demais elementos da estrutura para posições mais a frente.

Insertion Sort

- A simples implementação deste algoritmo demanda duas estruturas do mesmo tipo (dois vetores ou duas listas).

- A estrutura de origem possui os dados que você deseja ordenar e a estrutura de destino será onde os dados serão inseridos já ordenados.

Insertion Sort

- Para economizar memória, a maioria das implementações *utilizam uma única estrutura* onde o elemento a ser inserido vai caminhando contrário a lista até chegar na sua posição correta...

- ou seja, se um elemento antes dele for maior do que ele próprio, ele passa para a posição desse elemento, deslocando-o uma posição a frente.

Insertion Sort

- Pros
  - Fácil entendimento
  - Fácil de implementar

- Contra
  - Ineficiente para uma quantidade grande de dados a serem ordenados.

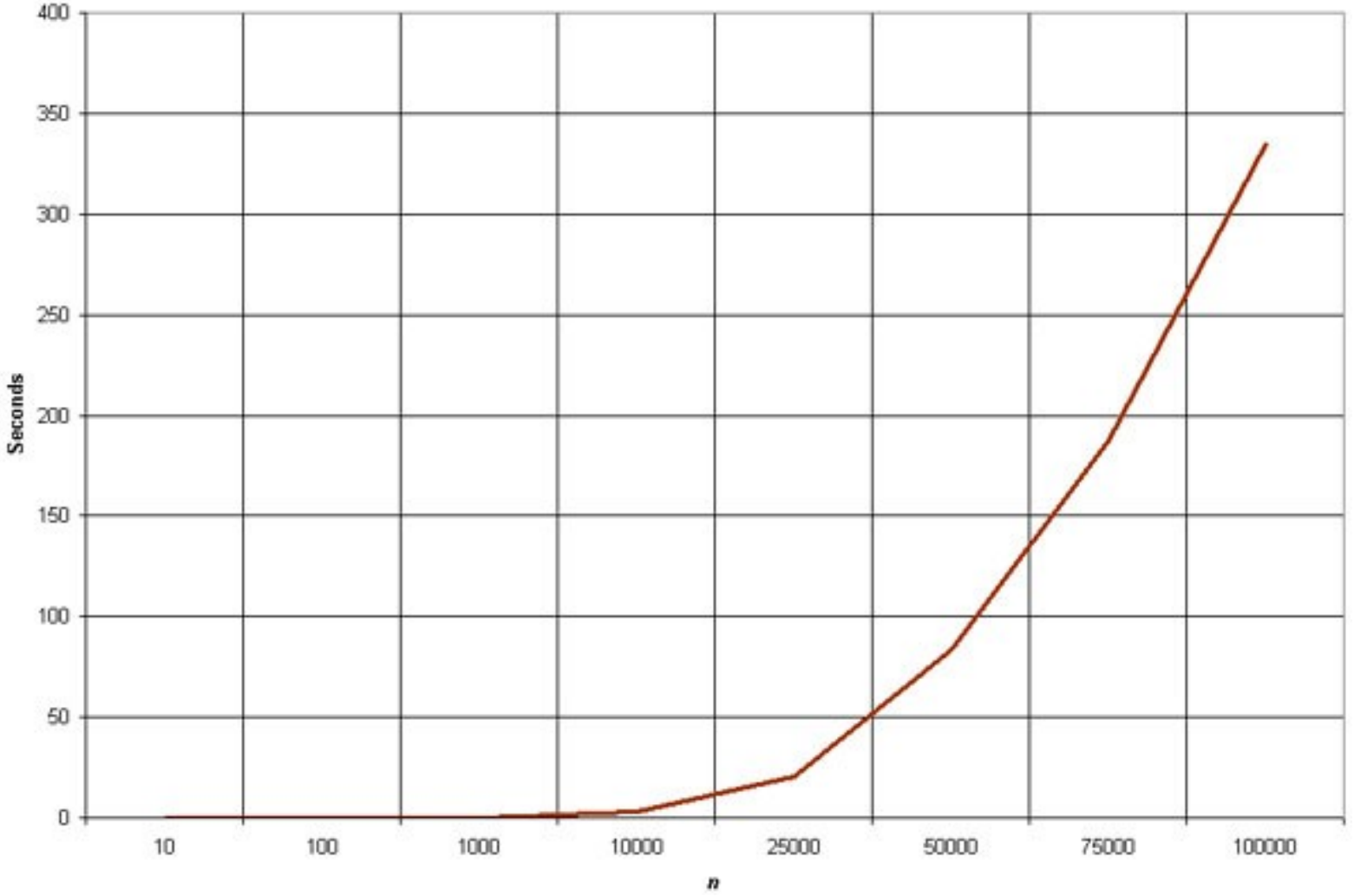
Insertion Sort – $O(n^2)$

- Eficiência:
- O algoritmo insertion sort tem a mesma complexidade $O(n^2)$ do algoritmo bubble sort.

- Apesar disso, o algoritmo de inserção consegue ser um pouco melhor (mais eficiente) do que o algoritmo bubble sort.
  - é mais rápido do que o bubble sort, mais do que o dobro e 40% mais rápido do que o selection sort.

Insertion Sort – $O(n^2)$

- Eficiência:



Insertion Sort

- Execução (7 Iterações)

Insertion Sort para vetor de 4 posições (0-3)

0	1	2	3	--	aux
8		10	2	8 > aux (move 8 dir)	4
8	**8**	10	2	insere aux na posição	4
4	8		2	8 > aux (não-move) insere aux na posição	10
4	8	**10**		10 > aux (move 10 dir)	2
4	**8**		**10**	8 > aux (move 8 dir)	2
4		**8**	10	4 > aux (move 4 dir)	2
	4	8	10	Insere aux na posição	2
2	4	8	10	Fim - Ordenado!	

Selection Sort

Seleção direta

Selection Sort

- Seleciona *o menor valor* na estrutura de dados e o coloca na primeira posição.

- Ignora a primeira posição, seleciona o segundo menor e o coloca na segunda posição e assim sucessivamente até a estrutura (Lista ou Vetor) estar ordenada.

Selection Sort

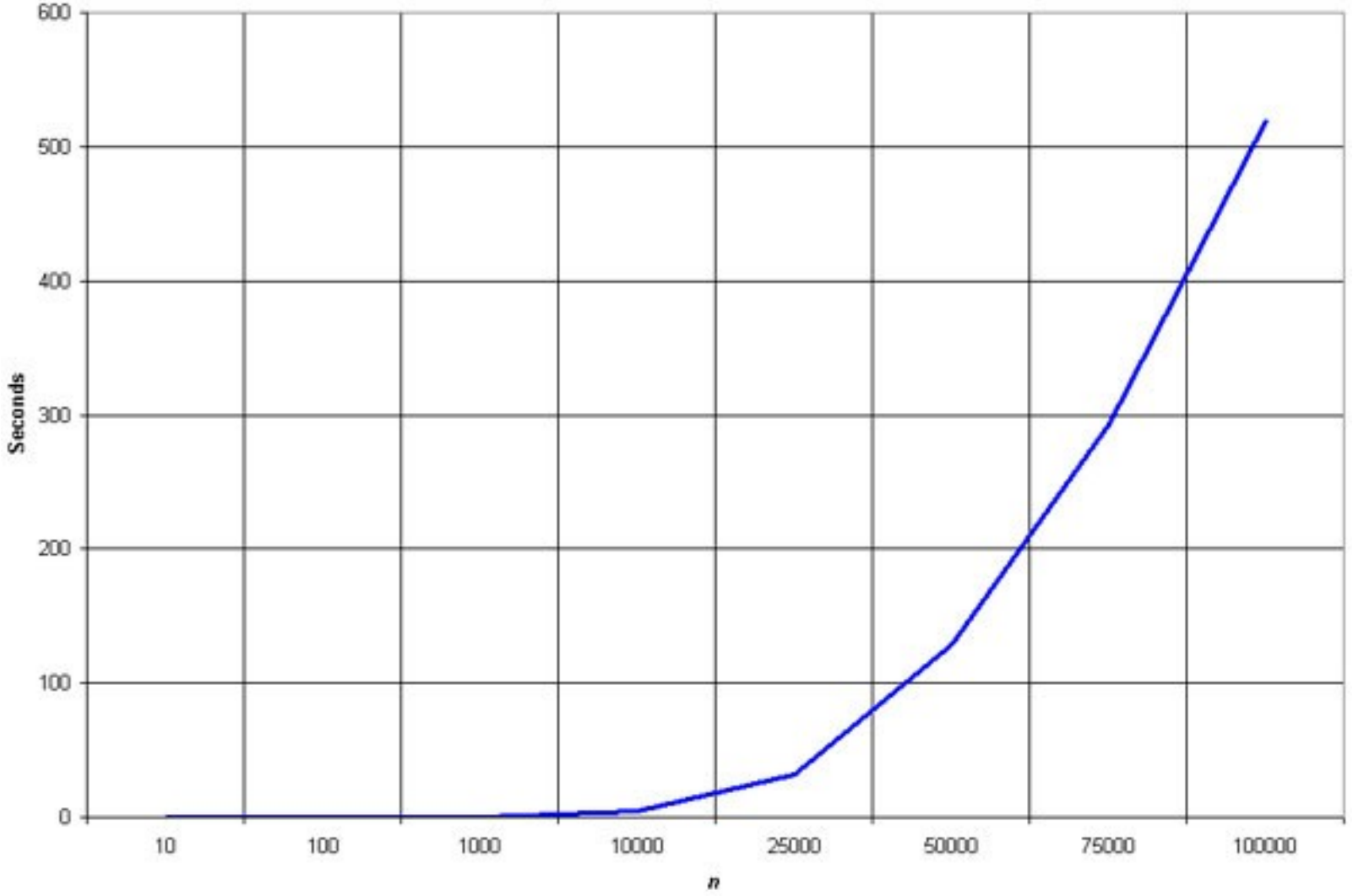
- Pros
  - Fácil entendimento
  - Fácil de implementar

- Contra
  - Não é eficiente para grandes estruturas de dados.
  - É tão similar ao algoritmo insertion sort (mais eficiente), que raramente é utilizado.

A complexidade do selection sort também é $O(n^2)$

Selection Sort – $O(n^2)$

- Eficiência:



Selection Sort

- O algoritmo selection sort é o patinho feio da família dos algoritmos de ordenação.
  - Ele consegue obter um desempenho até 60% melhor do que o bubble sort...
- O insertion sort, muito parecido com o selection sort, consegue obter mais do que o dobro de desempenho em comparação com o algoritmo bubble sort...
  - ou seja, não existe nenhuma razão para se utilizar o selection sort, utilize sempre o insertion sort.

Selection Sort

- Execução (9 Iterações)

Selection Sort para vetor de 4 posições (0-3)

0	1	2	3	--	menor
8	4	10	2	4 < menor (troca menor)	8
8	4	10	2	10 < menor (não-troca)	4
8	4	10	2	2 < menor (troca menor)	2
2	4	10	8	Troca menor com pos[0]	2
2	4	10	8	10 < menor (não-troca)	4
2	4	10	8	8 < menor (não-troca)	4
2	4	10	8	Mantem menor na pos[1]	4
2	4	10	8	8 < menor (troca menor)	10
2	4	8	10	Fim - Ordenado!	

Shell Sort

Incrementos decrescentes

Shell Sort

- Este método de ordenação foi desenvolvido por Donald Shell em 1959

- Ele é o mais eficiente dentre algoritmos que possuem complexidade $O(n^2)$ e por isso é também o mais difícil de se implementar.

Shell Sort

- O shell sort é uma extensão do algoritmo de inserção direta (insertion sort).
- A única diferença esta no fato de que o algoritmo shell sort primeiro divide o vetor a ser ordenados em sub-partes (segmentos / blocos)
- ... para depois aplicar o algoritmo insertion sort separadamente em cada uma delas.

Shell Sort

- Pros
  - Faz ordenações parciais do vetor, aumentando o desempenho nos passos seguintes, pois a inserção direta é acelerada quando o vetor está parcialmente ordenado.
- Contra
  - É considerado, de certa forma, complexo.
  - Não chega nem perto da eficiência dos algoritmos Merge, Heap e Quick Sort (n log n), mas é tão difícil quanto eles para se implementar.

A complexidade do shell sort também é $O(n^2)$

Shell Sort

- O algoritmo shell sort é de longe o mais rápido dentre os algoritmos de ordenação de complexidade n^2.

- Ele é mais do que 5 vezes mais rápido se comparado ao algoritmo bubble sort e um pouco mais do que 2 vezes mais rápido se comparado ao insertion sort.

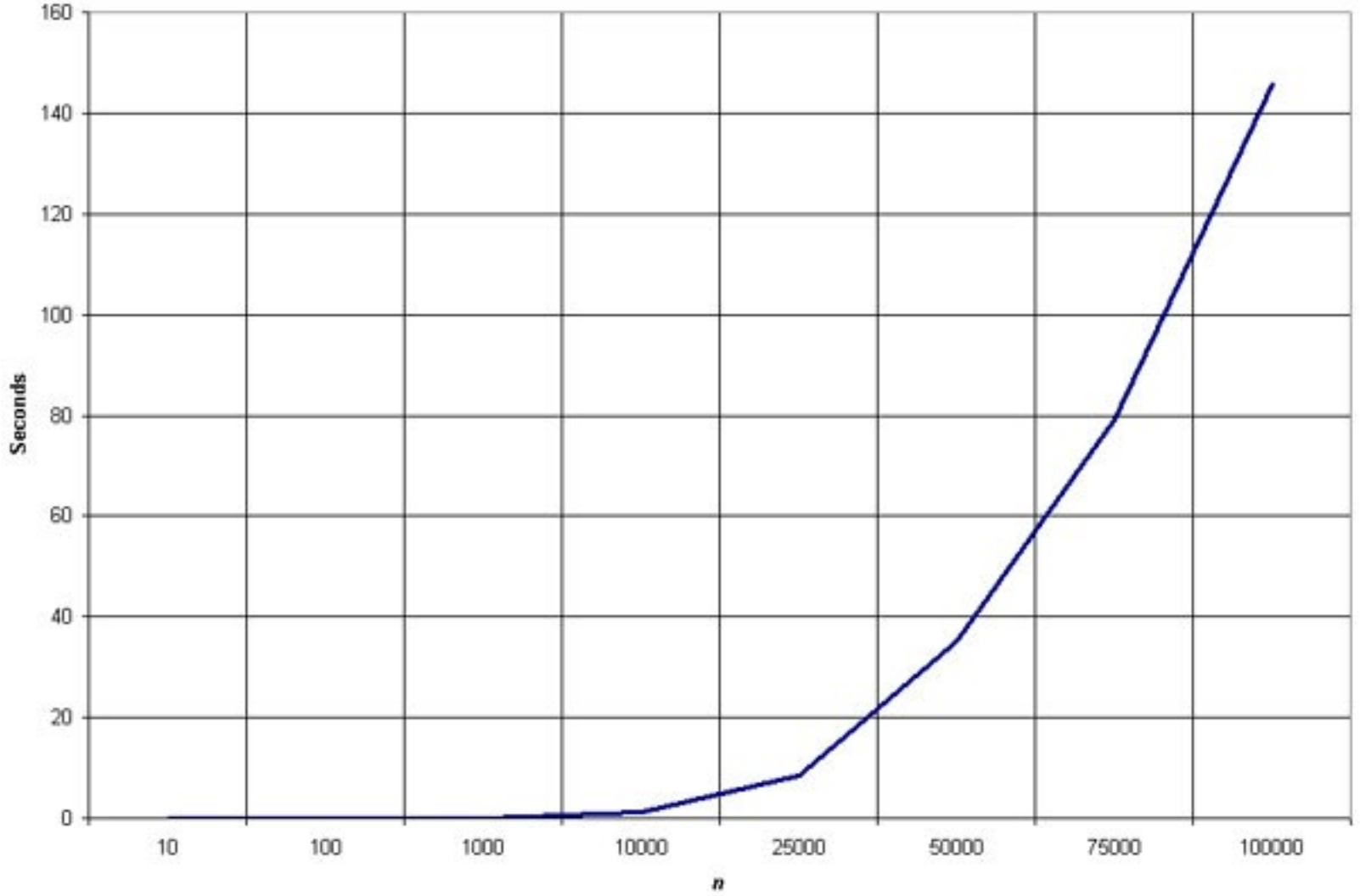
Shell Sort

- O shell sort é ainda significantemente mais lento do que os algoritmos de ordenação merge, heap e quick sort, mas ainda é uma boa escolha para ordenar listas e vetores com até 5000 elementos.

- É também uma boa escolha para ordenar pequenas listas que necessitam ser ordenadas repetitivamente.

Shell Sort – $O(n^2)$

- Eficiência:



Shell Sort

Execução

Shell Sort - Execução

- Vetor original (não-ordenado)

0	1	2	3	4	5	6	7	8	9	10	11
17	29	42	15	21	22	47	37	52	43	27	12

- Para NP=2 (Numero de Passos)
  - $i = 2^{NP}$
  - $i = 2^2 = 4$
  - $i = 2^1 = 2$
  - $i = 2^0 = 1$, ou seja, 3 passos para NP=2

Shell Sort - Execução

- Para sabermos quantos elementos estará em cada segmento:
- N/i, onde N é o tamanho do vetor.
  - Ex:
    - N / i = 12 / 4 = 3 (elementos)

- Resumindo:
  - O vetor será dividido em 4 segmentos e cada segmento terá 3 elementos com distancia 4 entre os índices

Shell Sort - Execução

- Passo 1
  - i = $2^{NP} = 2^2 = 4$ segmentos

Segmento 1			Segmento 2			Segmento 3			Segmento 4		
0	4	8	1	5	9	2	6	10	3	7	11
17	21	52	29	22	43	42	47	27	15	37	12

Aplicar a inserção direta em cada segmento

0	4	8	1	5	9	2	6	10	3	7	11
17	21	52	22	29	43	27	42	47	12	15	37

Obtem-se o vetor

0	1	2	3	4	5	6	7	8	9	10	11
17	22	27	12	21	29	42	15	52	43	47	37

Shell Sort - Execução

- Passo 2
  - i = i / 2 = 4 / 2 = 2 segmentos

Segmento 1

0	2	4	6	8	10
17	27	21	42	52	47

Segmento 2

1	3	5	7	9	11
22	12	29	15	43	37

Aplicar a inserção direta em cada segmento

0	2	4	6	8	10
17	21	27	42	47	52

1	3	5	7	9	11
12	15	22	29	37	43

Obtem-se o vetor

0	1	2	3	4	5	6	7	8	9	10	11
17	12	21	15	27	22	42	29	47	37	52	43

Shell Sort - Execução

- Passo 3
  - i = i / 2 = 2 / 2 = 1 segmento

Aplicar a inserção direta no segmento

0	1	2	3	4	5	6	7	8	9	10	11
17	12	21	15	27	22	42	29	47	37	52	43

Obtem-se o vetor – Ordenado!

0	1	2	3	4	5	6	7	8	9	10	11
12	15	17	21	22	27	29	37	42	43	47	52

Exercícios

Atividade Auto-Instrucional

Exercícios (4hs)

- Implemente todos os algoritmos desta apresentação utilizando os vetores dados nos exemplos
- O uso dos protótipos abaixo são obrigatórios:
  - void bubbleSort(int *numbers, int array_size)
  - void insertionSort(int *numbers, int array_size);
  - void selectionSort(int *numbers, int array_size);
  - void shellSort(int *numbers, int array_size);
- Adicione um contador para verificar o numero de iterações de cada algoritmo.

Até a próxima...