Comparação entre os métodos de ordenação

Algoritmos e Estruturas de Dados II

Número de comparações

Método	Complexidade
Inserção	$O(n^2)$
Seleção	$O(n^2)$
Bolha	$O(n^2)$
Shellsort	$O(n\ lg(n)^2)$
Quicksort	$O(n\ lg(n))$
Heapsort	$O(n\ lg(n))$
Radixsort	$O(kn)$

Tempo de execução

- O método que levou menos tempo real para executar recebeu o valor 1 e os outros receberam valores relativos
- Elementos em ordem aleatória:

	5.00	5.000	10.000	30.000
Inserção	11,3	87	161	–
Seleção	16,2	124	228	–
Shellsort	1,2	1,6	1,7	2
Quicksort	1	1	1	1
Heapsort	1,5	1,6	1,6	1,6

Tempo de execução

- Elementos em ordem crescente

	500	5.000	10.000	30.000
Inserção	1	1	1	1
Seleção	128	1.524	3.066	–
Shellsort	3,9	6,8	7,3	8,1
Quicksort	4,1	6,3	6,8	7,1
Heapsort	12,2	20,8	22,4	24,6

Tempo de execução

- Elementos em ordem decrescente

	500	5.000	10.000	30.000
Inserção	40,3	305	575	–
Seleção	29,3	221	417	–
Shellsort	1,5	1,5	1,6	1,6
Quicksort	1	1	1	1
Heapsort	2,5	2,7	2,7	2,9

Observações

- Observações sobre os métodos:
  - 1. Shellsort, quicksort e heapsort têm a mesma ordem de grandeza para arranjos de até 30 mil elementos
  - 2. O quicksort é o mais rápido para todos os tamanhos aleatórios experimentados
  - 3. A relação heapsort/quicksort se mantém constante para todos os tamanhos de entrada
  - 4. A relação shellsort/quicksort aumenta com o tamanho da entrada
  - 5. Para arquivos pequenos (500 elementos), o shellsort é mais rápido que o heapsort (e vice versa)
  - 6. Inserção é o método mais rápido para qualquer tamanho se os elementos estão ordenados (e vice versa)
  - 7. Entre os algoritmos de custo quadrático, o inserção é melhor para entradas aleatórias

Influencia da ordem inicial dos elementos

	Shellsort			Quicksort			Heapsort		
	5.000	10.000	30.000	5.000	10.000	30.000	5.000	10.000	30.000
Asc	1	1	1	1	1	1	1,1	1,1	1,1
Des	1,5	1,6	1,5	1,1	1,1	1,1	1	1	1
Ale	2,9	3,1	3,7	1,9	2,0	2,0	1,1	1	1

- 1. O shellsort é bastante sensível à ordenação ascendente ou descendente da entrada
- 2. Em arquivos do mesmo tamanho, o shellsort executa mais rápido para arquivos ordenados
- 3. Em arquivos do mesmo tamanho, o quicksort executa mais rápido para arquivos ordenados
- 4. O heapsort não depende da ordenação da entrada

Inserção

- É o mais interessante para arquivos com menos do que 20 elementos
- O método é estável
- Possui comportamento melhor do que o método da bolha que também é estável
- Sua implementação é tão simples quanto as implementações do bolha e seleção
- Para arquivos já ordenados, o método é O(n)
- O custo é linear para adicionar alguns elementos a um arquivo já ordenado

Seleção

- É vantajoso quanto ao número de movimentos de registros, que é O(n)
- Deve ser usado para arquivos com elementos muito grandes, desde que o número de elementos a ordenar seja pequeno

Shellsort

- Bom para ordenar um número moderado de elementos
- Quando encontra um arquivo parcialmente ordenado trabalha menos

Quicksort

- É o algoritmo mais eficiente que existe para uma grande variedade de situações
- O algoritmo é recursivo, o que demanda uma pequena quantidade de memória adicional
- Pior caso realiza $O(n^2)$ operações
- O principal cuidado a ser tomado é com relação à escolha do pivô
  - A escolha do elemento do meio do arranjo melhora o desempenho quando o arquivo está total ou parcialmente ordenado
  - O pior caso tem uma probabilidade muito pequena de ocorrer quando os elementos forem aleatórios
  - Geralmente se usa a mediana de uma amostra de três elementos para evitar o pior caso
- Usar inserção em partições pequenas melhora desempenho significativamente

Heapsort

- É um método de ordenação elegante e eficiente
- Não necessita de nenhuma memória adicional
- Executa sempre em tempo proporcional a $O(n \lg(n))$
- Aplicações que não podem tolerar eventuais variações no tempo esperado de execução devem usar o heapsort