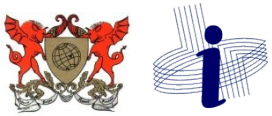
Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas

INF 112
Programação 2
Aula “8”
Ordenação - 2

ShellSort

- O algoritmo ShellSort foi proposto por Ronald Shell em 1959.
- É uma extensão do método de inserção.
- Shell observou que o algoritmo de inserção tem os seguintes problemas:
  - Troca itens adjacentes para determinar o ponto de inserção.
  - Quando o menor item está mais à direita no vetor, são realizadas muitas trocas de itens.
- A ideia do método é contornar tais problemas realizando trocas de elementos distantes um dos outros.



ShellSort

- Funcionamento:
  - Primeiro, define-se uma distância h e os elementos *h-distantes* são ordenados utilizando ordenação por inserção.
  - A ordenação por inserção é realizada apenas em subconjuntos dos elementos a serem ordenados. Tais subconjuntos são bem menores do que o vetor original.
  - Note que o método da inserção é eficiente para processar conjuntos pequenos de dados.

ShellSort

- Funcionamento:
  - Após ordenar todos os “subvetores” *h-distantes*, o valor de *h* é decrementado e o processo reiniciado.
  - Com um valor menor de h, os subvetores ficam maiores. Porém, os dados estarão mais ordenados – nesse caso, o algoritmo da inserção é mais eficiente.
  - O valor de *h* é decrementado até se tornar 1. Quando *h* vale 1, o algoritmo se comporta de forma similar ao algoritmo original de inserção. Porém, ele ficará mais eficiente pois os dados estarão “quase” ordenados.

ShellSort

- Funcionamento:
  - Existem várias formas de se calcular a sequencia de valores de *h*.
  - Em 1973, Knuth propôs a seguinte fórmula (obtida experimentalmente):
    - h(s) = 1, para s = 1
    - h(s) = 3h(s-1)+1, para s > 1
  - Knuth mostrou que essa sequência é difícil de ser batida por mais de 20% de eficiência.

ShellSort

```
void shellSort(int *v, int n) {
	int h=1;
	while (h < n) h = 3*h + 1;

	do {
		h = h/3;
		for (int i=h; i < n; i++) {
			int elemInserir = v[i];
			int j = i-h;
			while( j>=0 && v[j] > elemInserir) {
				v[j+h] = v[j];
				j -= h;
			}
			v[j+h] = elemInserir;
		}
	} while (h > 1);
}
```

ShellSort

- Exemplo de sequencia de iterações:

Arranjo não ordenado

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	**6**	**1**	**11**	**2**	**4**	**8**	**12**	**10**	**5**	**13**	**15**	**7**	**9**	**3**

h=13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	**3**	**1**	**11**	**2**	**4**	**8**	**12**	**10**	**5**	**13**	**15**	**7**	**14**	**6**

h=4

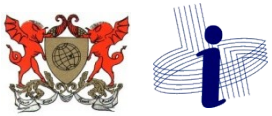
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	**3**	**1**	**11**	**7**	**4**	**6**	**12**	**9**	**5**	**8**	**15**	**10**	**14**	**13**

h=1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

ShellSort

- Considerações:
  - O tempo de execução do método é relativamente baixo (próximo aos melhores métodos, mas um pouco mais lento).
  - O algoritmo é simples de implementar.
  - Até hoje não se conseguiu provar a razão da eficiência do algoritmo. A complexidade dele também não é conhecida.
  - A sua análise envolve problemas matemáticos complexos.
  - Empiricamente, há duas conjecturas sobre a comlexidade do algoritmo utilizando a sequencia proposta por Knuth:
    - O algoritmo é $O(n^{1.25})$
    - O algoritmo é $O(n \log^2 n)$
  - Sabe-se que cada incremento não deve ser múltiplo do anterior.
  - Não é estável.



MergeSort – Ordenação por intercalação

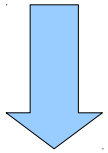
- O algoritmo MergeSort é um método bastante eficiente para ordenação de dados.
- Ele é baseado em uma operação chamada de “intercalação” (*merge*).
- Antes de estudar o algoritmo, vamos estudar um pouco de intercalação.

MergeSort – Ordenação por intercalação

- Intercalação: dados dois vetores ordenados v[p...q-1] e v[q...r-1], o objetivo da intercalação é rearranjar o vetor v[p...r-1] de modo que ele fique ordenado.

p	p+1	q-1	q	q+1	q+2	q+3	r-1
1	8	9	4	5	9	12	15

- Exemplo:



p	p+1	q-1	q	q+1	q+2	q+3	r-1
1	4	5	8	9	9	12	15

MergeSort – Ordenação por intercalação

- Como podemos resolver isso?
- Opção 1: ordenar o vetor v[p...r-1] utilizando o *bubble sort*??

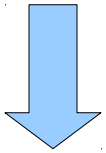
MergeSort – Ordenação por intercalação

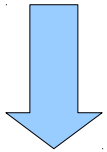
- Como podemos resolver isso?
- Opção 1: ordenar o vetor v[p...r-1] utilizando o *bubble sort*?? → muito lento!
- Opção 2: utilizar um algoritmo um pouco mais elaborado: basta criar um arranjo auxiliar (com tamanho igual a (r-p) e dois cursores (que inicialmente apontam para o início dos dois vetores). A ideia é preencher o arranjo auxiliar com o menor número apontado por um dos cursores e, então, ir andando com o cursor.

MergeSort – Ordenação por intercalação

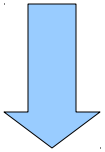
- Exemplo da operação de intercalação:

cursores:





p	p+1	q-1	q	q+1	q+2	q+3	r-1
1	8	9	4	5	9	12	15



Arranjo auxiliar:

p	p+1	q-1	q	q+1	q+2	q+3	r-1

Intercalação:

```
/* Supondo que v[p...q-1] e v[q...r-1] estejam ordenados, intercala os elementos
do arranjo de modo que v[p...r-1] fique ordenado */
void merge(int *v, int p, int q, int r) {
	int tam = r-p;
	int *aux = new int[tam];
	int i = p; //cursor 1
	int j = q; //cursor 2
	int k = 0; //cursor para aux
	while( i< q && j<r) {
		if (v[i] ≤ v[j])
			aux[k++] = v[i++];
		else
			aux[k++] = v[j++];
	}
	while(i<q)
		aux[k++] = v[i++];
	while(j<r)
		aux[k++] = v[j++];
	for(k=0;k<tam;k++)
		v[p+k] = aux[k];

	delete []aux;
}
```

MergeSort – Ordenação por intercalação

- O algoritmo de intercalação apresentado é executado em tempo linear em relação ao número de elementos no arranjo (ou seja, r-p). O uso de memória também é linear.
- O algoritmo MergeSort é baseado na estratégia *Divide & Conquer* funciona da seguinte forma:
  - Divida o vetor original em dois vetores menores (de tamanhos aproximadamente iguais).
  - Ordene os dois vetores menores (de forma recursiva).
  - Intercale os dois vetores menores.

MergeSort

- Implementação:

```
/*
Ordena o vetor v entre as posicoes p e r-1
*/
void mergeSort(int *v, int p, int r) {
	//Se o vetor tiver 1 ou menos elementos então ele ja está ordenado
	if (p<r-1) {
		int meio = (p+r)/2;
		mergeSort(v,p, meio);
		mergeSort(v,meio,r);
		merge(v,p,meio,r); //intercala
	}
}

void mergeSort(int *v, int n) {
	mergeSort(v,0,n);
}
```

MergeSort – Ordenação por intercalação

- Considerações:
  - O *MergeSort* é bastante eficiente: realiza O(n log(n)) comparações.
  - Problema: exige O(n) de espaço extra (porque?): ou seja, não é “in-place”.
  - Existe uma versão *in-place* do método, mas na prática ela normalmente é mais lenta do que a versão “original”.
  - O *MergeSort* é um método estável.

MergeSort – Ordenação por intercalação

- Exercícios:
  - Rastreie os algoritmos *ShellSort* e *MergeSort* supondo que se deseja ordenar o arranjo abaixo:

0	1	2	3	4	5	6
14	**6**	**1**	**11**	**2**	**4**	3

  - Explique porque o *MergeSort* é estável.
  - Mostre que o *ShellSort* não é estável.