Algoritmos e Estrutura de Dados II

Exercício Prático
Shellsort e Mergesort

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Agenda

Roteiro da Aula

Exercício 1

Utilizando o algoritmo de ordenação **Shellsort**, ordene um vetor formado por mil valores. Para cada um, execute os seguintes testes:

- vetor composto por números aleatórios;
- vetor composto por números em ordem crescente;
- vetor composto por números em ordem decrescente.

Exercício 1

Pseudocódigo

Algoritmo 1: Shellsort

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Vetor V ordenado

```
1  início
2      h ← 1
3      enquanto (h < n) faça
4          h ← 3h + 1
5      enquanto (h ≥ 1) faça
6          h ← h/3
7          para (i ← h até n − 1) faça
8              chave ← V[i]
9              j ← i − h
10             enquanto (j ≥ 0 AND V[j] > chave) faça
11                 V[j + h] ← V[j]
12                 j ← j − h
13             V[j + h] ← chave
```

Exercício 1

Solução

```
1  void shellsort(int n, int *vetor) {
2      int h, i, j, chave;
3      h = 1;
4      while (h < n)
5          h = 3 * h + 1;
6      while(h>=1) {
7          h=h/3;
8          for(i = h; i < n; i++) {
9              chave = vetor[i];
10             j = i - h;
11             while(j>=0 && vetor[j]>chave) {
12                 vetor[j+h] = vetor[j];
13                 j = j - h;
14             }
15             vetor[j+h] = chave;
16         }
17     }
18 }
```

Roteiro da Aula

Exercício 2

Tempo Processamento

- Calcule o tempo de processamento para ordenar cada um dos vetores do exercício anterior utilizando o algoritmo **Shellsort**.
- Para isso, utilize a função `clock()`. Acesse esse link para entender o funcionamento da função `clock()`.

Exercício 2

Tempo Processamento

- O código a seguir apresenta um exemplo de ordenação utilizando um vetor com valores aleatórios.
- Modifique esse código a fim de obter o tempo de processamento.

Exercício 2

Solução

```c
#define tamanho_vetor 1000
#define valor_max 1000
#define valor_min 0
int main() {
    int i, *vetor;
    srand(0);
    vetor = malloc(tamanho_vetor*sizeof(int));
    for (i = 0; i < tamanho_vetor; i++)
        vetor[i] = (rand() % valor_max) + valor_min;
    for(i = 0; i < tamanho_vetor; i++)
        printf("%d ", vetor[i]);
    shellsort(tamanho_vetor, vetor);
    printf("\n");
    for(i = 0; i < tamanho_vetor; i++)
        printf("%d ", vetor[i]);
    free (vetor);
    return 0;
}
```

Roteiro da Aula

Exercício 3

Utilizando o algoritmo de ordenação **Mergesort**, ordene um vetor formado por mil valores. Para cada um, execute os seguintes testes:

- vetor composto por números aleatórios;
- vetor composto por números em ordem crescente;
- vetor composto por números em ordem decrescente.

Exercício 3

Pseudo-código

Algoritmo 2: MergesortOrdena

Entrada: Vetor $V[0..n-1]$, tamanho do vetor n.
Saída: Vetor V ordenado

```
1 início
2     Mergesort(V,0,n)
```

Algoritmo 3: Mergesort

Entrada: Vetor $V[i..f-1]$, início i de V, e o final f de V.
Saída: Vetor V ordenado

```
1 início
2     se (i < f − 1) então
3         m ← (i+f)/2
4         Mergesort(V,i,m)
5         Mergesort(V,m,f)
6         Merge(V, i, m, f)
```

Exercício 3

Solução

```
1  void mergesort(int *vetor, int i, int f) {
2      int m;
3      if(i<f-1) {
4          m = (i+f)/2;
5          mergesort(vetor, i, m);
6          mergesort(vetor, m, f);
7          merge(vetor, i, m, f);
8      }
9  }
10 void mergesort_ordena(int n, int *vetor) {
11     mergesort(vetor, 0, n);
12 }
```

Exercício 3

Pseudo-código

Algoritmo 4: Merge

Entrada: Vetor $V[ini..fim-1]$, ini, $meio$, fim.
Saída: Vetor V ordenado

```
1  início
2      // Considere o vetor auxiliar W[ini..fim-1]
3      i ← ini; j ← meio; k ← 0
4      enquanto ( i < meio e j < fim) faça
5          se V[i] ≤ V[j] então
6              W[k] ← V[i]
7              i ← i + 1
8          senão
9              W[k] ← V[j]
10             j ← j + 1;
11         k ← k + 1
12     enquanto (i < meio) faça
13         W[k] ← V[i]
14         i ← i + 1; k ← k + 1
15     enquanto (j < fim) faça
16         W[k] ← V[j]
17         j ← j + 1; k ← k + 1
18     para (i ← ini até fim − 1) faça
19         V[i] ← W[i − ini]
```

Exercício 3

Solução

```
1  void merge(int *vetor, int ini, int meio, int fim) {
2      int i = ini, j = meio, k = 0, *w = malloc(fim*sizeof(int));
3      while(i < meio && j < fim) {
4          if(vetor[i] <= vetor[j]) {
5              w[k] = vetor[i];
6              i++;
7          }
8          else {
9              w[k] = vetor[j];
10             j++;
11         }
12         k++;
13     }
14     while(i < meio) {
15         w[k] = vetor[i];
16         i++;k++;
17     }
18     while(j < fim) {
19         w[k] = vetor[j];
20         j++; k++;
21     }
22     for(i = ini; i < fim; i++) {
23         vetor[i] = w[i-ini];
24     }
25     free(w);
26 }
```

Roteiro da Aula

Exercício 4

Tempo Processamento

Calcule o tempo de processamento para ordenar cada um dos vetores do exercício anterior utilizando o algoritmo **Mergesort**.

Algoritmos e Estrutura de Dados II

Exercício Prático
Shellsort e Mergesort

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia