Estrutura de Dados

José Gildásio Freitas do Ó - 473901

Maio 2023

1. Implemente cada um dos métodos de ordenação a seguir e informe o número de trocas (em função de n) que cada um deles faz:

   (a) Selection Sort — Complexidade: O(n^2)

```
void SelectionSort(int vet[], int p, int r){
int index;
while(p<r){
    index = p;
    for(int i = p+1; i <= r-1; i++){
        if(vet[i] < vet[index]) {
            index = i;
        }
    }
    swap(vet[index], vet[p]);
    p++;
}
}
```

   (b) Bubble Sort — Complexidade: O(n^2)

```
void BoubbleSort(int vet[], int p, int r){
int index;
while(r > p){
    index = p;
    for(int i = p+1; i <= r-1; i++){
        if(vet[i] > vet[index]){
            index = i;
        }
    }
    swap(vet[index],vet[r-1]);
    r--;
}
}
```

(c) Insertion Sort — Complexidade: O(n^2)

```
void InsertionSort(int vet[], int p, int r){
int j;
for(int i = p+1; i <= r-1; i++){
    int x = vet[i];
    j = i-1;
    while( j > p-1 && vet[j] > x) {
        vet[j+1] = vet[j];
        j--;
    }
    vet[j+1] = x;
}
}
```

2. Dados dois vetores ordenados, contendo cada um deles n números inteiros, deseja-se uni-los em um outro vetor maior, contendo 2n números, que também serão armazenados de forma ordenada. Argumente qual será a complexidade de tempo desse processo na estratégia mais eficiente.

- **Resposta:**
  - Algoritmo: ConcatenarVetores(vet1, vet2, n)
  - Entrada: vet1 = vetor1, vet2 = vetor2 e n sendo o tamanho dos 2 vetores
  - Saída: vetor concatenado

```
1- vetor3[n]
2- int i = 0
3- int j = 0
4- int k = 0
5-      enquanto i < n e j < n faça
6-          se vet1[i] ≤ vet2[j]
7-              vetor3[k] = vet1[i]
8-              i++
9-          senão
10-             vetor3[k] = vet2[j]
11-             j++
12-     enquanto i < n faça
13-         vetor3[k] = vet1[j]
14-         i++
15-         k++
16-     enquanto j < n
17-         vetor3[k] = vet2[j]
18-         j++
19-         k++
20- retornar vetor3
```

  - Algoritmo: Ordena(S, p, r)

- Entrada: vetor S, índices p e r
- Saída: vetor S ordenado de p a r -1

```
1- enquanto R > faça
2-     m = p
3-     para i = p + 1 até r - 1 faça
4-         se S[i] > S[m] então
5-             m = i
6-         S[m] ↔ S[r-1]
7-         r = r - 1
```

- Complexidade: O(n+n^2)

3. Apresente o pseudocódigo de uma função que ordena uma lista encadeada. Qual a complexi- dade do seu algoritmo?

- **Resposta:**
  - Algoritmo: swap(no1, no2)
  - Entrada: 2 nós que serão trocados
  - Saída: Returna os nós trocados

```
1-   int aux = nó1 →data
2-   nó1 data = nó2 →data
3-   nó2→data = aux
```

  - Algoritmo: OrdenaçãoBubbleSort(lista)
  - Entrada: lista para ser ordenada
  - Saída: lista ordenada com BubbleSort

```
1-   Boolean teste
2-   Nó* inicial
3-   Nó* final
4-   repetir
5-       teste = false
6-       enquanto inicial ≠ final faça
7-           se inicial →chave > inicial→prox→chave
8-               swap(inicial, inicial→prox)
9-               teste = true
10-              inicial = inicial→prox
11-      final = inicial
12-  enquanto teste for false
13-  retornar lista
```

- **Complexidade:** O(n^2)

4. Apresente o pseudocódigo de uma função RandomizedPartition(V, n). Essa função é similar à função Partition apresentada em sala, porém ela seleciona um elemento aleatório do vetor como pivô e faz o particionamento utilizando esse valor. Para isso, considere uma função Random(n) que retorna um valor aleatório entre 1 e n. Se desejar, você também pode utilizar a função Partition(V, n) vista em sala, que seleciona o primeiro elemento do vetor (V [1]) como pivô para fazer o particionamento.

   - **Resposta:**
     - Algoritmo: Partition(S, p, r)
     - Entrada: vetor S, índices p e r
     - Saída: posição q do pivô em S

```
1- piv = S[p]
2- i = p
3- j = r
4- enquanto i ≤ j faça
5-       enquanto S[i] ≤ piv faça
6-           i++
7-       enquanto S[j] > piv faça
8-           j++
9-       se i < j então
10-          trocar S[i] = S[j]
11- S[p] = S[j]
12- S[j] = piv
13- retorne j
```

   - **Complexidade:** O(n)