Relatorio de Laboratório de Introdução à Ciência da Computação II – Avaliativo 3 Parte 2

Leonardo Kenzo Tanaka e Pedro Teidi de Sá Yamacita

24 de setembro de 2025

1 Descrição do funcionamento do BubbleSort

O **BubbleSort** é um algoritmo de ordenação simples baseado em comparações sucessivas entre pares de elementos adjacentes. A cada iteração, os elementos são comparados e, se estão fora de ordem, trocam-se as posições. Repete-se esse processo até que a lista esteja totalmente ordenada.

Apesar de sua simplicidade, o BubbleSort **não é eficiente para grandes volumes de dados**, pois possui complexidade de tempo quadrática $O(n^2)$ no pior caso.

2 Código de implementação

```c
//Funcao para realizar bubbleSort
void BubbleSort(char array[], int tamanhoArray, int *comparacao, int *trocas){
    char auxiliar;
    *comparacao = 0;
    *trocas = 0;

    //Verifica quantos passos deve realizar
    for(int i = 0; i < tamanhoArray; i++){

        //Itera por todo o array e realiza a troca se necessaria
        for(int j = 0; j < tamanhoArray - i - 1; j++){
            (*comparacao)++;
            if(array[j] > array[j + 1]){
                auxiliar = array[j + 1];
                array[j + 1] = array[j];
                array[j] = auxiliar;
                (*trocas)++;
            }
        }
    }
}
```

3 Análise de desempenho

3.1 Tempo de Execução (Run.Codes)

- Em listas pequenas, o tempo de CPU é praticamente instantâneo (0.0012 s).
- Para listas muito grandes, o tempo cresce rapidamente devido à natureza quadrática do algoritmo ($O(n^2)$), podendo chegar a 2.2482 s no caso teste 5.

3.2 Número de comparações e trocas

- **Comparações:** em média o algoritmo realiza $\frac{n(n-1)}{2}$ comparações, ou seja, da ordem $O(n^2)$.
- **Trocas:** dependem da ordem inicial da lista:
  - Melhor caso $\rightarrow$ 0 trocas.
  - Pior caso $\rightarrow$ número máximo de trocas, próximo de $\frac{n(n-1)}{2}$.
  - Caso médio $\rightarrow$ aproximadamente metade do pior caso.

3.3 Cenários distintos

3.3.1 Melhor caso (lista já ordenada)

- Exemplo: $[2, 3, 4, 5]$.
- O algoritmo percorre toda a lista, realizando comparações, mas sem efetuar trocas.
- Trocas: 0.

3.3.2 Pior caso (lista em ordem inversa)

- Exemplo: $[5, 4, 3, 2]$.
- Cada elemento precisa ser deslocado até sua posição final, resultando no número máximo de trocas.
- Trocas: da ordem de $O(n^2)$.

3.3.3 Caso médio (lista em ordem aleatória)

- Exemplo: $[3, 1, 4, 2]$.
- O desempenho esperado fica entre o melhor e o pior caso.
- Trocas: aproximadamente metade do pior caso.

4 Simplicidade e uso de funções auxiliares

- O código é bem simples, implemetando o BubbleSort de forma tradicional.
- Uma peculiaridade do código é: o número de letras de cada palavra é convertido de ints para chars quando vamos armazená-los no seu devido array de comunidade (USP ou Externa), visto que, em casos grandes, a memória do Run.Códigos não era suficiente para aguentar o programa.
- Foi usada uma função auxiliar contadorLetras, fora do BubbleSort, para ajudar a separar a lógica de contagem do tamanho dos nomes.
- A função BubbleSort usa ponteiros para contar o número de comparações e trocas para ajudar na análise de desempenho.
- Uma possível melhoria seria interromper a execução caso em uma iteração nenhuma troca seja realizada — isso reduziria o custo no **melhor caso** para $O(n)$.