Tecnologia em Análise e Desenvolvimento de SistemasDisciplina de Estrutura de Dados - 3 º Semestre

CASOS DE TESTE PARA ALGORITMOS DE ORDENAÇÃO BUBBLE SORT

Crislaine Cristina Sotello de Souza

Orientador: Marques Moreira de Sousa

Campos do Jordão,

2023

1. Caso de teste para melhor caso

O caso de teste abaixo foi elaborado para a ordenação dos elementos em um vetor de N posições que foi estabelecido no sistema.

Caso de teste para melhor caso – Array já ordenado	
Exigencias especiais	Ter acesso ao um compilador de linguagem C
Item de Teste	Verificação do vetor para saber se os elementos estão ordenados.
Procedimento	1 . Compilar o programa e aguardar.
	2. O sistema solicitará o tamanho do array.
	3.. Digitar o número desejado.
	3. O sistema solicitará os elementos do array.
	4. Digitar os elementos
	5. No menu escolha a opção melhor caso.
	6. O sistema exibirá o array antes, as trocas e depois o array ordenado.
Resultado esperado	1. Exibição do array antes da ordenação
	2. As trocas e o flag.
	3.Exibiçao do array ordenado.

O melhor caso para o algoritmo Bubble Sort ocorre quando a lista já está ordenada. Nesse cenário, nenhuma troca é necessária durante a primeira passagem e, portanto, o algoritmo pode detectar que a lista está ordenada e encerrar precocemente. A análise de complexidade nesse caso é O(n).
Vou explicar o passo a passo para o melhor caso com a lista já ordenada de forma crescente:
Considere a lista de entrada: 17, 20, 26, 31, 44, 54, 55, 77, 93.

Passagem 1:

Comparação: 17 < 20 (sem troca)
Comparação: 20 < 26 (sem troca) ... (sem trocas, pois a lista já está ordenada)
Comparação: 77 < 93 (sem troca)
A lista permanece inalterada após a primeira passagem, e o algoritmo reconhece que nenhuma troca foi feita.

Passagem 2:

Não há comparações nem trocas, pois a lista já está ordenada.
Passagens subsequentes:

Não há comparações nem trocas em passagens subsequentes, pois o algoritmo detecta que a lista já está ordenada e encerra.
A lista permanece ordenada ao longo do processo, e o algoritmo encerra prematuramente após a primeira passagem, resultando em um melhor caso com complexidade de tempo linear O(n). Isso ocorre porque o Bubble Sort otimiza a execução quando a lista já está ordenada.

2. Caso de teste para médio caso

Nesse caso é feito o teste para o caso médio de excussão considerando o algoritmo de ordenação.

Caso de teste para caso médio – Array em ordem aleatória	
Exigencias especiais	Ter acesso ao um compilador de linguagem C
Item de Teste	Verificação do vetor para saber se os elementos estão ordenados.
Procedimento	1 . Compilar o programa e aguardar.
	2. O sistema solicitará o tamanho do array.
	3.. Digitar o número desejado.
	3. O sistema solicitará os elementos do array.
	4. Digitar os elementos
	5. No menu escolha a opção médio caso.
	6. O sistema exibirá o array antes, as trocas e depois o array ordenado.
Resultado esperado	1. Exibição do array antes da ordenação
	2. As trocas e o flag.
	3.Exibiçao do array ordenado.

Para explicar o passo a passo da ordenação de caso médio para o Bubble Sort, considere a lista de entrada desordenada aleatóriamente: [54, 26, 93, 17, 77, 31, 44, 55, 20].

Passo a passo:

Passagem 1:

- Comparação: 54 > 26 (troca: 54 e 26)
- Comparação: 54 < 93 (sem troca)
- Comparação: 93 > 17 (troca: 93 e 17)
- Comparação: 93 > 77 (troca: 93 e 77)
- Comparação: 93 > 31 (troca: 93 e 31)
- Comparação: 93 > 44 (troca: 93 e 44)
- Comparação: 93 > 55 (troca: 93 e 55)
- Comparação: 93 > 20 (troca: 93 e 20)

Lista após a primeira passagem: [26,54,17,77,31,44,55,20,93]

Passagem 2:

- Comparação: 26 < 54 (sem troca)
- Comparação: 54 > 17 (troca: 54 e 17)
- Comparação: 54 < 77 (sem troca)
- Comparação: 77 > 31 (troca: 77 e 31)
- Comparação: 77 > 44 (troca: 77 e 44)
- Comparação: 77 > 55 (troca: 77 e 55)
- Comparação: 77 > 20 (troca: 77 e 20)

Lista após a segunda passagem: [17,26,31,44,54,55,20,77,93]

Passagem 3:

- Comparação: 17 < 26 (sem troca)
- Comparação: 26 < 31 (sem troca)

- Comparação: 31 < 44 (sem troca)
- Comparação: 44 < 54 (sem troca)
- Comparação: 54 < 55 (sem troca)
- Comparação: 55 > 20 (troca: 55 e 20)

Lista após a terceira passagem: [17,26,31,44,54,20,55,77,93]

Passagem 4:

- Comparação: 17 < 26 (sem troca)
- Comparação: 26 < 31 (sem troca)
- Comparação: 31 < 44 (sem troca)
- Comparação: 44 < 54 (sem troca)
- Comparação: 54 > 20 (troca: 54 e 20)

Lista após a quarta passagem: [17,26,31,44,20,54,55,77,93]

Passagens subsequentes:

As passagens subsequentes não fazem trocas, pois os elementos já estão em suas posições corretas.
O algoritmo continua passando pela lista até que nenhuma troca seja feita em uma passagem completa, indicando que a lista está ordenada. No caso médio, algumas trocas são feitas, mas a lista ainda não está totalmente desordenada, o que leva a uma complexidade de tempo média de O (n^2).

3. Caso de teste para pior caso

O teste realiza o pior caso de excussão considerando o algoritmo de ordenação inversa.

Caso de teste para pior - Array em ordem inversa	
Exigencias especiais	Ter acesso ao um compilador de linguagem C
Item de Teste	Verificação do vetor para saber se os elementos estão ordenados.
Procedimento	1 . Compilar o programa e aguardar.
	2. O sistema solicitará o tamanho do array.
	3.. Digitar o número desejado.
	3. O sistema solicitará os elementos do array.
	4. Digitar os elementos
	5. No menu escolha a opção pior caso.
	6. O sistema exibirá o array antes, as trocas e depois o array ordenado.
Resultado esperado	1. Exibição do array antes da ordenação
	2. As trocas e o flag.
	3.Exibiçao do array ordenado.

O pior caso para o algoritmo Bubble Sort ocorre quando a lista está totalmente desordenada. Vamos explicar o passo a passo para o pior caso usando a lista de entrada desordenada: [93, 71, 55, 54, 44, 31, 26].

Passagem 1:

- Comparação: 93 > 71 (troca: 93 e 71)
- Comparação: 93 > 55 (troca: 93 e 55)
- Comparação: 93 > 54 (troca: 93 e 54)
- Comparação: 93 > 44 (troca: 93 e 44)
- Comparação: 93 > 31 (troca: 93 e 31)
- Comparação: 93 > 26 (troca: 93 e 26)

Lista após a primeira passagem: [71,55,54,44,31,26,93]

Passagem 2:

- Comparação: 71 > 55 (troca: 71 e 55)
- Comparação: 71 > 54 (troca: 71 e 54)
- Comparação: 71 > 44 (troca: 71 e 44)
- Comparação: 71 > 31 (troca: 71 e 31)
- Comparação: 71 > 26 (troca: 71 e 26)

Lista após a segunda passagem: [55,54,44,31,26,71,93]

Passagem 3:

- Comparação: 55 > 54 (troca: 55 e 54)
- Comparação: 55 > 44 (troca: 55 e 44)
- Comparação: 55 > 31 (troca: 55 e 31)
- Comparação: 55 > 26 (troca: 55 e 26)

Lista após a terceira passagem: [54,44,31,26,55,71,93]

Passagem 4:

- Comparação: 54 > 44 (troca: 54 e 44)
- Comparação: 54 > 31 (troca: 54 e 31)
- Comparação: 54 > 26 (troca: 54 e 26)

Lista após a quarta passagem: [44,31,26,54,55,71,93]

Passagem 5:

- Comparação: 44 > 31 (troca: 44 e 31)
- Comparação: 44 > 26 (troca: 44 e 26)

Lista após a quinta passagem: [31,26,44,54,55,71,93]

Passagem 6:

- Comparação: 31 > 26 (troca: 31 e 26)

Lista após a sexta passagem: [26,31,44,54,55,71,93]

Passagens subsequentes:

As passagens subsequentes continuam fazendo trocas até que a lista esteja totalmente ordenada.

O algoritmo continua passando pela lista até que nenhuma troca seja feita em uma passagem completa, indicando que a lista está ordenada. No pior caso, muitas trocas são feitas, levando a uma complexidade de tempo quadrática O(n2).

Implementação do código com todos os casos.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Função para trocar dois elementos
void trocar(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// Função para ordenar um array usando o algoritmo Bubble Sort
void bubbleSort(int array[], int quantidade, int *trocas) {
   *trocas = 0;
   int i, j;
   for (i = 0; i < quantidade - 1; i++) {
      for (j = 0; j < quantidade - i - 1; j++) {
         // Compara e troca os elementos se estiverem fora de ordem
         if (array[j] > array[j + 1]) {
            trocar(&array[j], &array[j + 1]);
            (*trocas) += 1;

            // Imprime os detalhes da troca
            printf("Iteração %d - Troca %d <--> %d\n", i + 1, array[j], array[j + 1]);
         }
      }
   }
}

// Função para imprimir um array
void imprimirArray(int array[], int quantidade) {
   for (int i = 0; i < quantidade; i++) {
      printf("%d ", array[i]);
   }
   printf("\n");
}

int main() {
   int quantidade;
   int opcao;

   do {
      // Recebe a quantidade de elementos do usuário
      printf("\nDigite a quantidade de elementos: ");
      scanf("%d", &quantidade);

      // Verifica se a quantidade é válida
      if (quantidade <= 0) {
         printf("A quantidade deve ser maior que zero. Tente novamente.\n");
         continue;
      }

      // Cria um array para os casos
```

```
int frascos[quantidade];

printf("\nEscolha uma opção:\n");
printf("1. Melhor Caso\n");
printf("2. Médio Caso\n");
printf("3. Pior Caso\n");
printf("Opção: ");
scanf("%d", &opcao);

switch (opcao) {
   case 1:
      // Melhor Caso: Array já ordenado
      for (int i = 0; i < quantidade; i++) {
         frascos[i] = i + 1;
      }

      printf("\nArray antes de ordenar (Melhor Caso):\n");
      imprimirArray(frascos, quantidade);

      // Chama a função Bubble Sort para ordenar o array
      bubbleSort(frascos, quantidade, &opcao);

      // Imprime os resultados
      printf("\nArray ordenado em ordem crescente:\n");
      imprimirArray(frascos, quantidade);
      printf("Total de Trocas: %d\n", opcao);
      break;

   case 2:
      // Médio Caso: Array com elementos aleatórios
      srand(time(NULL));
      for (int i = 0; i < quantidade; i++) {
         frascos[i] = rand() % 1000;
      }

      printf("\nArray antes de ordenar (Médio Caso):\n");
      imprimirArray(frascos, quantidade);

      // Chama a função Bubble Sort para ordenar o array
      bubbleSort(frascos, quantidade, &opcao);

      // Imprime os resultados
      printf("\nArray ordenado em ordem crescente:\n");
      imprimirArray(frascos, quantidade);
      printf("Total de Trocas: %d\n", opcao);
      break;

   case 3:
      // Pior Caso: Array inversamente ordenado
      for (int i = 0; i < quantidade; i++) {
         frascos[i] = quantidade - i;
      }

      printf("\nArray antes de ordenar (Pior Caso):\n");
      imprimirArray(frascos, quantidade);

      // Chama a função Bubble Sort para ordenar o array
```

```
            bubbleSort(frascos, quantidade, &opcao);

            // Imprime os resultados
            printf("\nArray ordenado em ordem crescente:\n");
            imprimirArray(frascos, quantidade);
            printf("Total de Trocas: %d\n", opcao);
            break;

        default:
            printf("Opção inválida. Tente novamente.\n");
            break;
    }

    // Pergunta se deseja continuar
    printf("\nDeseja continuar? (1 para Sim, 0 para Não): ");
    scanf("%d", &opcao);

  } while (opcao == 1);

  return 0;
}
```