UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 10

Nome: Geraldo Arthur Detomi

1.1) Faça um programa para ler um valor N e em seguida N inteiros, armazenando esses inteiros em um vetor. Em seguida, ordenar esses valores utilizando os 3 metodos de ordenacao vistos: SelectionSort, InsertionSort e BubbleSort.

roteiro_10/1-1.c

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

#define QTD_ALGORITMOS_TESTE 3

void imprimeVetor(int *v, int n) {
  int i, prim = 1;
  printf("[");
  for (i = 0; i < n; i++)
    if (prim) {
      printf("%d", v[i]);
      prim = 0;
    } else
      printf(", %d", v[i]);
  printf("]\n");
}

void troca(int *a, int *b) {
  int aux = *a;
  *a = *b;
  *b = aux;
}

void selectionSort(int *v, int n, int *comp, int *mov) {
  int i, j, menor;
  for (i = 0; i < n - 1; i++) {
    menor = i;
    for (j = i + 1; j < n; j++) {
      (*comp)++;
      if (v[j] < v[menor])
        menor = j;
    }
    if (i != menor) {
      troca(&v[i], &v[menor]);
      (*mov)++;
    }
  }
}

void insertionSort(int *v, int n, int *comp, int *mov) {
  int i, j, atual;
  for (i = 1; i < n; i++) {
    atual = v[i];
    (*comp)++;
    for (j = i; (j > 0) && (atual < v[j - 1]); j--) {
      v[j] = v[j - 1];
      (*comp)++;
      (*mov)++;
    }
    v[j] = atual;
  }
}

void BubbleSort(int *v, int n, int *comp, int *mov) {
  int i, j;
  for (i = 0; i < n - 1; i++)
    for (j = 0; j < n - i - 1; j++) {
      (*comp)++;
      if (v[j] > v[j + 1]) {
        troca(&v[j], &v[j + 1]);
        (*mov)++;
      }
    }
}
```

```c

int *copiaVetor(int *v, int n) {
  int i;
  int *v2;
  v2 = (int *)malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    v2[i] = v[i];
  return v2;
}

void liberar_vetor(int **v) {
  free(*v);
  *v = NULL;
}

enum algoritmos_para_teste { SELECION_SORT = 0, INSERTION_SORT, BUBLE_SORT };

int main() {
  int n;

  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);

  int *array_original = (int *)malloc(n * sizeof(int));

  for (int i = 0; i < n; i++) {
    scanf("%d", &array_original[i]);
  }

  for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
    int *array_copia = copiaVetor(array_original, n);

    int comp, mov;

    switch (i) {
    case SELECION_SORT:
      printf("Ordenando com Selecion Sort:\n");
      selectionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case INSERTION_SORT:
      printf("Ordenando com Insertion Sort:\n");
      insertionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case BUBLE_SORT:
      printf("Ordenando com Buble Sort:\n");
      BubbleSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    }

    liberar_vetor(&array_copia);
  }

  liberar_vetor(&array_original);
  return 0;
}
```

Saída do terminal:

```
arthurdetomi at arthurdetomi-System-Product-Name
∴ ./1-1.out
Digite o tamanho do vetor:4
7
8
2
5
Ordenando com Selecion Sort:
[2, 5, 7, 8]
Ordenando com Insertion Sort:
[2, 5, 7, 8]
Ordenando com Buble Sort:
[2, 5, 7, 8]
```

1-2) Modifique os algoritmos de ordenacao anteriores, para que ordenem de forma decrescente os numeros (do maior para o menor) e teste os novos metodos.

roteiro_10/1-2.c

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

```

```c
#define QTD_ALGORITMOS_TESTE 3

void imprimeVetor(int *v, int n) {
  int i, prim = 1;
  printf("[");
  for (i = 0; i < n; i++)
    if (prim) {
      printf("%d", v[i]);
      prim = 0;
    } else
      printf(", %d", v[i]);
  printf("]\n");
}

void troca(int *a, int *b) {
  int aux = *a;
  *a = *b;
  *b = aux;
}

void selectionSort(int *v, int n, int *comp, int *mov) {
  int i, j, maior;
  for (i = 0; i < n - 1; i++) {
    maior = i;
    for (j = i + 1; j < n; j++) {
      (*comp)++;
      if (v[j] > v[maior])
        maior = j;
    }
    if (i != maior) {
      troca(&v[i], &v[maior]);
      (*mov)++;
    }
  }
}

void insertionSort(int *v, int n, int *comp, int *mov) {
  int i, j, atual;
  for (i = 1; i < n; i++) {
    atual = v[i];
    (*comp)++;
    for (j = i; (j > 0) && (atual > v[j - 1]); j--) {
      v[j] = v[j - 1];
      (*comp)++;
      (*mov)++;
    }
    v[j] = atual;
  }
}

void BubbleSort(int *v, int n, int *comp, int *mov) {
  int i, j;
  for (i = 0; i < n - 1; i++)
    for (j = 0; j < n - i - 1; j++) {
      (*comp)++;
      if (v[j] < v[j + 1]) {
        troca(&v[j], &v[j + 1]);
        (*mov)++;
      }
    }
}

int *copiaVetor(int *v, int n) {
  int i;
  int *v2;
  v2 = (int *)malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    v2[i] = v[i];
  return v2;
}

void liberar_vetor(int **v) {
  free(*v);
  *v = NULL;
}

enum algoritmos_para_teste { SELECION_SORT = 0, INSERTION_SORT, BUBLE_SORT };

int main() {
  int n;

```

```c
  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);

  int *array_original = (int *)malloc(n * sizeof(int));

  for (int i = 0; i < n; i++) {
    scanf("%d", &array_original[i]);
  }

  printf("Agora a ordenação será decrescente:\n");
  for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
    int *array_copia = copiaVetor(array_original, n);

    int comp, mov;

    switch (i) {
    case SELECION_SORT:
      printf("Ordenando com Selecion Sort:\n");
      selectionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case INSERTION_SORT:
      printf("Ordenando com Insertion Sort:\n");
      insertionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case BUBLE_SORT:
      printf("Ordenando com Buble Sort:\n");
      BubbleSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    }

    liberar_vetor(&array_copia);
  }

  liberar_vetor(&array_original);
  return 0;
}
```

Saída do terminal:

```
arthurdetomi at arthurdetomi-System-Product-Name
.:: ./1-2.out
Digite o tamanho do vetor:5
4
10
3
6
2
Agora a ordenação será decrescente:
Ordenando com Selecion Sort:
[10, 6, 4, 3, 2]
Ordenando com Insertion Sort:
[10, 6, 4, 3, 2]
Ordenando com Buble Sort:
[10, 6, 4, 3, 2]
```

1-3) Considerando os 3 metodos vistos, utilize um programa para ordenar grande quantidade de valores e verifique o aumento do numero de comparacoes, movimentacoes e tempo de execucao conforme se aumenta o volume de dados processados

roteiro_10/tempo.h

```c
#ifndef TEMPO_H
#define TEMPO_H

#include <stdio.h>
#include <sys/resource.h>
#include <sys/time.h>

// Estrutura para armazenar os tempos de execução
typedef struct Temporizador {
  struct timeval start_tv, end_tv;
  struct rusage start_usage, end_usage;
} Temporizador;

// Inicia a contagem do temporizador
void iniciarTemporizador(Temporizador *t);

// Finaliza a contagem do temporizador
void finalizarTemporizador(Temporizador *t);

// Calcula o tempo real decorrido em segundos
```

```c
double calcularTempoReal(Temporizador *t);

// Calcula o tempo de CPU em modo usuário em segundos
double calcularTempoUsuario(Temporizador *t);

// Calcula o tempo de CPU em modo sistema/kernel em segundos
double calcularTempoSistema(Temporizador *t);

// Imprime todos os tempos medidos de forma formatada
void imprimirTempos(Temporizador *t);

#endif
```

roteiro_10/tempo.c

```c
#include "tempo.h"

// Inicia a medição do tempo, armazenando os valores atuais
void iniciarTemporizador(Temporizador *t) {
  gettimeofday(&t->start_tv, NULL);
  getrusage(RUSAGE_SELF, &t->start_usage);
}

// Finaliza a medição do tempo, armazenando os valores finais
void finalizarTemporizador(Temporizador *t) {
  gettimeofday(&t->end_tv, NULL);
  getrusage(RUSAGE_SELF, &t->end_usage);
}

double calcularTempo(struct timeval inicio, struct timeval fim) {
  time_t seg = fim.tv_sec - inicio.tv_sec;
  suseconds_t microseg = fim.tv_usec - inicio.tv_usec;

  // Ajusta caso microsegundos do fim sejam menores que os do início
  if (microseg < 0) {
    seg -= 1;
    microseg += 1000000;
  }

  return (double)seg + (double)microseg / 1e6;
}

// Calcula diferença entre tempos reais (start e end) em segundos
double calcularTempoReal(Temporizador *t) {
  return calcularTempo(t->start_tv, t->end_tv);
}

// Calcula diferença entre tempos de sistema (start e end) em segundos
double calcularTempoSistema(Temporizador *t) {
  return calcularTempo(t->start_usage.ru_stime, t->end_usage.ru_stime);
}

// Imprime todos os tempos de execução de forma legível
void imprimirTempos(Temporizador *t) {
  printf("Tempo de execução:\n");
  printf("Tempo Real: %.8f segundos\n", calcularTempoReal(t));
  printf("Tempo Sistema: %.8f segundos\n", calcularTempoSistema(t));
}
```

roteiro_10/1-3.c

```c
#include "tempo.h"
#include <stdio.h>
#include <stdlib.h>

#define QTD_ALGORITMOS_TESTE 3

void imprimeVetor(int *v, int n) {
  int i, prim = 1;
  printf("[");
  for (i = 0; i < n; i++)
    if (prim) {
      printf("%d", v[i]);
      prim = 0;
    } else
      printf(", %d", v[i]);
  printf("]\n");
}

void troca(int *a, int *b) {
  int aux = *a;
  *a = *b;
```

```c
  *b = aux;
}

void selectionSort(int *v, int n, int *comp, int *mov) {
  int i, j, menor;
  for (i = 0; i < n - 1; i++) {
    menor = i;
    for (j = i + 1; j < n; j++) {
      (*comp)++;
      if (v[j] < v[menor])
        menor = j;
    }
    if (i != menor) {
      troca(&v[i], &v[menor]);
      (*mov)++;
    }
  }
}

void insertionSort(int *v, int n, int *comp, int *mov) {
  int i, j, atual;
  for (i = 1; i < n; i++) {
    atual = v[i];
    (*comp)++;
    for (j = i; (j > 0) && (atual < v[j - 1]); j--) {
      v[j] = v[j - 1];
      (*comp)++;
      (*mov)++;
    }
    v[j] = atual;
  }
}

void BubbleSort(int *v, int n, int *comp, int *mov) {
  int i, j;
  for (i = 0; i < n - 1; i++)
    for (j = 0; j < n - i - 1; j++) {
      (*comp)++;
      if (v[j] > v[j + 1]) {
        troca(&v[j], &v[j + 1]);
        (*mov)++;
      }
    }
}

int *copiaVetor(int *v, int n) {
  int i;
  int *v2;
  v2 = (int *)malloc(n * sizeof(int));
  for (i = 0; i < n; i++)
    v2[i] = v[i];
  return v2;
}

void liberar_vetor(int **v) {
  free(*v);
  *v = NULL;
}

enum algoritmos_para_teste { SELECION_SORT = 0, INSERTION_SORT, BUBLE_SORT };

int main() {
  int n;

  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);

  int *array_original = (int *)malloc(n * sizeof(int));

  for (int i = 0; i < n; i++) {
    scanf("%d", &array_original[i]);
  }

  for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
    int *array_copia = copiaVetor(array_original, n);

    int comp = 0, mov = 0;

    Temporizador tempo_teste;
    iniciarTemporizador(&tempo_teste);

```

```c
    switch (i) {
    case SELECION_SORT:
      printf("Ordenando com Selecion Sort:\n");
      selectionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case INSERTION_SORT:
      printf("Ordenando com Insertion Sort:\n");
      insertionSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    case BUBLE_SORT:
      printf("Ordenando com Buble Sort:\n");
      BubbleSort(array_copia, n, &comp, &mov);
      imprimeVetor(array_copia, n);
      break;
    }

    printf("Quantidade de movimentações realizadas: %d\n", mov);
    printf("Quantidade de comparações realizadas: %d\n", comp);

    finalizarTemporizador(&tempo_teste);

    imprimirTempos(&tempo_teste);

    liberar_vetor(&array_copia);

    printf("\n\n");
  }

  liberar_vetor(&array_original);
  return 0;
}
```

Saída do terminal:

```
┌─arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_10 on mainxxx 25-06-15
└─.:: ./1-3.out < input/100-misturado.txt
Digite o tamanho do vetor:Ordenando com Selecion Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68
88, 89, 94, 95, 96, 97, 100, 105, 106, 113, 113, 115, 115, 116, 117, 121, 123, 124, 127, 129, 129, 129, 134, 135, 139, 142, 142, 144,
 163, 164, 165, 167, 167, 167, 172, 173, 175, 178, 178, 180, 184, 185, 189, 192, 193, 193, 195, 197, 200, 200]
Quantidade de movimentações realizadas: 97
Quantidade de comparações realizadas: 4950
Tempo de execução:
Tempo Real: 0.00004000 segundos
Tempo Sistema: 0.00004000 segundos


Ordenando com Insertion Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68
88, 89, 94, 95, 96, 97, 100, 105, 106, 113, 113, 115, 115, 116, 117, 121, 123, 124, 127, 129, 129, 129, 134, 135, 139, 142, 142, 144,
 163, 164, 165, 167, 167, 167, 172, 173, 175, 178, 178, 180, 184, 185, 189, 192, 193, 193, 195, 197, 200, 200]
Quantidade de movimentações realizadas: 2426
Quantidade de comparações realizadas: 2525
Tempo de execução:
Tempo Real: 0.00002900 segundos
Tempo Sistema: 0.00002900 segundos


Ordenando com Buble Sort:
[2, 3, 4, 4, 9, 10, 12, 12, 14, 17, 18, 21, 25, 25, 27, 28, 29, 29, 33, 34, 38, 38, 39, 40, 42, 43, 43, 51, 52, 58, 62, 66, 67, 67, 68
88, 89, 94, 95, 96, 97, 100, 105, 106, 113, 113, 115, 115, 116, 117, 121, 123, 124, 127, 129, 129, 129, 134, 135, 139, 142, 142, 144,
 163, 164, 165, 167, 167, 167, 172, 173, 175, 178, 178, 180, 184, 185, 189, 192, 193, 193, 195, 197, 200, 200]
Quantidade de movimentações realizadas: 2426
Quantidade de comparações realizadas: 4950
Tempo de execução:
Tempo Real: 0.00004400 segundos
Tempo Sistema: 0.00004400 segundos
```

1-4) Modifique as funcoes que implementam o metodo Selecao e Insercao para fazer a ordenacao de um vetor de struct do tipo Pessoa,

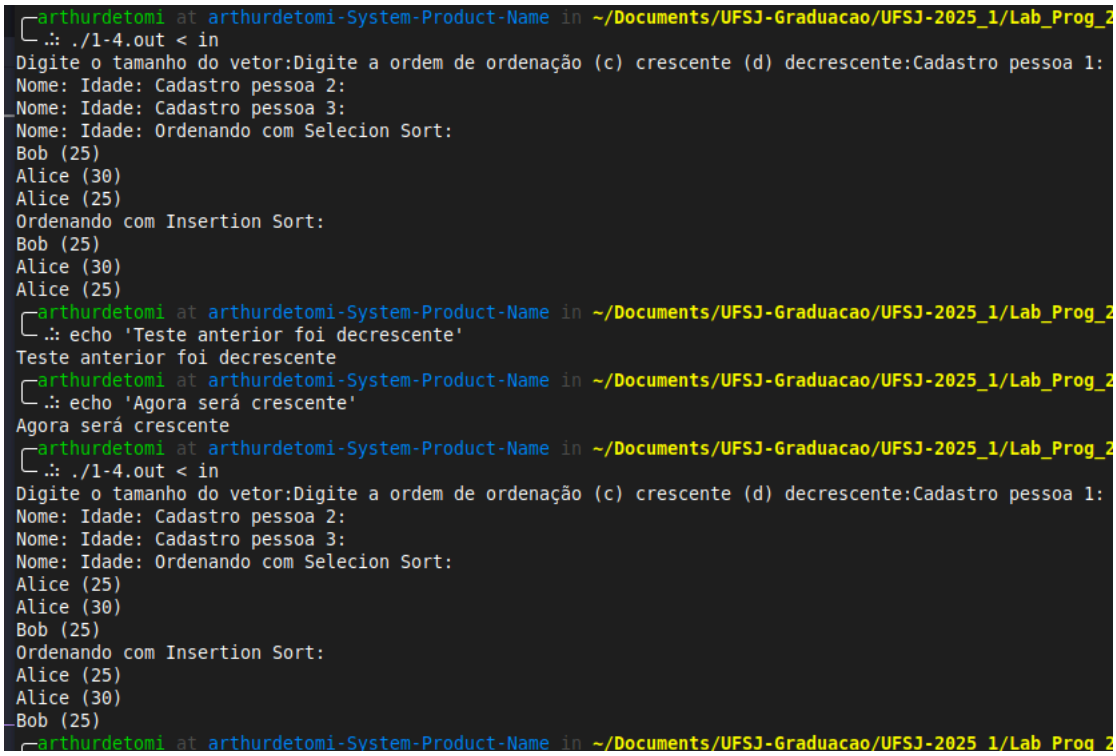
roteiro_10/1-4.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define QTD_ALGORITMOS_TESTE 2

enum algoritmos_para_teste { SELECION_SORT = 0, INSERTION_SORT, BUBLE_SORT };

enum ordens_ordenacao { CRESCENTE = 'c', DECRESCENTE = 'd' };
```

```c

typedef struct Pessoa {
  char nome[50];
  int idade;
} Pessoa;

void imprimeVetor(Pessoa *v, int n) {
  for (int i = 0; i < n; i++) {
    printf("%s (%d)\n", v[i].nome, v[i].idade);
  }
}

void troca(Pessoa *a, Pessoa *b) {
  Pessoa aux = *a;
  *a = *b;
  *b = aux;
}

int comparaCrescente(Pessoa p1, Pessoa p2) {
  int comparacao = strcmp(p1.nome, p2.nome);
  if (comparacao == 0) {
    return p1.idade - p2.idade;
  }
  return comparacao;
}

int comparaDecrescente(Pessoa p1, Pessoa p2) {
  int comparacao = strcmp(p2.nome, p1.nome);
  if (comparacao == 0) {
    return p2.idade - p1.idade;
  }
  return comparacao;
}

int comparaPorOrdem(Pessoa p1, Pessoa p2, char ordem) {
  if (ordem == CRESCENTE) {
    return comparaCrescente(p1, p2);
  }
  return comparaDecrescente(p1, p2);
}

void selectionSort(Pessoa *v, int n, int *comp, int *mov, char ordem) {
  int i, j, menor;
  for (i = 0; i < n - 1; i++) {
    menor = i;
    for (j = i + 1; j < n; j++) {
      (*comp)++;
      if (comparaPorOrdem(v[j], v[menor], ordem) < 0) {
        menor = j;
      }
    }
    if (i != menor) {
      troca(&v[i], &v[menor]);
      (*mov)++;
    }
  }
}

void insertionSort(Pessoa *v, int n, int *comp, int *mov, char ordem) {
  int i, j;
  for (i = 1; i < n; i++) {
    Pessoa atual = v[i];
    (*comp)++;
    for (j = i; (j > 0) && comparaPorOrdem(atual, v[j - 1], ordem) < 0; j--) {
      v[j] = v[j - 1];
      (*comp)++;
      (*mov)++;
    }
    v[j] = atual;
  }
}

Pessoa *copiaVetor(Pessoa *v, int n) {
  int i;
  Pessoa *v2;
  v2 = (Pessoa *)malloc(n * sizeof(Pessoa));
  for (i = 0; i < n; i++) {
    strcpy(v2[i].nome, v[i].nome);
    v2[i].idade = v[i].idade;
  }
  return v2;
```

```c
}

void liberar_vetor(Pessoa **v) {
  free(*v);
  *v = NULL;
}

int main() {
  int n;

  printf("Digite o tamanho do vetor:");
  scanf("%d", &n);

  char ordem;
  printf("Digite a ordem de ordenação (c) crescente (d) decrescente:");
  scanf(" %c", &ordem);

  if (ordem != CRESCENTE && ordem != DECRESCENTE) {
    printf("Ordem inserida invalida\n");
    exit(1);
  }

  Pessoa *array_original = (Pessoa *)malloc(n * sizeof(Pessoa));

  for (int i = 0; i < n; i++) {
    printf("Cadastro pessoa %d:\n", i + 1);
    printf("Nome: ");
    scanf("%s", array_original[i].nome);
    printf("Idade: ");
    scanf("%d", &array_original[i].idade);
  }

  for (int i = 0; i < QTD_ALGORITMOS_TESTE; i++) {
    Pessoa *array_copia = copiaVetor(array_original, n);

    int comp = 0, mov = 0;

    switch (i) {
    case SELECION_SORT:
      printf("Ordenando com Selecion Sort:\n");
      selectionSort(array_copia, n, &comp, &mov, ordem);
      imprimeVetor(array_copia, n);
      break;
    case INSERTION_SORT:
      printf("Ordenando com Insertion Sort:\n");
      insertionSort(array_copia, n, &comp, &mov, ordem);
      imprimeVetor(array_copia, n);
      break;
    }

    liberar_vetor(&array_copia);
  }

  liberar_vetor(&array_original);
  return 0;
}
```

Saída do terminal:


arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2
.:: ./1-4.out < in
Digite o tamanho do vetor:Digite a ordem de ordenação (c) crescente (d) decrescente:Cadastro pessoa 1:
Nome: Idade: Cadastro pessoa 2:
Nome: Idade: Cadastro pessoa 3:
Nome: Idade: Ordenando com Selecion Sort:
Bob (25)
Alice (30)
Alice (25)
Ordenando com Insertion Sort:
Bob (25)
Alice (30)
Alice (25)
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2
.:: echo 'Teste anterior foi decrescente'
Teste anterior foi decrescente
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2
.:: echo 'Agora será crescente'
Agora será crescente
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2
.:: ./1-4.out < in
Digite o tamanho do vetor:Digite a ordem de ordenação (c) crescente (d) decrescente:Cadastro pessoa 1:
Nome: Idade: Cadastro pessoa 2:
Nome: Idade: Cadastro pessoa 3:
Nome: Idade: Ordenando com Selecion Sort:
Alice (25)
Alice (30)
Bob (25)
Ordenando com Insertion Sort:
Alice (25)
Alice (30)
Bob (25)
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2