Projeto prático 01
Implementação de algoritmos de ordenação em assembly MIPS

Arquitetura de computadores e sistemas operacionais 2023-2

Autores:
Artur Henrique Teixeira do Amaral (DRE: 122032113)
Tales Coutinho Moreira (DRE: 119047549)

1) Introdução

Tomamos a decisão de implementar os seguintes algoritmos de ordenação: **bubble, insertion e selection sort**. A escolha foi feita baseada na simplicidade dos algoritmos frente ao nosso pouco conhecimento do conjunto de instruções MIPS em linguagem assembly.

2) Implementação dos algoritmos

Primeiramente, seguem considerações gerais sobre as implementações. Todos os códigos em assembly MIPS, inicialmente, declaram e atribuem valores a algumas strings utilitárias, como textos indicativos, espaço e nova linha.

Os vetores a serem ordenados são igualmente declarados na primeira seção, bem com seus respectivos tamanhos, que são 10 elementos nas três implementações.

Os códigos seguem uma estrutura geral, que basicamente implementa uma função *main*, uma função *sort* e uma função utilitária *printArray*. A *main* inicia a execução do programa, chama a *printArray* para escrever os elementos do vetor na saída do sistema, chama a função sort para executar a ordenação do vetor e novamente chama a *printArray* para exibir o vetor ordenado.

Na implementação específica do BubbleSort existe a implementação de um método *swap* e uma variável *swap_count*, que não foram transcritas para a implementação do Insertion Sort e do Selection Sort por limitações de tempo. Optamos por entregar a versão funcional destes 2 algoritmos a tempo, em vez de refatorar o código e perder o prazo.

a) Bubble sort

Código base:

```c
#include <stdio.h>

int a[] = {9, 8, 6, 5, 1, 2, 3, 4};
int n = 10;
```

```c
void swap(int a[], int j, int iMin) {
	int temp = a[j];
	a[j] = a[iMin];
	a[iMin] = temp;
}

void sort(int a[], int n) {
	int swapCount = 0;

	for (int j = 0; j < n - 1; j++) {
		int iMin = j;

		for (int i = j + 1; i < n; i++)
			if (a[i] < a[iMin])
				iMin = i;

		if (iMin != j) {
			swap(a, j, iMin);
			swapCount++;
		}
	}
	printf("Contagem de Trocas: %d\n", swapCount);
}

void printArray(int a[], int n) {
	for (int i = 0; i < n; i++) {
		printf("%d ", a[i]);
	}
	printf("\n");
}

int main() {
	printf("Array não ordenado: ");
	printArray(a, n);
	sort(a, n);
	printf("Array ordenado: ");
	printArray(a, n);
	return 0;
}
```

Mapeamento de variáveis

- `$t0 = j`
- `$t1 = i`

- $t2 = n-1
- $t3 = endereço de int[] a
- $t4 = iMin
- $t5 = n
- $s0 = swapCount

b) Insertion sort

Código base:

Por simplicidade, daqui em diante mostrarei apenas as funções de ordenação, uma vez que as funções *main()* e *printArray()* e *swap()* são essencialmente iguais.

```
void insertionSort(int arr[], int n)
{
	int i, key, j;
	for (i = 1; i < n; i++) {
		key = arr[i];
		j = i - 1;

		while (j >= 0 && arr[j] > key) {
			arr[j + 1] = arr[j];
			j = j - 1;
		}
		arr[j + 1] = key;
	}
}
```

Mapeamento de variáveis:

- $t0 -> endereço do array a
- $t1 -> n-1 (último índice de a)
- $t2 -> i
- $t3 -> j
- $t4 -> key
- $t5 -> Usado como variável auxiliar
- $t6 -> Usado como variável auxiliar

c) Selection sort

Código base:

```
void selectionSort(int arr[], int n)
{
	int i, j, min_idx;

	for (i = 0; i < n-1; i++)
	{
		min_idx = i;
		for (j = i+1; j < n; j++)
		{
			if (arr[j] < arr[min_idx])
			min_idx = j;
		}

		if(min_idx != i)
			swap(&arr[min_idx], &arr[i]);
	}
}
```

Mapeamento de variáveis:

- $t0 -> endereço do array a
- $t1 -> n-1 (último índice de a)
- $t2 -> i
- $t3 -> j
- $t4 -> min_idx
- $t5 -> n
- $t6 -> Usada como variável auxiliar
- $t7 -> Usada como variável auxiliar