Relatório 04 - Ordenando brinquedos

SCC0220 - Laboratório de Introdução à Ciência da Computação II (2022)

Prof. Diego Furtado Silva

Departamento de Ciências da Computação (SCC)

Instituto de Ciências Matemáticas e de Computação (ICMC)

Universidade de São Paulo

Discentes:

- Felipi Yuri Santos (11917292)
- Vicenzo D'Arezzo Zilio (13671790)

Sumário

- III - Comparação dos tempos de execução

I - Introdução

O problema é esboçado quando precisamos ordenar diferentes brinquedos (o elemento do nosso vetor) com cores diferentes (por ordem alfabética), tamanhos diferentes (em ordem crescente) e notas (em ordem decrescente).
Não só precisamos fazer isso utilizando um método de ordenação. Temos que usar o BubbleSort, o InsertionSort, o MergeSort e o Quicksort.

II - Implementações dos algoritmos de ordenação

Um adendo: para os próximos três algoritmos será utilizado uma função chamada ehMenor(a, b) que recebe dois brinquedos ‘a’ e ‘b’ e retorna *true* se a deve vir numa posição anterior à b e *false* caso contrário de acordo com as regras de ordenação do exercício.

```
bool ehMenor(brinquedo a, brinquedo b){
    //Sabemos que os brinquedos são da mesma cor, portanto queremos analisar seus tamanhos.

    if(a.tamanho == b.tamanho){
        //para brinquedos de mesmo tamanhos, o desempate se dá pela nota:
        if( a.nota ≥ b.nota){
            return true;
        }else{
            return false;
        }
    }else{
        if( a.tamanho < b.tamanho){
            return true;
        }else{
            return false;
        }
    }
}
```

BubbleSort

```c
void bubblesort(brinquedo *v, int tamanho){
    brinquedo aux;
    for(int i = 0; i < tamanho; i++){
        for(int j = 0; j < tamanho - 1; j++){
            if(v[j].cor > v[j+1].cor){
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }else if (v[j].cor == v[j+1].cor){
                if(!ehMenor(v[j], v[j+1])){
                    aux = v[j];
                    v[j] = v[j+1];
                    v[j+1] = aux;
                }
            }
        }
    }
}
```

InsertionSort

```c
void insertionSort (brinquedo *v, int tamanho){
    brinquedo eleito;
    int j;

    for(j = 1; j < tamanho; j++){
        eleito = v[j];

        int i = j - 1;
        while(i >= 0 && v[i].cor >= eleito.cor){
            if(v[i].cor == eleito.cor){
                if(!ehMenor(eleito, v[i]))
                    break;
            }
            v[i+1] = v[i];
            i--;
        }

        v[i + 1] = eleito;
    }
}
```

MergeSort

```
void mergeSort(brinquedo *v, int ini, int fim){

    if(fim <= ini){
        return;
    }

    int meio = (int)(ini + fim)/2;

    mergeSort(v, ini, meio);
    mergeSort(v, meio + 1, fim);
    ordenacao_interacalando(ini, fim, v);
}
```

```
void ordenacao_interacalando(int inicio, int fim, brinquedo *v){
    int meio = (int) (inicio + fim)/2;
    int contador1 = inicio, contador2 = meio + 1;
    int contador_vetor = 0;
    brinquedo vetor_aux[fim - inicio + 1];

    while(contador1 <= meio && contador2 <= fim){
        if(v[contador1].cor < v[contador2].cor){
            vetor_aux[contador_vetor] = v[contador1];
            contador1++;
        }else if(v[contador1].cor == v[contador2].cor){
            if(!ehMenor(v[contador1], v[contador2])){
                vetor_aux[contador_vetor] = v[contador2];
                contador2++;
            }else{
                vetor_aux[contador_vetor] = v[contador1];
                contador1++;
            }
        }else{
            vetor_aux[contador_vetor] = v[contador2];
            contador2++;
        }
        contador_vetor++;
    }
```

```
while(contador1 ≤ meio) {
    vetor_aux[contador_vetor] = v[contador1];
    contador1++;
    contador_vetor++;
}
while(contador2 ≤ fim) {
    vetor_aux[contador_vetor] = v[contador2];
    contador2++;
    contador_vetor++;
}

for(int i = inicio, j = 0; i ≤ fim; i++, j++){
    v[i] = vetor_aux[j];
}
```

QuickSort

Para a função QuickSort, o algoritmo de desempate é ligeiramente diferente:

```
bool desemp(brinquedo a, brinquedo b){
    if(a.cor ≠ b.cor)
        return a.cor < b.cor;

    if(a.tamanho ≠ b.tamanho)
        return a.tamanho < b.tamanho;

    if(a.nota ≠ b.nota)
        return a.nota ≥ b.nota;

    return false;
}
```

```
void quickSort(brinquedo* vetor, int inicio, int fim){
    if (fim < inicio)
        return;

    brinquedo aux;

    int pivo = inicio;
    int i = inicio+1;
    int j = fim;

    while (i <= j){
        while (i <= fim && desemp(vetor[i], vetor[pivo])) i++;
        while (desemp(vetor[pivo], vetor[j])) j--;

        if (j > i){
            aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
        }
    }

    pivo = j;
    aux = vetor[pivo];
    vetor[pivo] = vetor[inicio];
    vetor[inicio] = aux;

    quickSort(vetor, inicio, pivo-1);
    quickSort(vetor, pivo+1, fim);
}
```

III - Comparação dos tempos de execução

Algoritmo	Caso 1 (s)	Caso 2 (s)	Caso 3 (s)	Caso 4 (s)

BubbleSort	0,001	0,005	0,507	1m30,143
InsertionSort	0,001	0,002	0,055	5,2
MergeSort	0,001	0,001	0,005	0,044
QuickSort	0,001	0,075	0,079	0,112