1.3/busca.h

```c
#ifndef BUSCA_H
#define BUSCA_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define MAX 200

typedef struct{

    char nome[MAX];
    int matricula;
    double nota1, nota2, nota3;

} Aluno;

//Medidas de Complexidade
int comp; //Num. de comparacoes

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;
}

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

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 preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void troca_nome(Aluno* a, Aluno *b){
    Aluno aux = *a;
    *a = *b;
    *b = aux;
}

```

```c
int buscaSequencial_nome(Aluno *v, int n, char *elem){
    int i;
    for(i=0; i<n; i++){
        comp++;
        if(strcmp(v[i].nome, elem) == 0)
            return i; //Elemento encontrado
    }
    return -1; //Elemento nao encontrado
}

int particao_nome(Aluno *v, int ini, int fim){
    int i = ini, j = fim;
    Aluno pivo = v[(ini+fim)/2];
    while (1) {
        while(strcmp(v[i].nome, pivo.nome) < 0){ i++; } //procura algum >= pivo do lado esquerdo
        while(strcmp(v[j].nome, pivo.nome) > 0){ j--; } //procura algum <= pivo do lado direito

        if(i<j){
            troca_nome(&v[i], &v[j]); //troca os elementos encontrados
            i++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort_nome(Aluno *v, int ini, int fim){
    if(ini < fim ){
        int q = particao_nome(v, ini, fim);
        QuickSort_nome(v, ini, q);
        QuickSort_nome(v, q+1, fim);
    }
}

int rec_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
    if(ini > fim) return -1;
    int meio = (ini + fim)/2;
    comp++;
    if(strcmp(v[meio].nome, elem) == 0)
        return meio;
    else
        if(strcmp(v[meio].nome, elem) > 0)
            return rec_buscaBinaria_nome(v, ini, meio-1, elem);
        else
            return rec_buscaBinaria_nome(v, meio+1, fim, elem);
}

int it_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
    int meio;
    while(ini <= fim){
        meio = (ini + fim)/2;
        comp++;
        if(strcmp(v[meio].nome, elem) == 0) return meio;
        else
            if(strcmp(v[meio].nome, elem) > 0)
                fim = meio-1;
            else
                ini = meio+1;
```

```
116     }
117     return -1;
118 }
119 
120 void troca_matricula(Aluno* a, Aluno *b){
121     Aluno aux = *a;
122     *a = *b;
123     *b = aux;
124 }
125 
126 int buscaSequencial_matricula(Aluno *v, int n, int elem){
127     int i;
128     for(i=0; i<n; i++){
129         comp++;
130         if(v[i].matricula == elem)
131             return i; //Elemento encontrado
132     }
133     return -1; //Elemento encontrado
134 }
135 
136 int particao_matricula(Aluno *v, int ini, int fim){
137     int i = ini, j = fim;
138     Aluno pivo = v[(ini+fim)/2];
139     while (1) {
140         while(v[i].matricula < pivo.matricula){ i++; } //procura algum >= pivo do
    lado esquerdo
141         while(v[j].matricula > pivo.matricula){ j--; } //procura algum <= pivo do
    lado direito
142 
143         if(i<j){
144             troca_matricula(&v[i], &v[j]); //troca os elementos encontrados
145             i++;
146             j--;
147         }else
148             return j; //retorna o local onde foi feita a particao
149     }
150 }
151 
152 void QuickSort_matricula(Aluno *v, int ini, int fim){
153     if(ini < fim ){
154         int q = particao_matricula(v, ini, fim);
155         QuickSort_matricula(v, ini, q);
156         QuickSort_matricula(v, q+1, fim);
157     }
158 }
159 
160 int rec_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
161     if(ini > fim) return -1;
162     int meio = (ini + fim)/2;
163     comp++;
164     if(v[meio].matricula == elem)
165         return meio;
166     else
167         if(elem < v[meio].matricula)
168             return rec_buscaBinaria_matricula(v, ini, meio-1, elem);
169         else
170             return rec_buscaBinaria_matricula(v, meio+1, fim, elem);
171 }
172 
173 int it_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
```

```
174     int meio;
175     while(ini <= fim){
176         meio = (ini + fim)/2;
177         comp++;
178         if(elem == v[meio].matricula) return meio;
179         else
180             if(elem < v[meio].matricula)
181                 fim = meio-1;
182             else
183                 ini = meio+1;
184     }
185     return -1;
186 }
187 
188 // void troca(int* a, int *b){
189 //  int aux = *a;
190 //  *a = *b;
191 //  *b = aux;
192 // }
193 
194 // int buscaSequencial(int *v, int n, int elem){
195 //     int i;
196 //     for(i=0; i<n; i++){
197 //         comp++;
198 //         if(v[i] == elem)
199 //             return i; //Elemento encontrado
200 //     }
201 //     return -1; //Elemento encontrado
202 // }
203 
204 // int particao(int *v, int ini, int fim){
205 //     int i = ini, j = fim;
206 //     int pivo = v[(ini+fim)/2];
207 //     while (1) {
208 //         while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
209 //         while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito
210 
211 //         if(i<j){
212 //             troca(&v[i], &v[j]); //troca os elementos encontrados
213 //             i++;
214 //             j--;
215 //         }else
216 //             return j; //retorna o local onde foi feita a particao
217 //     }
218 // }
219 
220 // void QuickSort(int *v, int ini, int fim){
221 //     if(ini < fim ){
222 //         int q = particao(v, ini, fim);
223 //         QuickSort(v, ini, q);
224 //         QuickSort(v, q+1, fim);
225 //     }
226 // }
227 
228 // int rec_buscaBinaria(int *v, int ini, int fim, int elem){
229 //     if(ini > fim) return -1;
230 //     int meio = (ini + fim)/2;
231 //     comp++;
232 //     if(v[meio] == elem)
233 //         return meio;
```

```
234 //     else
235 //         if(elem < v[meio])
236 //             return rec_buscaBinaria(v, ini, meio-1, elem);
237 //         else
238 //             return rec_buscaBinaria(v, meio+1, fim, elem);
239 // }
240 
241 // int it_buscaBinaria(int *v, int ini, int fim, int elem){
242 //     int meio;
243 //     while(ini <= fim){
244 //         meio = (ini + fim)/2;
245 //         comp++;
246 //         if(elem == v[meio]) return meio;
247 //         else
248 //             if(elem < v[meio])
249 //                 fim = meio-1;
250 //             else
251 //                 ini = meio+1;
252 //     }
253 //     return -1;
254 // }
255 
256 #endif
```