```
Aluno: Bruno Adriano Menegotto
#include<cstdlib>
#include<time.h>
#include<iostream>
#include<stdio.h>

using namespace std;

const int TAM = 15000;
const int rep = 1000;
int lista[TAM], num_elementos_vetor;
double operacoes_seq = 0, operacoes_bin = 0,  operacoes_arvore_bin = 0,
operacoes_arvore_avl = 0;

struct no{
    struct no *dir;
    struct no *esq;
    int dado;
    int bal;
};

struct no *raiz = NULL;
struct no *raiz2 = NULL;

void Busca_Sequencial(int chave){
    int i;
    for(i = 0; i < num_elementos_vetor && lista[i] <= chave; i++){
        operacoes_seq++;
        if(chave == lista[i]){
            return;
        }
    }
}

void Busca_Binaria(int chave){
    int inicio = 0;
    int fim = TAM-1;
    int meio;

    while (inicio <= fim){
        meio = (inicio + fim)/2;
        operacoes_bin++;
        if (chave == lista[meio]){
            return;
        }
        if (chave == lista[inicio]){
            return;
        }
        if (chave == lista[fim]){
            return;
        }
        if (chave < lista[meio])
            fim = meio - 1;
        else
            inicio = meio + 1;
    }
}

void Insere_Arvore_Binaria(int valor){
    struct no *atual;
    struct no *anterior;
    struct no *novo;

    novo = new(struct no);
    novo -> esq = NULL;
    novo -> dir = NULL;
```

```
    novo -> dado = valor;

    if(raiz == NULL){
        raiz = novo;
        return;
    }
    atual = raiz;
    while(atual != NULL){
        anterior = atual;
        if(atual -> dado > valor){
            atual = atual -> esq;
        }
        else{
            atual = atual -> dir;
        }
    }
    if(anterior -> dado > valor){
        anterior -> esq = novo;
    }
    else{
        anterior -> dir = novo;
    }
}

void Busca_Arvore_Binaria(int chave){
    struct no *atual;
    atual = raiz;

    while(atual != NULL){
        operacoes_arvore_bin++;
        if(chave == atual -> dado){
            return;
        }
        if(chave < atual -> dado){
            atual = atual -> esq;
        }
        else{
            atual = atual -> dir;
        }
    }
    if(atual == NULL){
        return;
    }
}

void Esquerda(struct no *p){
    struct no *q, *hold;

    q = p -> dir;
    hold = q -> esq;
    q -> esq = p;
    p -> dir = hold;

}

void Direita(struct no *p){
    struct no *q, *hold;

    q = p -> esq;
    hold = q -> dir;
    q -> dir = p;
    p -> esq = hold;
}

struct no *cria_no(int valor){
    struct no *aux = new (struct no);
```

```
    aux -> dado = valor;
    aux -> dir = NULL;
    aux -> esq = NULL;
    aux -> bal = 0;
    return aux;
};


void Insere_Arvore_AVL(int valor){
    struct no *pp = NULL, *p = raiz2, *pajovem = NULL, *ajovem = raiz2, *q, *filho;
    int imbal;

    if (p == NULL){
        raiz2 = cria_no(valor);
        return;
    }
    while (p != NULL){
        if (valor < p -> dado)
            q = p -> esq;
        else
            q = p -> dir;
        if (q != NULL)
            if (q -> bal != 0){
                pajovem = p;
                ajovem = q;
            }
        pp = p;
        p = q;
    }

    q = cria_no(valor);

    if (valor < pp -> dado)
        pp -> esq = q;
    else
        pp -> dir = q;

    if (valor < ajovem -> dado)
        filho = ajovem -> esq;
    else
        filho = ajovem -> dir;
    p = filho;

    while (p != q){
        if (valor < p -> dado){
            p -> bal = 1;
            p = p -> esq;
        }
        else{
            p -> bal = - 1;
            p = p -> dir;
        }
    }

    if (valor < ajovem -> dado)
        imbal = 1;
    else
        imbal = -1;

    if (ajovem -> bal == 0){
        ajovem -> bal = imbal;
        return;
    }

    if (ajovem -> bal != imbal){
        ajovem -> bal = 0;
```

```
        return;
    }

    if (filho -> bal == imbal){
        p = filho;
        if (imbal == 1)
            Direita(ajovem);
        else
            Esquerda(ajovem);
        ajovem -> bal =0 ;
        filho -> bal = 0;
    }
    else{
        if (imbal == 1){
            p = filho -> dir;
            Esquerda(filho);
            ajovem -> esq = p;
            Direita(ajovem);
        }
        else{
            p = filho -> esq;
            Direita(filho);
            ajovem -> dir = p;
            Esquerda(ajovem);
        }

        if (p -> bal == 0){
            ajovem -> bal = 0;
            filho -> bal = 0;
        }
        else{
            if (p -> bal == imbal){
                ajovem -> bal = - imbal;
                filho -> bal = 0;
            }
            else{
                ajovem -> bal = 0;
                filho -> bal = imbal;
            }
        }
        p -> bal = 0;
    }

    if (pajovem == NULL)
        raiz2 = p;
    else if (ajovem == pajovem -> dir)
        pajovem -> dir = p;
    else
        pajovem -> esq = p;
    return;
}

void Busca_Arvore_AVL(int chave){
    struct no *atual;
    atual = raiz2;

    if(atual == NULL){
        return;
    }

    while(atual != NULL){
        operacoes_arvore_avl++;
        if(chave == atual -> dado){
            return;
        }
        if(chave < atual -> dado){
```

```
            atual = atual -> esq;
        }
        else{
            atual = atual -> dir;
        }
        if(atual == NULL){
            return;
        }
    }
}

int partition(int p, int r){
    int aux, piv, i, j;
    piv = lista[p];
    i = p - 1;
    j = r + 1;
    while(true){
        do{
            j = j - 1;
        }
        while(lista[j] > piv);
        do{
            i = i + 1;
        }
        while(lista[i] < piv);
        if(i < j){
            aux = lista[i];
            lista[i] = lista[j];
            lista[j] = aux;
        }
        else{
            return j;
        }
    }
}

void QuickSort(int p, int r){
    int q;
    if(p < r){
        q = partition(p, r);
        QuickSort(p, q);
        QuickSort(q + 1, r);
    }
}

void Repeticoes(){
    for(int i = 0; i < rep; i++){
        Busca_Sequencial(rand());
    }
    for(int i = 0; i < rep; i++){
        Busca_Binaria(rand());
    }
    for(int i = 0; i < rep; i++){
        Busca_Arvore_Binaria(rand());
    }
    for(int i = 0; i < rep; i++){
        Busca_Arvore_AVL(rand());
    }
}

void Gerar_Vetor(){
    for(int i = 0; i < TAM; i++){
        lista[i] = rand();
        num_elementos_vetor++;
    }
}
```

```
int main(){
    int i;
    srand(time(NULL));
    Gerar_Vetor();
    for(i = 0; i < TAM; i++)
        Insere_Arvore_Binaria(lista[i]);
    for(i = 0; i < TAM; i++)
        Insere_Arvore_AVL(lista[i]);
    QuickSort(0, TAM - 1);
    Repeticoes();
    cout << "\n A media das operacoes na Busca Sequencial eh: " << operacoes_seq / rep <<
" operacoes" << endl;
    cout << "\n A media das operacoes na Busca Binaria eh: " << operacoes_bin / rep << "
operacoes" << endl;
    cout << "\n A media das operacoes na Busca em Arvore Binaria eh: " <<
operacoes_arvore_bin / rep << " operacoes"<< endl;
    cout << "\n A media das operacoes na Busca em Arvore AVl eh: " <<
operacoes_arvore_avl / rep << " operacoes";
}
```