```
#include <bits/stdc++.h>

using namespace std;

int CHECKING_COUNT = 0;

int binary_search(vector<int> list, int element) {
	int left = 0;
	int right = list.size() - 1;

	while (left <= right) {
	int mid = left + (right - left) / 2;
	CHECKING_COUNT = CHECKING_COUNT + 1;
	// Verifica se o elemento alvo está presente no meio
	if (list[mid] == element) {
	return mid; // Retorna o índice do elemento
	}

	// Se o elemento alvo for maior, ignore a metade esquerda
	if (list[mid] < element) {
	left = mid + 1;
	}
	// Se o elemento alvo for menor, ignore a metade direita
	else {
	right = mid - 1;
	}
	}

	return -1;
}

int main() {
	int n; // quantidade de elementos do vetor
	int x; // número procurado
	vector <int> vec; // lista de entrada

	cin >> n;
	cin >> x;

	for(int i = 0; i < n; i++){
	int aux;
	cin >> aux;
	vec.push_back(aux);
	}

	// execução cronometrada da busca
	auto start = std::chrono::high_resolution_clock::now();
	int result = binary_search(vec, x);
```

```
	auto end = chrono::high_resolution_clock::now();

	chrono::duration<double> duration = end - start;

	// exibição dos resultados
	cout << "result: " << result << '\n';
	cout << "checking count: " << CHECKING_COUNT << '\n';
	cout << "elapsed time: " << duration.count() << '\n';

	return 0;
}
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

int CHECKING_COUNT = 0;

int linear_search(vector<int> list, int element) {
	for(int i = 0; i < list.size(); i++){
	CHECKING_COUNT++;
	if(list[i] == element){
	return i;
	}
	}

	return -1;
}

int main() {
	int n; // quantidade de elementos do vetor
	int x; // número procurado
	vector <int> vec; // lista de entrada

	// leitura
	cin >> n >> x;
	for(int i = 0; i < n; i++){
	int aux;
	cin >> aux;
	vec.push_back(aux);
	}

	// execução cronometrada da busca
	auto start = std::chrono::high_resolution_clock::now();
	int result = linear_search(vec, x);
	auto end = chrono::high_resolution_clock::now();

	chrono::duration<double> duration = end - start;
```

```
	// exibição dos resultados
	cout << "result: " << result << '\n';
	cout << "checking count: " << CHECKING_COUNT << '\n';
	cout << "elapsed time: " << duration.count() << '\n';

	return 0;
}
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

long long COUNT = 0;

long long exp(int expoent) {
	COUNT++;
	if(expoent == 0)
	return 1;

	return 2 * exp(expoent - 1);
}

int main() {
	int n;
	cin >> n;

	COUNT = 0;
	// execução cronometrada da busca

	for(int i = 0; i <= n; i++){
	auto start = std::chrono::high_resolution_clock::now();
	int result = exp(i);
	auto end = chrono::high_resolution_clock::now();
	chrono::duration<double> duration = end - start;

	// exibição dos resultados
	cout << "n: " << i << '\n';
	cout << "2^n: " << result << '\n';
	cout << "count: " << COUNT << '\n';
	cout << "elapsed time: " << duration.count() << "\n\n";
	}

	return 0;
}
#include <bits/stdc++.h>
#include <chrono>
```

```
using namespace std;

long long COUNT = 0;

long long fib(long long n) {
	COUNT++;
	if(n == 0 || n == 1)
	return 1;

	return fib(n-1) + fib(n-2);
}

int main() {
	int n;
	cin >> n;

	COUNT = 0;
	// execução cronometrada da busca

	for(int i = 0; i <= n; i++){
	auto start = std::chrono::high_resolution_clock::now();
	long long result = fib(i);
	auto end = chrono::high_resolution_clock::now();
	chrono::duration<double> duration = end - start;

	// exibição dos resultados
	cout << "n: " << i << '\n';
	cout << "f_n: " << result << '\n';
	cout << "count: " << COUNT << '\n';
	cout << "elapsed time: " << duration.count() << "\n\n";
	}

	return 0;
}
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

long long COUNT = 0;

long long non_recursive_fib(int n) {
	COUNT++;
	long long a = 0, b = 1, c;

	if (n == 0)
	return a;
```

```
        for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
        }
        return b;
}

int main() {
        int n;
        cin >> n;

        COUNT = 0;
        // execução cronometrada da busca

        for(int i = 0; i <= n; i++){
        auto start = std::chrono::high_resolution_clock::now();
        long long result = non_recursive_fib(i);
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<double> duration = end - start;

        // exibição dos resultados
        cout << "n: " << i << '\n';
        cout << "f_n: " << result << '\n';
        cout << "count: " << COUNT << '\n';
        cout << "elapsed time: " << duration.count() << "\n\n";
        }

        return 0;
}
#include <bits/stdc++.h>
#include <chrono>

using namespace std;

int CHECKING_COUNT = 0;

bool is_sorted(vector<int> vec) {
        for(int i = 0; i < vec.size() - 1; i++) {
        CHECKING_COUNT++;
        if(vec[i+1] < vec[i])
        return false;
        }
        return true;
}

int main() {
        int n;
```

```
	vector<int> vec;
	cin >> n;

	for(int i = 0; i < n; i++) {
	int aux;
	cin >> aux;
	vec.push_back(aux);
	}

	// execução cronometrada da busca
	auto start = std::chrono::high_resolution_clock::now();
	int result = is_sorted(vec);
	auto end = chrono::high_resolution_clock::now();

	chrono::duration<double> duration = end - start;

	// exibição dos resultados
	cout << "result: " << result << '\n';
	cout << "checking count: " << CHECKING_COUNT << '\n';
	cout << "elapsed time: " << duration.count() << '\n';

	return 0;
}
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

int COMPARISONS_COUNT = 0;
int SWAPS_COUNT = 0;

// troca de dois elementos
void swap(int &a, int &b) {
	SWAPS_COUNT++;
	int temp = a;
	a = b;
	b = temp;
}

// ordena um vetor pelo bubble sort
void bubble_sort(vector<int> &arr) {
	int n = arr.size();
	for (int i = 0; i < n-1; i++) {
	// Últimos i elementos já estão na ordem correta
	for (int j = 0; j < n-i-1; j++) {
	COMPARISONS_COUNT++;
	if (arr[j] > arr[j+1]) {
```

```
                swap(arr[j], arr[j+1]);
        }
        }
        }
}

// exibe o conteúdo do vetor
void print_vector(const vector<int> &arr) {
        for (int val : arr) {
        cout << val << " ";
        }
        cout << endl;
}

int main() {
        int n;
        vector<int> vec;
        cin >> n;

        for(int i = 0; i < n; i++) {
        int aux;
        cin >> aux;
        vec.push_back(aux);
        }

        // execução cronometrada da ordenação
        auto start = std::chrono::high_resolution_clock::now();
        bubble_sort(vec);
        auto end = chrono::high_resolution_clock::now();

        chrono::duration<double> duration = end - start;

        // exibição dos resultados
        cout << "comparisons count: " << COMPARISONS_COUNT << '\n';
        cout << "swaps count: " << SWAPS_COUNT << '\n';
        cout << "elapsed time: " << duration.count() << '\n';
        cout << "result: " << '\n';
        print_vector(vec);



        return 0;
}
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;
```

```
int COMPARISONS_COUNT = 0;
int SWAPS_COUNT = 0;

// troca de dois elementos
void swap(int &a, int &b) {
	SWAPS_COUNT++;
	int temp = a;
	a = b;
	b = temp;
}

// ordena o vetor pelo insertion sort
void insertion_sort(vector<int> &arr) {
	int n = arr.size();
	for (int i = 1; i < n; i++) {
	int key = arr[i];
	int j = i - 1;

	// Move os elementos de arr[0..i-1], que são maiores que a chave,
	// para uma posição à frente da sua posição atual
	while (j >= 0) {
	COMPARISONS_COUNT++;
	if (arr[j] > key) {
		arr[j + 1] = arr[j];
		j--;
		SWAPS_COUNT++;
	} else {
		break;
	}
	}
	arr[j + 1] = key;
	}
}

// exibe o conteúdo do vetor
void print_vector(const vector<int> &arr) {
	for (int val : arr) {
	cout << val << " ";
	}
	cout << endl;
}

int main() {
	int n;
	vector<int> vec;
	cin >> n;
```

```
	for(int i = 0; i < n; i++) {
	int aux;
	cin >> aux;
	vec.push_back(aux);
	}

	// execução cronometrada da ordenação
	auto start = std::chrono::high_resolution_clock::now();
	insertion_sort(vec);
	auto end = chrono::high_resolution_clock::now();

	chrono::duration<double> duration = end - start;

	// exibição dos resultados
	cout << "comparisons count: " << COMPARISONS_COUNT << '\n';
	cout << "swaps count: " << SWAPS_COUNT << '\n';
	cout << "elapsed time: " << duration.count() << '\n';
	cout << "result: " << '\n';
	print_vector(vec);

	return 0;
}
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

long long int COMPARISONS_COUNT = 0;
long long int SWAPS_COUNT = 0;

// troca de dois elementos
void swap(int &a, int &b) {
	SWAPS_COUNT++;
	int temp = a;
	a = b;
	b = temp;
}

// função para mesclar dois subvetores
void merge(vector<int> &arr, int left, int mid, int right) {
	int n1 = mid - left + 1;
	int n2 = right - mid;

	// vetores temporários
	vector<int> L(n1), R(n2);

	// copia os dados para os vetores temporários L[] e R[]
```

```
        for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

        // mescla os vetores temporários de volta em arr[left..right]
        int i = 0; // índice inicial do primeiro subvetor
        int j = 0; // índice inicial do segundo subvetor
        int k = left; // índice inicial do subvetor mesclado

        while (i < n1 && j < n2) {
        COMPARISONS_COUNT++;
        if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
        } else {
        arr[k] = R[j];
        j++;
        }
        k++;
        }

        // copia os elementos restantes de L[], se houver
        while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
        }

        // copia os elementos restantes de R[], se houver
        while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
        }
}

// ordena o vetor pelo merge sort
void merge_sort(vector<int> &arr, int left, int right) {
        if (left >= right)
        return;

        int mid = left + (right - left) / 2;

        // Ordena a primeira e a segunda metade
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
```

```
        // Mescla as duas metades ordenadas
        merge(arr, left, mid, right);
}

// exibe o conteúdo do vetor
void print_vector(const vector<int> &arr) {
        for (int val : arr) {
        cout << val << " ";
        }
        cout << endl;
}

int main() {
        int n;
        vector<int> vec;
        cin >> n;

        for(int i = 0; i < n; i++) {
        int aux;
        cin >> aux;
        vec.push_back(aux);
        }

        // execução cronometrada da ordenação
        auto start = std::chrono::high_resolution_clock::now();
        merge_sort(vec, 0, vec.size() - 1);
        auto end = chrono::high_resolution_clock::now();

        chrono::duration<double> duration = end - start;

        // exibição dos resultados
        cout << "comparisons count: " << COMPARISONS_COUNT << '\n';
        cout << "swaps count: " << SWAPS_COUNT << '\n';
        cout << "elapsed time: " << duration.count() << '\n';
        cout << "result: " << '\n';
        print_vector(vec);

        return 0;
}
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

int COMPARISONS_COUNT = 0;
int SWAPS_COUNT = 0;
```

```
// troca de dois elementos
void swap(int &a, int &b) {
	SWAPS_COUNT++;
	int temp = a;
	a = b;
	b = temp;
}

// Função para particionar o vetor usando o último elemento como pivô
int partition(vector<int> &arr, int low, int high) {
	int pivot = arr[high];  // pivô
	int i = (low - 1);  // Índice do menor elemento

	for (int j = low; j < high; j++) {
	COMPARISONS_COUNT++;
	// Se o elemento atual é menor ou igual ao pivô
	if (arr[j] <= pivot) {
	i++;	// incrementa o índice do menor elemento
	swap(arr[i], arr[j]);
	}
	}
	swap(arr[i + 1], arr[high]);
	return (i + 1);
}

// ordena o vetor pelo quick sort
void quick_sort(vector<int> &arr, int low, int high) {
	if (low < high) {
	// pi é o índice de particionamento, arr[pi] está no lugar certo
	int pi = partition(arr, low, high);

	// Ordena os elementos antes e depois da partição
	quick_sort(arr, low, pi - 1);
	quick_sort(arr, pi + 1, high);
	}
}

// exibe o conteúdo do vetor
void print_vector(const vector<int> &arr) {
	for (int val : arr) {
	cout << val << " ";
	}
	cout << endl;
}

int main() {
	int n;
	vector<int> vec;
```

```
    cin >> n;

    for(int i = 0; i < n; i++) {
    int aux;
    cin >> aux;
    vec.push_back(aux);
    }

    // execução cronometrada da ordenação
    auto start = std::chrono::high_resolution_clock::now();
    quick_sort(vec, 0, vec.size() - 1);
    auto end = chrono::high_resolution_clock::now();

    chrono::duration<double> duration = end - start;

    // exibição dos resultados
    cout << "comparisons count: " << COMPARISONS_COUNT << '\n';
    cout << "swaps count: " << SWAPS_COUNT << '\n';
    cout << "elapsed time: " << duration.count() << '\n';
    cout << "result: " << '\n';
    print_vector(vec);

    return 0;
}
```