Cấu trúc dữ liệu và giải thuật

Nhóm 15

TỔNG HỢP TÌM KIẾM

Họ và tên	**Mã sinh viên**
Vũ Công Tuấn Dương	B22DCKH024

Ngày nộp : 09/03/2024

1 TỔNG HỢP TÌM KIẾM

1. Cài đặt thuật toán tìm kiếm đã học: 3

2. Khảo sát thời gian thực thi các thuật toán lần lượt với các giá trị n khác nhau của cùng 1 dãy số: 3

3. Thời gian thực thi của các thuật toán với cùng 1 giá trị n(rất lớn, > 10000) với cùng 1 dãy số có sự khác biệt do ảnh hưởng bởi nhiều yếu tố: may mắn(nếu trong trường hợp tốt nhất), độ phức tạp thuật toán về thời gian-nguyên nhân chủ yếu. Cụ thể:

   - Tìm kiếm tuyến tính: O(n)
   - Tìm kiếm nhị phân: O(log(n))
   - Tìm kiếm tam phân: O(log(n))
   - Tìm kiếm nội suy: O(log(log(n))
   - Tìm kiếm Fibonacci: O(log(x)) với x là số cần tìm

     Nhìn vào các thuật toán ta dễ thấy chỉ có duy nhất tìm kiếm tuần tự là có độ phức tạp là O(n), 4 thuật toán còn lại có độ phức tạp nhỏ hơn, trong đó tìm kiếm nội suy có độ phức tạp thấp nhất nhưng không thấp hơn quá nhiều so với 3 thuật toán còn lại
   - Vẽ đồ thị thể hiện thời gian thực thi của mỗi thuật toán phụ thuộc vào n: Hình 1. Ta có thể thấy tìm kiếm tuần tự có thời gian có thời gian chạy nhiều nhất- gấp nhiều lần 4 thuật toán còn lại, do đó ta quan tâm hơn tới 4 thậut toán còn lại: Hình 2. Nhìn vào đồ thị ta thấy tìm kiếm nhị phân có độ lệch lớn giữa các n khác nhau, tìm kiếm fibonacci và tìm kiếm tam phân có hình dạng khá tương đồng nhau. Tìm kiếm nội suy có thời gian chạy ít nhất, phù hợp với lí thuyết
   - Chương trình được viết như sau:

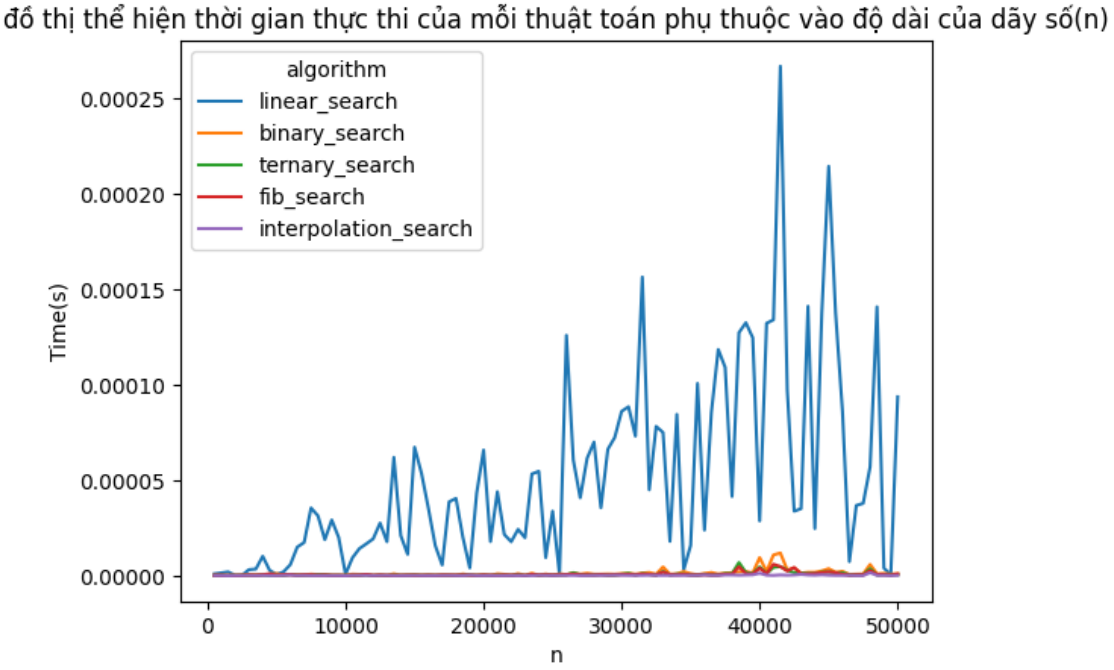
```
#include <bits/stdc++.h>
#include <chrono>
#include <fstream>
#include <random>
using namespace std;
long long fibo[92] = {0};
int linearSearch(int *a, int n, int x);
int binarySearch(int *a, int n, int x, int lo, int hi);
int ternarySearch(int *a, int n, int x, int lo, int hi);
int interpolationSearch(int *a, int n, int x, int lo, int hi);
void init();
int fibSearch(int *a, int n, int x);
void test_search_func(ofstream& csvfile, const string& func_name, vector

int main () {
    init();
    vector<int> sizes;  // Adjust as needed
    for (int i = 50000; i >= 10; i -= 500) {
        sizes.push_back(i);
    }
```

```
    random_device rd;
    mt19937 gen(rd());
    ofstream csvfile("search_times.csv");
    csvfile << "algorithm,n,time\n";
    for (int n : sizes) {
        uniform_int_distribution<> dis(0, n);
        vector<int> arr(n);
        for (int& num : arr) {
            num = dis(gen);
        }
        int x = dis(gen);
        test_search_func(csvfile, "linear_search", arr, x);
        sort(arr.begin(), arr.end());
        test_search_func(csvfile, "binary_search", arr, x);
        test_search_func(csvfile, "ternary_search", arr, x);
        test_search_func(csvfile, "fib_search", arr, x);
        test_search_func(csvfile, "interpolation_search", arr, x);
    }
    csvfile.close();
    return 0;
}

void test_search_func(ofstream& csvfile, const string& func_name, vector
    auto start_time = chrono::high_resolution_clock::now();
    if (func_name == "linear_search") {
        linearSearch(arr.data(), arr.size(), x);
    } else if (func_name == "binary_search") {
        binarySearch(arr.data(), arr.size(), x, 0, arr.size() - 1);
    } else if (func_name == "ternary_search") {
        ternarySearch(arr.data(), arr.size(), x, 0, arr.size() - 1);
    } else if (func_name == "interpolation_search") {
        interpolationSearch(arr.data(), arr.size(), x, 0, arr.size() - 1
    } else if (func_name == "fib_search") {
        fibSearch(arr.data(), arr.size(), x);
    }
    auto end_time = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed = end_time - start_time;
    csvfile << func_name << "," << arr.size() << "," << elapsed.count()
}

// Your existing functions go here
int fibSearch(int *a, int n, int x){
    int inf = 0, pos, k, kk = -1, nn = -1;
    if (nn != n) {
        k = 0;
        while (fibo[k] < n) {
            k++;
        }
    } else {
        k = kk;
```

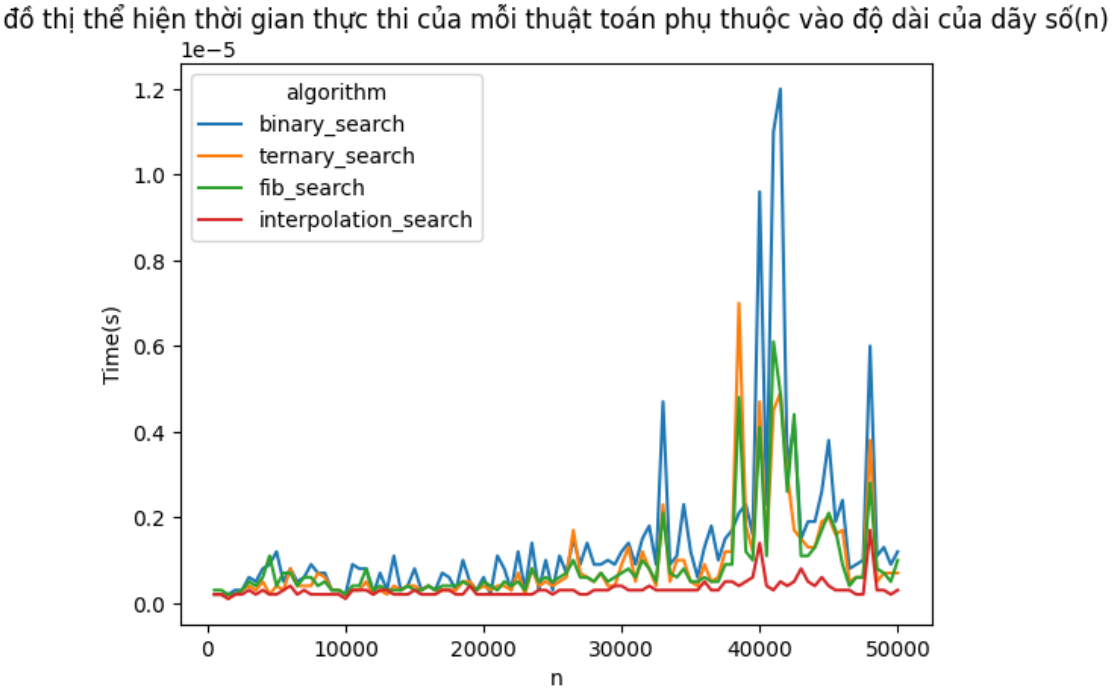
```
    }
    while (k > 0) {
        pos = inf + fibo[--k];
        if (pos >= n || x < a[pos]) {
            // Handle the case when x is less than a[pos]
            k--;
        } else if (x > a[pos]) {
            inf = pos + 1;
            k--;
        } else {
            return pos;
        }
    }
    return -1;
}
void init() {
    fibo[0] = 0;
    fibo[1] = 1;
    for (int i = 2; i < 92; i++) {
        fibo[i] = fibo[i - 1] + fibo[i - 2];
    }
}
int interpolationSearch(int *a, int n, int x, int lo, int hi) {
    int mid;
    while (a[lo] <= x && a[hi] >=  x) {
        if (a[lo] - a[hi] == 0) {
            return (lo + hi) / 2;

        }
        mid = lo + ((x - a[lo]) * (hi - lo)) / (a[hi] - a[lo]);
        if (a[mid] < x) {
            lo = mid + 1;

        } else if (a[mid] > x) {
            hi = mid - 1;
        } else {
            return mid;
        }
    }
    if (a[lo] == x) {
        return lo;
    }
    return -1;
}
int ternarySearch(int *a, int n, int x, int lo, int hi) {
    while (lo <= hi) {
        int mid1 = lo + (hi - lo) / 3;
        int mid2 = lo + 2 * (hi - lo) / 3;
        if (a[mid1] == x) {
            return mid1;
```

```
            }
            if (a[mid2] == x) {
                return mid2;
            }
            if (x < a[mid1]) {
                hi = mid1 - 1;
            } else if (x > a[mid2]) {
                lo = mid2 + 1;
            } else {
                lo = mid1;
                hi = mid2;
            }
        }
        return -1;
    }
    int linearSearch(int *a, int n, int x) {
        for (int i = 0; i < n; i++) {
            if (a[i] == x) {
                return i;
            }
        }
        return -1;
    }
    int binarySearch(int *a, int n, int x, int lo, int hi) {

        while (lo <= hi) {
            int mid = (lo + hi) / 2;
            if (a[mid] < x) {
                lo = mid + 1;

            } else if (a[mid] > x) {
                hi = mid - 1;
            } else {
                return mid;
            }
        }
        return -1;
    }
```

- Các đồ thị:



Hình 1: Đồ thị thể hiện thời gian thực thi của mỗi thuật toán phụ thuộc vào độ dài của dãy số(n)



Hình 2: Đồ thị thể hiện thời gian thực thi của các thuật toán trừ tìm kiếm tuyến tính phụ thuộc vào độ dài của dãy số(n)