**Bài tập tuần 2**

**Tổng quan:**

**- Làm quen với các thao tác tìm kiếm so sánh trên mảng ( sử dụng class vector)**

**- Làm quen với các độ phức tạp O(N), O(N2), và chênh lệch thời gian tính toán thực trên máy tính**

**- Yêu cầu hiểu và viết đc pseudo-code**

**Phần A**: Làm quen với các thuật toán trên dãy số có độ phức tạp O(N)

**Bài 1**:

Biên dịch và Chạy chương trình array1.cpp

1. Trả lời câu hỏi chương trình làm gì
2. Ý nghĩa của số N, K, seed, filename
3. Tìm cách sử dụng hàm đo thời gian

**TRẢ LỜI:**

a, chương trình tạo ngẫu nhiên N số.

b,

N: số lượng phần tử được sinh ngẫu nhiên

K: giới hạn mỗi phần tử A[i]<K

seed: hạt giống để cung cấp cho thuật toán sinh ngẫu nhiên

filename: tên file xuất kết quả

c, hàm clock() trả về giá trị gần đúng số xung của vi xử lý kể từ thời điểm chương trình bắt đầu.

**Bài 2**:

Sử dụng file template default1.cpp.

File default1.cpp sử dụng kiểu vector để đọc vào 1 mảng integer từ file cho trước

1. Thay đổi tham số để đọc vào từ file day1.in

Save lại dưới tên file bai2.cpp

1. Thêm vào đoạn code để tìm số bé nhất của dãy số và in ra màn hình (sử dụng **cout**)
2. Sử dụng chương trình array1.cpp để thay đổi độ dài của file day1.in và in ra thời gian chạy chương trình.

Test với độ dài dãy là N = 10000, 100000, 1000000

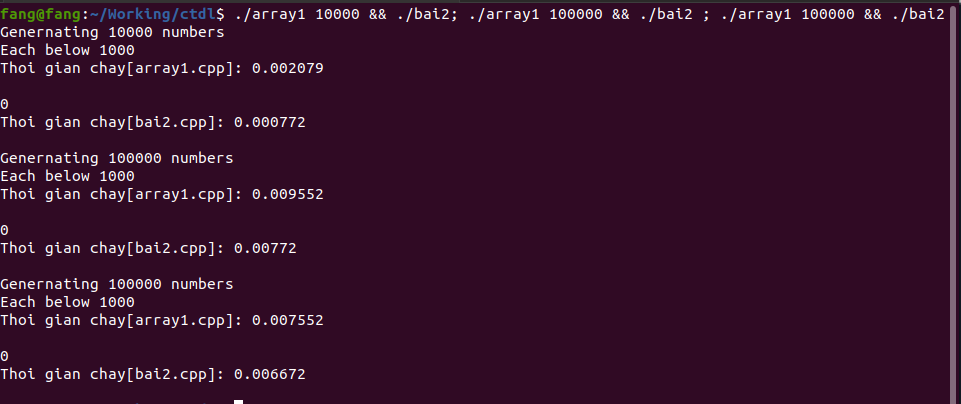
**TRẢ LỜI**

a, thay đổi filename thành day1.in

b, đã thêm trong code (đọc commit bai2 [finish] file bai2.cpp)

c,

hình dưới là thời gian chạy với từng N= 10000, 100000, 1000000



**Bài 3**:

a) Sử dụng template default1.cpp, viết chương trình tính tổng số lớn nhất số bé nhất của dãy số có N chữ số. Dữ liệu vào được lấy từ file **day1.in** và in kết quả ra màn hình (sử dụng **cout**)

Ví dụ:

|  |  |
| --- | --- |
| **day1.in** | **Output** |
| 5  1 2 8 2 5 | 9 |
| 10  1 2 3 4 5 6 7 8 9 25 | 26 |
| 6  1 1 1 1 1 1 | 6 |

b) Tính độ phức tạp của thuật toán.

c) Dùng file array1.cpp, để tạo input cho bài toán. Tính thời gian chạy với:

N = 10000, 100000, 1000000

**TRẢ LỜI**

a, đã thêm

b,

- Trong bài sử dụng 1 vòng for lặp các phần tử của vecter A

- Vị chi sử dụng n lần lặp, với mỗi lần lặp sử dụng 2 lần so sánh ( với min và max) tức là 2n lần so sánh

--> độ phức tạp là O(n)

**Bài 4 : Tìm kiếm – Search (Quan trọng)**

Mô tả: Cho 1 dãy số có N phần tử và 1 số x, in ra vị trí xuất hiện đầu tiên của x trong A, hoặc là NO. File dữ liệu vào là **search.in**, in kết quả ra file **search.out**

**Input**

- Dòng đầu tiên, gồm 2 số: N và x cách nhau 1 dấu cách

- Dòng tiếp theo là N phần tử, cách nhau bởi dấu cách

**Output**

Dòng duy nhất là vị trí xuất hiện đầu tiên của x trong A (tính từ phần tử 0) hoặc là NO

|  |  |
| --- | --- |
| **search.in** | **search.out** |
| 5 8  1 2 8 2 5 | 2 |
| 10 5  1 2 3 4 5 | 5 |
| 6 2  0 0 0 1 1 1 | NO |

Các bước cần làm

Step 1: Dựa vào file default.cpp để viết lại đoạn nhập dữ liệu từ file (đọc thêm số x sau cin << N)

Step 2: Viết thuật toán tìm kiếm vị trí của x

Step 3: Kiểm tra lại kết quả theo kết quả của file mẫu

Step 4: Sửa file array1.in để in ra file search1.in theo đúng format (in thêm 1 số x đằng sau số N). Làm sao để biết là số x có hay ko xuất hiện trong dãy số?

Step 5: Tính độ phức tạp của thuật toán tìm kiếm và thay đổi các giá trị của N để tính thời gian chạy.

N = 10000, 100000, 1000000

Step 6: Xác định thời gian chạy xấu nhất tốt nhất, trung bình với N = 1000000

**TRẢ LỜI**

Step 1: đã sửa trong source

Step 2: viết thành hàm trong file bai4.cpp

Step 3: .ok

Step 4:

Cách 1: viết chương trình chuẩn để tìm kiếm và xem kết quả (rủi do cao)

Cách 2: sử dụng editor có chức năng tìm kiếm để tìm kiếm.

Cách 3: sửa file array1.cpp để in số cần tìm là phần tử ngẫu nhiên trong A.

VD: x = A[rand() % N] : phần tử thứ bất kỳ (chia dư cho N ( luôn < N))

Như vậy x luôn xuất hiện trong dãy A.

Step 5:

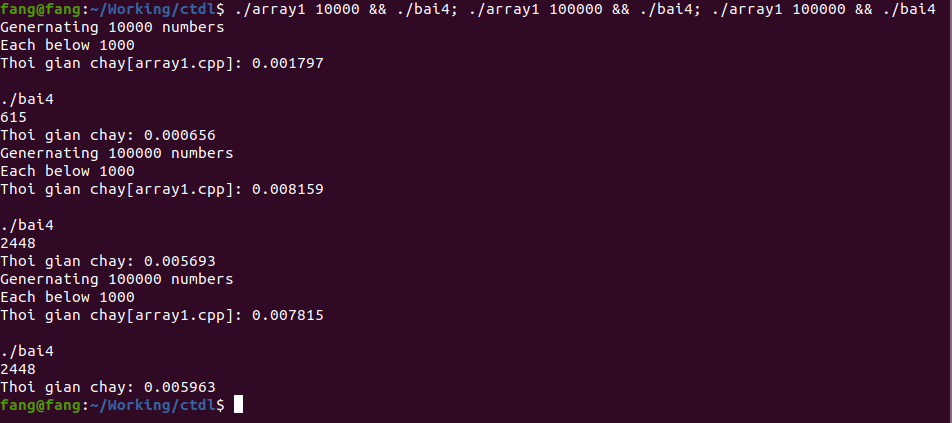
Độ phức tạp:

chương trình thực hiện n lần so sánh. -> độ phức tạp O(n)

Trường hợp tốt nhất là tìm được ở vị trí đầu tiên -> O(1)

Trường hợp tồi nhất là tìm được ở vị trí cuối cùng -> O(n)

-> thời gian chạy trung bình là O(n/2)



Step 6:

*fang@fang:~/Working/ctdl$ echo tốt nhất && ./bai4*

*tốt nhất*

*./bai4*

*0*

*Thoi gian chay: 0.060419*

*fang@fang:~/Working/ctdl$ echo tồi nhất && ./bai4*

*tồi nhất*

*./bai4*

*999999*

*Thoi gian chay: 0.057786*

Output cho thấy thời gian chạy tốt nhất hay tồi nhất khoogn chênh lệch quá nhiều do các bước tính chủ yếu ở đọc và ghi tệp nhưng khi N lớn lên, trhời gian thực hiện vòng for sẽ khác đi đáng kể.

**Phần B**: Làm quen với các thuật toán trên dãy số có độ phức tạp O(N2)

**Bài 5:**

Cho 1 dãy số có N phần tử. Tìm phần tử có số lần xuất hiện lớn nhất, in ra phần tử đó và số lần xuất hiện.

**Input**: day1.in

Dòng đầu tiên là số N. Dòng thứ 2 là N dòng của dãy số.

**Output**: xuathien.out

In ra 2 số: x và k. Cách nhau bởi dấu cách. Số x là số xuất hiện nhiều nhất và số k là số lần xuất hiện. **Trong trường hợp có nhiều số x cùng có số lần xuất hiện nhiều nhất thì in số bé nhất.**

|  |  |
| --- | --- |
| **day1.in** | **xuathien.out** |
| 5  1 2 8 2 5 | 2 2 |
| 10  1 2 3 4 5 6 7 8 9 25 | 1 1 |
| 6  1 1 1 1 1 1 | 1 6 |

Dùng file array1.cpp để tạo ra file day1.in có N trong khoảng [100, 10000]. Tính tốc độ chạy của chương trình. Có nhận xét gì so với lớp thuật toán O(N)

**Bài 6: Quan trọng, thuật toán sắp xếp chèn (insertion sort)**

Dựa vào file **day1.in** và file **default.cpp**, viết chương trình sắp xếp 1 dãy số theo pseudocode sau:

Input: N và A[0] … A[N-1]

Output: Dãy A sắp xếp tăng dần

Pseudocode:

for i = 1 to N-1 do

for j = i to 1 do

if (A[j] < A[j-1]) then

doi\_cho(A[j], A[j-1])

else

break

\* Hàm doi\_cho(a, b): Đổi chổ giá trị của a, b

Yêu cầu: In ra kết quả trên màn hình và tính toán độ phức tạp

**TRẢ LỜI**

Độ phức tạp : ~ O(N^2)

TH tốt nhất: thuật toán sử dụng N-1 phép so sánh

TH tồi nhất: thuật toán sử dụng ~1/2 N^2 phép so sánh và ~1/2N^2 phép đổi chỗ

Với dãy bất kỳ, thuật toán này sử dụng ~ 1⁄4 N^2 phép so sánh và ~ 1⁄4 N^2 phép đổi chỗ

**Bài 7:**

Kết hợp bài 5 với bài 6

- Nếu dãy A sắp xếp tăng dần thì độ phức tạp của bài 5 là bao nhiêu ?

- Viết pseudocode tìm phần tử xuất hiện nhiều nhất trong trường hợp dãy A đc sắp xếp tă`ng dần.

**TRẢ LỜI:**

**- đã làm ở bài 6.**

Pseudocode:

Dem := 1

Max := 0

P\_tu:=A[0]

For i:=2 to N+1 do

If (A[i] = A[i-1]) then

Dem := Dem + 1

Else

IF (Dem > max)

Max:= Dem;

P\_tu:= A[i-1];

Dem := 1;

**Phần C**: Làm quen với các thuật toán có độ phức tạp khác.

**Bài 8:**

Viết chương trình nhập vào số x, in ra biểu diễn nhị phân của số x. Tính độ phức tạp của thuật toán.

Ví dụ

Nhập vào: 10

In ra: 1010

**Trả lời**

Phương pháp dịch bit.

Độ phức tạp tối đa O(64) (tùy kiểu dữ liệu sử dụng)

Ưu điểm : nhanh, dễ dàng cài đặt và sử dụng

Nhược điểm: chỉ xử lý được số ở trong khoảng cho phép. Tối đa 2^64-1 (tùy vi xử lý)

Phương pháp chia liên tiếp

Độ phức tạp : O(n)

Cụ thể O(α) Trong đó α = log2(số cần chuyển) + 1

**Bài 9:**

Viết chương trình nhập vào 1 số a, in ra tất cả các số là ước của số a với độ phức tạp là sqrt(a).

Ví dụ nhập vào 10:

In ra: 1, 2, 5, 10

**Bài 10:**

Cho 2 số a và b, in ra ước chung lớn nhất của a và b. Tính độ phức tạp của thuật toán.