**Bài tập về nhà số 3:**

**Thuật toán + BIểu diễn số nguyên**

Vũ Thị Minh Thư

22028116

Bài 1:

1. (n2  + 8)(n + 1) = O(n2)O(n) = O(n3)
2. (nlogn + n2)(n3+2) = O(n2)O(n3) = O(n5)
3. (n! + 2n)(n3 + log(n2 + 1)) = O(n!)O(n3) = O(n!n3)

Ta có: n! + 2n < nn + 2n => O(n!)

1. (n3 + n2logn)(logn + 1) + (17logn + 19)(n3 + 2)

=max( O(n3)O(logn), O(logn)O(n3)) = O(n3logn)

1. (2n + n2)(n3+3n) = O(2n)O(3n) = O(6n)
2. (nn + n2n + 5n)(n! + 5n) = O(nn)O(n!) = O(nnn!)

Ta có: n!+ 5n < nn + 5n => O(n!)

Bài 2: Thuật toán tìm kiếm tam phân (ternary search) xác định vị trí xuất hiện của khóa tìm kiếm 𝑥 trong một dãy số nguyên đã sắp tăng dần bằng cách liên tục chia dãy thành 3 phần kích thước đều nhau (hoặc gần nhất có thể với kích thước đều), rồi giới hạn vùng tìm kiếm trong phần con phù hợp.

Cách hoạt động của thuật toán:

Chọn 2 biến l,r là vị trí đầu tiên và cuối cùng của mảng a.

Xét 2 vị trí m1 và m2 thuộc [l,r] sao cho l < m1 < m2 < r. Vì mảng tăng dần nên ta biết chắc chắn a[m1] < a[m2].

Chia dãy thành 3 dãy con có kích thước tương đương nhau:

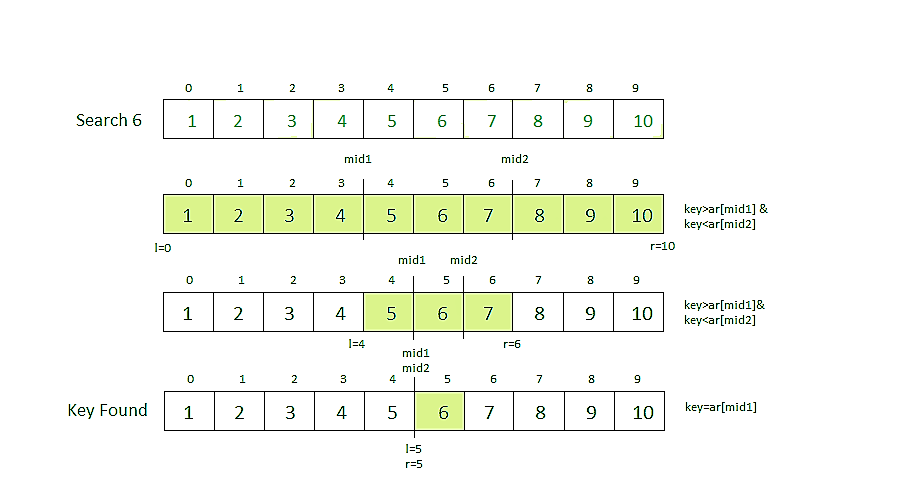
m1 = l+(r-l)/3

m2 = r + (r-l)/3

Tìm kiếm sẽ dừng lại nếu khóa x bằng giá trị của một trong 2 khóa trên, ngược lại sẽ tiếp tục tìm kiếm trong 1 trong 3 dãy con bằng cách so sánh giá trị x với giá trị các khóa để xác định dãy con cần tìm kiếm khóa x:

1. Đầu tiên, chúng ta so sánh khóa x với phần tử ở m1. Nếu tìm thấy bằng nhau, trả về m1.
2. Nếu không thì so sánh khóa x với phần tử ở m2. Nếu tìm thấy bằng nhau, trả về m2.
3. Nếu không, thì chúng tôi kiểm tra xem khóa x có nhỏ hơn phần tử ở m1 hay không. Nếu có thì quay lại phần đầu tiên.
4. Nếu không, thì chúng tôi kiểm tra xem khóa có lớn hơn phần tử ở m2 hay. Nếu có thì quay lại phần thứ ba.
5. Nếu không thì quay lại phần thứ hai (giữa).

Quá trình cứ tiếp tục cho đến khi dãy đang sét là rỗng.



Độ phức tạp về thời gian: O(log3n)

Không gian phụ: O(log3n)

CODE:

```int ternarySearch(int *l*, int *r*, int *key*, int *a*[]) {

    while (*r* >= *l*) {

        int m1 = *l* + (*r* - *l*) / 3;

        int m2 = *r* - (*r* - *l*) / 3;

        if (*a*[m1] == *key*) {

            return m1;

        }

        if (*a*[m2] == *key*) {

            return m2;

        }

        if (*key* < *a*[m1]) {

*r* = m1 - 1;

        }

        else if (*key* > *a*[m2]) {

*l* = m2 + 1;

        }

        else {

*l* = m1 + 1;

*r* = m2 - 1;

        }

    }

    return -1;

}```

Bài 3: Hãy cho biết kết quả từng bước thực hiện thuật toán ở câu trên cho khóa tìm kiếm 𝑥 = 19 trong dãy 1,2,3,5,6,7,8,10,12,13,15,16,19,19,20,23.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bước | Left | Right | m1 | m2 | Kết quả |
| 1 | 0 | 15 | 5 | 10 | Key > a[m2]  Key > 15 |
| 2 | m2+1 = 11 | 15 | 12 | 14 | Key = a[12]  Key = 19 |

Vậy x = 19 ở vị trí 12 trong dãy ban đầu.

Bài 4: Hãy đánh giá độ phức tạp thời gian của thuật toán ở câu 2 cho trường hợp xấu nhất sử dụng kí hiệu O lớn.

Kích thước ban đầu của mảng là n.

Sau lần chạy thứ nhất thì kích thước các dãy con là: n/3

Sau lần chạy thứ 2 thì kích thước các dãy con là: n/32

…

Sau lần chạy thứ k thì kích thước các dãy còn là: n/3k

* TH xấu nhất vòng lặp dừng lại khi n/3k < 3 (vì nếu chia tiếp thì dãy con có kích thước < 1 không thỏa mãn) => n < 3\*3k = 3k+1 => k > log3n – 1

Vì đó là điều kiện dừng của k nên k <= log3n thì thuật toán mới được thực hiện.

=> Độ phức tạp là: O(log3n)

Bài 5: Xét thuật toán sắp xếp chèn nhị phân (binary insertion sort), là bản cải tiến cho thuật toán sắp xếp chèn (insertion sort) đã trình bày trong bài giảng. Hãy trình bày dưới dạng giả mã thuật toán này nhằm sắp xếp tăng dần một dãy số nguyên.

Khởi tạo biến i để chạy hết mảng. Nếu ai<ai-1 thì dùng thuật toán tìm kiếm nhị phân để chạy hết mảng đằng sau và chèn vào vị trí tương ứng.

CODE:

``` #include "bits/stdc++.h"

using namespace std;

int find\_Position(int *a*[], int *left*, int *right*, int *key*) {

    int l= *left*, r= *right*;

    while(l <= r) {

        int mid = l + (r-l)/2;

        if(*key* < *a*[mid])    r = mid-1;

        else l = mid+1;

    }

    return l;

}

void binary\_Insertion(int *a*[], int *n*) {

    for(int i=1; i < *n*; i++) {

        int key = *a*[i];

        int position = find\_Position(*a*,0,i-1,key);

        for(int j = i-1; j >= position; j--) { //xet day[position,i-1]

*a*[j+1] = *a*[j]; // swap

        }

*a*[position] = key; // dao vi tri

    }

}

int main() {

    int a[8] = { 7, 4, 3, 8, 1, 5, 4, 2 };

    int n = 8;

    binary\_Insertion(a, n);

    for(auto it : a) cout << it << " ";

    return 0;

}```

Bài 6: Hãy cho biết kết quả từng bước thực hiện thuật toán ở câu trên

cho dãy 3, 2, 4, 5, 1, 6.

|  |  |  |  |
| --- | --- | --- | --- |
| Bước | Vị trí tìm kiếm | Kiểm tra i-1>=position | Dãy sắp xếp |
| 1 | 0 | True | 2 3 4 5 6 1 |
| 2 | 2 | False | 2 3 4 5 1 6 |
| 3 | 3 | False | 2 3 4 5 1 6 |
| 4 | 0 | True | 1 2 3 4 5 6 |
| 5 | 5 | False | 1 2 3 4 5 6 |

Bài 7: Hãy đếm số phép so sánh dùng bởi sắp xếp chèn và sắp xếp chèn nhị phân khi thực hiện sắp xếp tăng dần cho dãy 7, 4, 3, 8, 1, 5, 4, 2.

Số phép so sánh cần dùng là 53 phép.

``` #include "bits/stdc++.h"

using namespace std;

int cnt=0;

int find\_Position(int *a*[], int *left*, int *right*, int *key*) {

    int l= left, r= right;

    while(l <= r) {

        cnt += 2; // 1dk ở while và trong if

        int mid = l + (r-l)/2;

        if(key < a[mid])    r = mid-1;

        else l = mid+1;

    }

    return l;

}

void binary\_Insertion(int *a*[], int *n*) {

    for(int i=1; i < n; i++) {

        cnt++; // dk lap

        int key = a[i];

        int position = find\_Position(a,0,i-1,key);

        for(int j = i-1; j >= position; j--) { // xet day con [position,i-1]

            a[j+1] = a[j]; // swap

            cnt++; //dk lap

        }

        a[position] = key; // dao vi tri

    }

}

int main() {

    int a[8] = { 7, 4, 3, 8, 1, 5, 4, 2 };

    int n = 8;

    binary\_Insertion(a, n);

    //for(auto it : a) cout << it << " ";

    cout << "So phep so sanh thuc hien la: " << cnt;

    return 0;

}```

Bài 8: Hãy tìm hiểu, trình bày và phân tích độ phức tạp của thuật toán counting sort.

Cách thực hiện:

1. Đếm tần suất xuất hiện của các phần tử bằng map.
2. Kiểm tra nếu tần suất xuất hiện của phần tử lớn 0 thì lưu vào mảng mới ngược lại xét phần tử tiếp theo. Quá trình cứ tiếp tục cho đến khi tới phần tử lớn nhất trong mảng được duyệt.

CODE:

```#include "bits/stdc++.h"

using namespace std;

map<int,int> mp;

vector<int> v;

void Counting\_Sort(int *a*[], int *n*) {

    for(auto it : mp) {

        while(it.second--) {

            v.push\_back(it.first);

        }

    }

}

int main() {

    int n; cin >> n;

    int a[n];

    for(int &x : a) cin >> x, mp[x]++;

    Counting\_Sort(a,n);

    for(auto it : v) cout << it << " ";

}```

Độ phức tạp: O(nlogn) (= max(O(nlogn), O(n))

Bài 9: Hãy cho biết nhiệm vụ tính toán, yêu cầu đầu vào, quy ước đầu ra của thuật toán. Liệt kê các bước của thuật toán để thực hiện 11x49. Phân tích độ phức tạp tính toán.

Chuyển từ hệ cơ số 10 sang hệ cơ số 2:

11 = 1011 => a[4] = {1,1,0,1}

49 = 110001 => b[6] = {1,0,0,0,1,1}