Bài 1:

a/

* Trường hợp 1: chọn thuật toán tìm kiếm tuyến tính.

Cài đặt:

| #include <iostream> using namespace std;  int solve(int A[7], int n, int x) {  int i = 0;  while(i < n) {  if(A[i] == x) return i;  i++;  }  return -1; }  int main() {  int A[] = {1,2,3,4,5,6,100000};  int n = 7;  int x; cin >> x;  cout << solve(A, 7,x);  return 0; } |
| --- |

Trường hợp tốt nhất (best case): chứa khóa x -> Số lần lặp là 1 -> độ phức tạp là O(1).

Trường hợp xấu nhất (worst case): A không có phần tử khóa x -> Số lần lặp là 7 -> Độ phức tạp tuyến tính O(n).

Trường hợp trung bình (Average case): độ phức tạp tuyến tính O(7).

* Trường hợp 2: chọn thuật toán tìm kiếm nhị phân.

Cài đặt:

| #include <iostream> using namespace std;  int solve(int A[], int n, int x) {  int l = 0, r = n -1, m;  while( l <= r) {  m = l + (r - l) /2;  if(A[m] == x) {  return m;  } else if( A[m] < x) {  l = m + 1;  } else {  r = m - 1;  }  }  return -1; }  int main() {  int A[] = {1,2,3,4,5,6,100000};  int n = 7;  int x; cin >> x;  cout << solve(A, 7,x);   return 0; } |
| --- |

Trường hợp tốt nhất: phần tử cần tìm ở đúng vị trí l + (l-r)/2 -> số lần lặp là 1- > Độ phức tạp hằng số O(1).

Trường hợp xấu nhất : số lần tìm là số lần chia đôi dãy đến khi dãy tìm kiếm còn 1 phần tử → số lần lặp khoảng log2(7)+1 → độ phức tạp logarith O(log(7)).

Trường hợp trung bình : độ phức tạp O(log(7)).

* Trường hợp 3 : chọn giải thuật tìm kiếm nội suy.

Cài đặt:

| #include <iostream> using namespace std;  int solve(int A[], int n, int x) {  int l = 0, r = n -1, m;  while(l <= r) {  m = l + (r-l)\*(x-A[l])/(A[r] - A[l]);  if( x == A[m]) return m;  if(x < A[m]) r = m - 1;  else l = m + 1;  }  return -1; }  int main() {  int A[] = {1,2,3,4,5,6,100000};  int n = 7;  int x; cin >> x;  cout << solve(A, 7,x);  return 0; } |
| --- |

Trường hợp tốt nhất : phần tử cần tìm ở đúng vị được nội suy → số lần lặp là 1 → độ phức tạp hằng số O(1).

Trường hợp xấu nhất : giá trị khóa lớn nhất hoặc nhỏ nhất chênh lệch quá lớn so với giá trị kỳ vọng → tìm tuyến tính → độ phức tạp O(7)

Trường hợp trung bình : độ phức tạp O(log(7))

* Vì vậy ta chọn giải thuật tìm kiếm nhị phân là tốt nhất.

b/

* Ta có mảng A[] = {1,2,3,4,5,6, 100000} và x = 6.

Chọn thuật toán tìm kiếm nhị phân.

Đầu tiên, ta gọi và gán l = 0, r = n -1 = 7 -1 = 6 và m

Dùng while với điều kiện l <= r

Gán m = l + (r-l)/2 = 0 + (6 - 0)/ 2 = 3

Tại A[3] = 4 < x

Ta gán l = m + 1 = 3 + 1 = 4

Vì l <= r vẫn thỏa nên ta tiếp tục

Gán m = l + (r - l)/2 = 4 + (6 - 4)/2 = 5

A[5] = 6 = x

Ta kết thúc và trả về vị trí x trang danh sách A là 5 với 2 vòng lặp.

2.

* Dùng thuật toán tìm kiếm nhị phân biến thể với việc left và right đổi giá trị cho nhau và đảo ngược các phép toán logic

Cài đặt:

| #include <iostream> using namespace std; int solve(int A[], int n, int x) {  int l = 0, r = n - 1, m;  while( l > r) {  m = r + (l - r) /2;  if(A[m] == x) {  return m;  } else if( A[m] > x) {  l = m + 1;  } else {  r = m - 1;  }  }  return -1; }  int main() {  int A[] = {100000, 6, 5 , 4, 3, 2, 1 };  int x; cin >> x;  cout << solve(A, 7,x);    return 0; } |
| --- |

3.

Cài đặt:

| #include <iostream> using namespace std;  struct Point {  float x, y; }; int findMinSpace(float arr[], int n) {  int k;  float minn = 300000;  for(int i =0 ; i < n ;i++ ) {  if(minn > arr[i]) {  minn = arr[i];  k = i;  }  }  return k; }  int main() {  Point A[100];  Point target;  cout <<"Nhap toa do x cua diem B";  cin >> target.x >> target.y;  int n; cin >> n;  float arr[100];  cout <<"Nhap toa do mang: ";  for(int i = 0; i < n; i++) {  cin >> A[i].x >> A[i].y;  arr[i] = sqrt(pow((A[i].x - target.x),2) + pow((A[i].y - target.y), 2));  }  int k = findMinSpace(arr, n);   cout << A[k].x << " " << A[k].y;   return 0; } |
| --- |