Homework 2

**Bài 1:**

1. Mã giả:

**program selectionSort**

**input: Mảng A có n phần tử sắp xếp ngẫu nhiên**

**output: Mảng A sau khi sắp xếp từ bé đến lớn**

**start:**

**minIdex = i**

**for i := 0 to length(A) - 1 do**

**for j := i + 1 to length(A) do**

**if A[minIdex] > A[j]**

**minIdex = j**

**temp = A[i]**

**A[i] = A[minIndex]**

**A[minIdex] = A[i]**

**End;**

1. Ta chỉ cần thực hiện với n - 1 phần tử đầu tiên vì:

Sau n - 1 lần thực hiện sắp xếp thì các phần tử của mảng từ vị trí đầu tiên tới vị trí thứ n -1 của mảng là n - 1 phần tử nhỏ nhất của mảng, chúng được sắp xêp theo đúng thứ tự từ nhỏ đến lớn. Do đó phần tử a[n] >= a[n - 1] cho nên mảng đã được sắp xêp theo đung thứ tự sau n - 1 lần lặp.

1. Đánh giá thời gian

* Trường hợp tốt nhất: Với mỗi lần lặp thi phần tử nhỏ nhất cần tìm chinh là phần tử đầu tiên của mảng con và ta không cần đổi chõ phần tử nhỏ nhất với phần tử đầu tiên của mảng con. Hay nói cách khác mảng cần sắp xếp đã được sắp xếp rồi. Nhưng để tim phần tử nhỏ nhất của mảng con ta vẫn phải dùng lặp con. Vậy số lần thực hiện vong lặp con là n\*(n - 1)/2. Thuật toán có độ phức tạp O(n2).
* Trường hợp xấu nhất: Với mỗi lần lặp thì phần tử nhỏ nhất không phải là phần tử đầu tiên của mảng, nên ta cần đổi chõ 2 phần tử đó. Trường hợp này ta vòng lặp con vẫn chạy hết để tìm phần tử nhỏ nhất. Vì vậy thuật toán có độ phức tạp O(n2 ).

1. Biểu đồ.

Ảnh có chứa văn bản

Mô tả được tạo tự động

**Bai 2:**

1. Ý tưởng thuật toán:

* Tạo biến x := 0;
* Vòng lặp duyệt qua mọi giá trị của mảng và so sánh từng giá trị a[i] của mảng với v. Nếu a[i] == v thì gán x = i và hàm sẽ trả về giá trị x ngay trong vòng lặp. Nếu a[i] != v thì tiếp tục vòng lặp.
* Kết thúc vong lặp trả về giá trị x.

1. Mã giả:

**Program findIndex**

**Input: mảng A có n phần tử và giá trị x**

**Output: chỉ số của giá trị x trong mảng hoặc 0 nếu x hông xuât hiên trong mảng**

**Start:**

**For i := 0 to length(A) do**

**If A[i] == x return i**

**return 0;**

**End;**

1. Chứng minh tính đúng

Bất biến vòng lặp: findIndex(x) = i nếu A[i] == x

Khởi tạo: i = 0: findIndex = 0 nếu A[0] == 0

Duy trì: i = j , findIndex = j nếu A[j] == x

Kết thúc: i = n sau n lần lặp, findIndex = n nếu A[n] == x, nếu A[n] != x findIndex = 0

1. Đánh giá thời gian chạy

Trường hợp tốt nhất: Giá trị cần tìm là giá trị đầu tiên của mảng => độ phức tạp O(1)

Trường hợp xấu nhất: Giá trị cần tìm ở vị trí cuối cùng trong mảng hoặc không xuất hiện trong mảng => Độ phức tạp O(n).

1. Biểu đồ trong trường hợp không có giá trị của x trong mảng

Ảnh có chứa văn bản

Mô tả được tạo tự động

**Bài 3:**

1. 105 trang 29

Ta có = +∞

1. 106 trang 29

Ta có:

* log n + 1 = O(2(log n)2 )

1. 107 trang 29

Ta có:

* 4nlog n + n = O((n2 - n)/2)

1. 157 trang 32

Ta có :

Vậy chọn c1 = 99, c2 = 101 thì

C1 (n + (log n)2 ) < 100n + log n < c2 (n + (log n)2 )

Do đó

1. 158 trang 32

Xét log(log(n2 )) = log(2log(n))

* log log(n2 ) = O(log n)

1. 159 trang 32

Xét

* n(log n)2 = O (

**Bài 4:**

1. Bài 264 trang 51

Bất biến vòng lặp: vi =

Khởi tạo: i = 0 , v0 =

Duy trì: lần lặp thứ j, vn – j = A[j + 1] + A[j] \* x = A[j].+ ( =

Kết thúc: i = 0 sau n lần lặp, Horner(A, n) = vn =

1. Bài 265 trang 51

Bất biến vòng lặp: F(n) là số fibonacci thứ n

Khởi tạo: F(0) = 0; F(1) = 1; F(2) = F(0) + F(1) = 2;

Duy trì: F(k + 1) = F(k -1) + F(k)

Kết thúc: i = n sau n – 2 lần lặp => F(n) = F(n - 1 ) + F(n)