**THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

BÁO CÁO TUẦN 3-5

SEARCH - SORT

Tài liệu này ghi nhận lại nội dung các thuật toán search và sort bao gồm:

A. Search

Linear search, Binary search.

B. Sort

Selection Sort, Insertion Sort, Binary Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, Flash Sort.

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

**Biên bản họp nhóm**

# Thông tin chung

Các thành viên có tham dự:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **MSSV** | **Họ và tên** | **Email** |
| 1 | 21280115 | Trần Đức Trung | 21280115@student.hcmus.edu.vn |
| 2 |  |  |  |

Các thành viên vắng mặt:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **MSSV** | **Họ và tên** | **Email** |
| 1 |  |  |  |
| 2 |  |  |  |

Mục tiêu:

1. Hiểu được ý tưởng các thuật toán search và sort
2. Cài đặt thành công và vẽ trực quan so sánh các thuật toán

Địa điểm:

Thời gian bắt đầu: Tuần 3 Thời gian kết thúc: Tuần 5

# Nội dung

**B. SORT**

**Selection Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Mảng được chia thành 2 mảng con là mảng đã và chưa được sắp xếp. Selection Sort tìm phần tử nhỏ nhất trong mảng chưa được sắp xếp và đổi chỗ với phần tử ở đầu mảng chưa sắp xếp. Tức là mỗi vòng lặp, phần tử nhỏ nhất của mảng chưa sắp xếp sẽ được chuyển đến mảng đã sắp xếp.
* **Mã giả:**

Function swap(a, b):

temp = a

a = b

b = temp

Begin (a[], n)

i, j

For i = 0 to n-1 do

min = i

For j = i+1 to n do

If (a[j] < a[min])

min = j

Endif

Endfor

if ( min != i )

swap(a[min], a[j])

Endif

Endfor

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng lộn xộn | Mảng bị đảo ngược |
| Độ phức tạp | O(n2) | O(n2) | O(n2) |
| Stable | Không stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

Tìm phần tử nhỏ nhất trong a[0….4] : min = a[2] = 1. Đổi chỗ với phần tử đầu tiên

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 17 | 24 |

**B2**: i = 2. Lặp lần thứ 2

Tìm phần tử nhỏ nhất trong a[1…4] : min = a[2] = 2. Đổi chỗ với phần tử đầu tiên của a[1…4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B3**: i = 3. Lặp lần thứ 3

Tìm phần tử nhỏ nhất trong a[2..4] : min = a[2] = 5. Giữ nguyên vị trí

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B4**: i = 4. Lặp lần thứ 4

Tìm phần tử nhỏ nhất trong a[3…4]: min = a[4] = 17. Đổi chỗ với phần tử đầu tiên của a[3…4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Insertion Sort**

* **Ý tưởng chung**: Giả sử sắp xếp mảng tăng dần. Bắt đầu duyệt từ phần tử thứ 2 trở đi, so sánh phần tử hiện tại với phần tử trước nó. Nếu phần tử hiện tại lớn hơn thì đổi chỗ 2 phần tử với nhau, sau đó tiếp tục so sánh và đổi chỗ như vậy ở các vòng lặp tiếp theo sao cho đảm bảo tính tăng dần của mảng cho đến khi hoàn thành sắp xếp.
* **Mã giả:**

Begin (a[] ,n)

i, j, current

For i = 1 to n do

current = a[j]

j = i – 1

while (j >=0 and a[j] > current) do

a[j+1] = a[j]

j = j -1

endwhile

a[j +1] = current

Endfor

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp, chỉ có vòng lặp bên ngoài chạy n lần | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
| Độ phức tạp | O(n) | O(n2) | O(n2) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

Vì 5 lớn hơn 2 nên giữ nguyên vị trí của 5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 17 | 24 |

**B2**: i = 2. Lặp lần thứ 2

Vì 1 nhỏ hơn cả 5 và 2 nên 1 sẽ di chuyển về đầu dãy, các phần tử 2 và 5 sẽ di chuyển sang phải 1 vị trí so với hiện tại.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B3**: i = 3. Lặp lần thứ 3

Vì 24 lớn hơn tất cả phần tử từ a[0…2] nên 24 vẫn giữ nguyên vị trí

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B4**: i = 4. Lặp lần thứ 4

Vì 17 lớn hơn 1,2,5 nhưng nhỏ hơn 24 nên 17 sẽ di chuyển về vị trí sau 5, phần tử 24 sẽ di chuyển về sau 1 vị trí

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Binary Insertion Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Mảng được chia thành 2 mảng con đã và chưa được sắp xếp. Duyệt từ phần tử thứ 2 đến phần tử cuối cùng, phần tử hiện tại là key. Tiếp đó dùng Binary Search vào mảng đã được sắp xếp để tìm vị trí “pos” có giá trị lớn hơn key. Dịch chuyển toàn bộ phần tử từ “pos” về 1 sang phải để tạo vị trí trống cho key.
* **Mã giả:**

Function binarySearch(a, key, left, right)

If ( right <= left)

If (key> a[left]) return (left + 1)

Else return left

mid =(left + right) / 2

If (a[mid] == key) return mid+1

If (key > a[mid])

Return binarySearch(a ,key ,mid+1 ,right)

Return binarySearch(a ,key ,left ,mid - 1)

Begin (a[] ,n)

Idx, key

For i =1 to n-1 do

j = i – 1

key = a[i]

idx = binarySearch(a, 0, j, key)

while ( j >= idx) do

a[j +1] = a[j]

j = j – 1

endwhile

a[j+1] = key

Endfor

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp, chỉ có vòng lặp bên ngoài chạy n lần | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
| Độ phức tạp | O(n) | O(n2) | O(n2) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1:** Giả sử phần tử đầu tiên đã được sắp xếp. Key = a[1] = 5

Dùng binary search : 2 < 5 nên giữ nguyên vị trí

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B2**: Key = a[2] = 1

Vì 2 > 1 nên dịch chuyển 2 và 5 sang phải một vị trí.

a[0] = 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B3**: Key = a[3] = 24

Dùng binary search : 24 > 5 nên giữ nguyên vị trí

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B4**: Key = a[4] = 17

Dùng binary search tìm được phần tử lớn hơn 17 trong a[0…3] là 24

Dịch chuyển 24 sang phải một vị trí, lúc này a[3] = 17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

Mảng kết quả:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Bubble Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Ở mỗi vòng lặp, so sánh 2 phần tử liền kề và đổi chỗ nếu chúng xếp sai thứ tự tăng dần. Lặp lại cho đến khi hoàn thành sắp xếp mảng.
* **Mã giả:**

Begin (a[], n)

i, j

For i = 0 to n-1 do

For j = 0 to n – i – 1 do

If (a[j] > a[j+1])

swap(a[j], a[j+1])

Endif

Endfor

Endfor

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
| Độ phức tạp | O(n) | O(n2) | O(n2) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

So sánh a[0] và a[1]. Giữ nguyên vị trí hai phần tử do 5 > 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

So sánh a[1] và a[2]. Đổi chỗ 2 phần tử do 5 > 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 5 | 24 | 17 |

So sánh a[2] và a[3]. Giữ nguyên vị trí 2 phần tử do 5 < 24

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 5 | 24 | 17 |

So sánh a[3] và a[4]. Đổi chỗ hai phần tử do 24 > 17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 5 | 17 | 24 |

**B2** : i = 2. Lặp lần thứ 2

So sánh a[0] và a[1]. Đổi chỗ 2 phần tử do 2 > 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

So sánh a[1] và a[2]. Giữ nguyên vị trí do 2 < 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

So sánh a[2] và a[3]. Giữ nguyên vị trí do 5 < 17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

So sánh a[3] và a[4]. Giữ nguyên vị trí do 17 < 24

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Shaker Sort**

* **Ý tưởng chung:**

Shaker Sort là một cải tiến của Bubble Sort.

Sau khi đưa phần tử nhỏ nhất về đầu mảng sẽ đưa phần tử lớn nhất về cuối dãy. Do đưa các phần tử về đúng vị trí ở cả hai đầu nên Shaker Sort sẽ giúp cải thiện thời gian sắp xếp dãy số do giảm được độ lớn của mảng đang xét ở lần so sánh kế tiếp.

* **Mã giả:**

Begin (arr[], l, r):

Left=l, right=r, k=0

While( left <= right)

for i=left to right-1 do

if (arr[i] > arr[i+1])

swap(arr[i] , arr[i+1])

k=i

end if

end for

right=k

for i=right to leff+1 do

if (arr[i] < arr[i-1])

swap(arr[i] , arr[i-1])

k=i

end if

end for

left=k

end while

end

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(n) | O(n^2) | O(n^2) |
| Stable | Stable | | |

* **Ví dụ:**

**Tạo 2 tham số right=4 và left=0. Nếu right<left thì dừng**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1: Duyệt từ left sang right nếu a[trái] > a[phải] thì swap. Đưa được số lớn nhất về cuối mảng. Cập nhật right=3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 5 | 24 | 17 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 5 | 17 | 24 |

**B2: Duyệt từ right về left nếu a[phải] < a[trái] thì swap. Đưa số bé nhất về đầu mảng.**

**Cập nhật left=1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**B3: tiếp tục lặp lại bước 1 và 2 tới khi right < left và dừng**

**Shell Sort**

* **Ý tưởng chung:** Chọn ra 1 giá trị gap, đánh số từ 1 đến gap cho các phần tử trong mảng ví dụ gap là 3 thì các phần tử được đánh số là 1 2 3 1 2 3 1 2 3 ... Từ đó coi các phần tử được đánh cùng 1 số là 1 mảng con và thực hiện insertion sort trên các mảng con đó.Sau khi thực hiện xong bước trên, giá trị của gap sẽ được thu nhỏ dần về 1 và kết thúc
* **Mã giả:**

Begin (arr[] , l , r)

For ( gap=(r-l+1)/2 ;gap >0 ; gap /= 2 ) do

For i=gap to i==r do

Temp=arr[i]

For ( j=i; j>=gap and arr[i-gap] > temp; j-=gap ) do

arr[j] = arr[j-gap]

end for

arr[j] = temp

end for

end for

end

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(n^2) |
| Stable | Stable | | |

* **Ví dụ:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**Gap =2, i=2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 24 | 17 |

**Gap=2 , i=3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 24 | 17 |

**Gap =2 , i=4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 24 | 17 |

**Gap=1,i=1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 24 | 17 |

**Gap=1,i=2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**Gap=1,i=2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**Gap=1,i=2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Heap Sort**

* **Ý tưởng chung:** Tạo ra 1 cây max tree, và mô phỏng nó trên mảng. Sau đó ta sẽ đổi chỗ node max và node min ( tương ứng với vị trí tại mảng), như vậy giá trị lớn nhất đã được sắp xếp, liên tục lặp lại đến khi mảng được sắp xếp hoàn thiện
* **Mã giả:**

**// Hàm kiểm tra và thực hiện tính chất của max heap tại từng node**

Function heapify(a[], size, i)

nodeFather = i // Lưu lại vị trí node father

left=2\*i+1 // lưu node left của node father

right=2\*i+2 // lưu node right

if (left < size and a[left] > a[nodeFather] )

nodeFather=left

end if

if (right < size and a[right] > a[nodeFather] )

nodeFather=right

end if

if ( nodeFather != i ) // nếu node father ko phải lớn nhất, swap với node con đó

swap(a[i], a[nodeFather] )

// kiểm tra node con sau khi đổi có giữ tính chất của heap ko

heapify(a, size, nodeFather )

end if

Begin(a[] , size)

// Vòng lặp chạy qua từng nodeFather trong tree, bắt đầu tại node father cuối

For i=size/2 -1 to i==0 do:

heapify(a, size, i)

end for

for i=size-1 to i==0 do:

swap( a[0] ,a[i] ) // Đưa node max về cuối mảng

heapify(a , i , 0 ) // Tạo max heap mới ( không tính node max ở cuối mảng )

end for

end

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(nlog(n)) |
| Stable | Không stable | | |

* **Ví dụ:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**Ta minh họa bằng tree được**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 2 |  |  |
|  | 5 |  | 1 |  |
| 24 | 17 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 2 |  |  |
|  | 24 |  | 1 |  |
| 5 | 17 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 24 |  |  |
|  | 2 |  | 1 |  |
| 5 | 17 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 24 |  |  |
|  | 17 |  | 1 |  |
| 5 | 2 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 2 |  |  |
|  | 17 |  | 1 |  |
| 5 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 17 |  |  |
|  | 2 |  | 1 |  |
| 5 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 17 |  |  |
|  | 5 |  | 1 |  |
| 2 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 2 |  |  |
|  | 5 |  | 1 |  |
| 17 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 5 |  |  |
|  | 2 |  | 1 |  |
| 17 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 1 |  |  |
|  | 2 |  | 5 |  |
| 17 | 24 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 1 |  |  |
|  | 2 |  | 5 |  |
| 17 | 24 |  |  |  |

**Ra array là**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Merge Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Chia mảng thành 2 mảng con, tiếp tục lặp lại việc này ở các mảng con đã chia. Sau đó gộp các mảng nhỏ thành mảng lớn hơn đã được sắp xếp, gộp lần lượt tới khi tạo thành mảng ban đầu đã được xếp
* **Mã giả:**

Function Merge (a[], from, mid, to)

first1 = form, last1 = mid

first2 = mid + 1, last2 = to

index = from

aux = [to + 1]

While ( first1 <= last1 and first2 <= last2) do

If (a[first1] <= a[first2])

aux[index] = a[first1]

index+=1

first1+=1

Else

aux[index] = a[first2]

index+=1

first2+=1

Endif

Endwhile

While (first1 <= last1) do

aux[index] = a[first1]

index+=1

first1+=1

Endwhile

While (first2 <= last2) do

aux[index] = a[first2]

index+=1

first2+=1

Endwhile

For i = from to to do

a[i] = aux[i]

Endfor

Begin (a[], from, to)

If (from < to)

mid = (from + to)/2

MergeSort(a, from, mid)

MergeSort(a, mid + 1, to)

Merge(a, from, mid, to)

Endif

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(nlog(n)) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

**Mid=(left+right)/2**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1: Chia mảng thành các mảng con nhỏ hơn tại mid**

**Mid=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 5 | 1 |  | 24 | 17 |

**Mid=1 Mid=3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 |  | 1 |  | 24 |  | 17 |

**Mid=0**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 |  | 5 |  | 1 |  | 24 |  | 17 |

**B2: Hợp các mảng con lại với nhau**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 |  | 1 |  | 24 |  | 17 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 5 |  | 17 | 24 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Quick Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Chọn một phần tử trong mảng là pivot, từ đó chia các phần tử nhỏ hơn pivot sang 1 bên, bên còn lại là các phần tử lớn hơn pivot. Thực hiện tương tự với các mảng con của mảng ban đầu. Cứ chỉ nhỏ tiếp tới khi mảng được xếp
* **Mã giả:**

Begin (a[], first, last)

pivot = a[(first + last)/2]

i = first, j = last

While (i <= j) do

While (a[i] < pivot) do // Duyệt đầu bên trái tới khi gặp phần tử đầu

i = i + 1 //nằm sai vị trí so với pivot

Endwhile

While (a[j] > pivot) do // Duyệt đầu bên phải tới khi gặp phần tử đầu

j = j – 1 //nằm sai vị trí so với pivot

Endwhile

If (i <= j)

swap(a[i], a[j])

i = i + 1

j = j – 1

Endif

Endwhile

If (first < j) // sau khi kết thúc vòng while ta được j nằm bên trái còn i nằm bên phải

QuickSort(a, first, j)

Endif

If (i < last)

QuickSort(a, i, last)

Endif

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Phân đoạn hoàn hảo | Phân đoạn cân bằng | Phân đoạn không cân bằng, pivot nằm ở 1 trong đầu dãy |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(n2) |
| Stable | không stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1: Chọn pivot =a[2].Sau đó chạy vòng while lần 1 được i=0, j=2. Swap a[i] ,a[j] tiếp đó i++, j—được i=j=1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 24 | 17 |

**B2: Chạy vòng while lần 2 được j=0, i=1 => kết thúc while. Đệ quy lại quick sort cho 2 mảng a[0] đến a[j] với j=0 (không thay đổi gì) và a[i] với i=1 đến a[4] (pivot = a[2]).**

**Chạy vòng while được i=1, j=2 , swap và i++, j—được i=2, j=1 dừng while**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 24 | 17 |

**B3: Đệ quy lại quick sort cho từ a[1] đến a[1] ( không đổi ) và từ a[2] đến a[4] ( pivot=a[3])**

**Chạy vòng while được i=3, j=4, swap và i++, j—được i=4, j=3 dừng while**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Counting Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Đầu tiên đếm số lần xuất hiện của các phần tử trong mảng A và lưu vào mảng C, sau đó thay đổi chỉ số giới hạn mảng C. Cuối cùng duyệt từng phần tử của A vào mảng B chứa kết quả sắp xếp thông qua mảng C.
* **Mã giả:**

Begin (a[], n)

output[n]

max = a.max(), min = a.min()

range = max – min + 1

count[range]={0}

For i = 0 to n do

count[a[i] - min] = count[a[i] – min] + 1

Endfor

For i = 1 to k do

count[i] = count[i] + count[i + 1]

Endfor

For i = 0 to n do

output[count[a[i] – min] – 1] = a[i]

count[a[i] - min] = count[a[i] – min] – 1

Endfor

For i = 0 to n do

a[i] = output[i]

Endfor

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(n) | O(n) | O(n) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1**: Đếm số lần xuất hiện của từng phần tử trong mảng a. Kết quả lưu vào mảng count

Max = 24

Min = 1

K = max – min + 1 = 24

Mảng count :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chỉ số | 0 | 1 | 2 | 3 | 4 | 5 | … | 17 | …. | 23 | 24 |
| P.tử | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

**B2**: Sửa đổi giá trị mảng count thể hiện giới hạn của chỉ số phần tử sau khi sắp xếp

Mảng count

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chỉ số | 0 | 1 | 2 | 3 | 4 | 5 | … | 17 | …. | 23 | 24 |
| P.tử | 0 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 5 |

**B3**: Duyệt từng phần tử của mảng a và đặt nó vào đúng vị trí trong mảng output dựa vào mảng count

Mảng output

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Chỉ số | 0 | 1 | 2 | 3 | 4 |
| Phần tử | 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Radix Sort**

* **Ý tưởng chung:** : Giả sử sắp xếp mảng tăng dần. Radix Sort so sánh từng chữ số với nhau từ chữ số bé nhất (hàng đơn vị) đến chữ số lớn nhất.
* **Mã giả:**

Function findMax(a[], n)

max = a[0]

i

For i = 1 to n do

If (a[i] > max)

max = a[i]

Endif

Endfor

max

Function countingSort( a[], n, digit)

A[n]

count[10]={0}

i

For i = 0 to n do

count[(a[i] / digit) % 10 ] = count[(a[i] / digit) % 10 ] +1

Endfor

For i = 1 to max do

count[i] = count[i] + count[i-1]

Endfor

For i = n – 1 to 0 do

A[count[(a[i] / digit) % 10] – 1 ] = a[i]

count[(a[i] / digit) % 10] = count[(a[i] / digit) % 10 ] – 1

Endfor

For i = 0 to n do

a[i] = A[i]

Endfor

Begin (a[], n)

digit

max = findMax(a, n)

For digit = 1 and max/digit > 0 and digit = digit \*10

countingSort(a, n, digit)

Endfor

End

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Cơ số b nhỏ nhất và bằng 0 | K là phần tứ lớn nhất có thể có, b là cơ số | Giá trị b lớn và b = n |
| Độ phức tạp | O(nlogbn) | O((n+b) \* logb(k)) | O(n) |
| Stable | Stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

**B1**: Tìm phần tử lớn nhất trong mảng : a[3] = 24 có 2 chữ số. VÌ vậy sẽ cần dùng 2 vòng lặp để sắp xếp.

**B2:** i = 1. Lặp lần thứ 1

Dùng counting sort để sắp xếp các chữ số hàng đơn vị của từng phần tử

Mảng a1 :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

Đếm số lần xuất hiện của các chữ số mảng a1

Mảng count

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Phần tử sau bằng phần tử liền trước cộng chính nó

Mảng count

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Mảng kết quả A :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 5 | 7 |

Mảng a

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 24 | 5 | 17 |

**B3** : i = 2. Lặp lần thứ 2

Dùng counting sort để sắp xếp các chữ số hàng chục

Mảng a2 :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 2 | 0 | 1 |

Đếm số lần xuất hiện các chữ số mảng a2 :

Mảng count:

|  |  |  |
| --- | --- | --- |
| 3 | 1 | 1 |

Phần tử liền sau bằng phần tử liền trước cộng chính nó, đồng thời giảm count

Mảng count:

|  |  |  |
| --- | --- | --- |
| 3 | 4 | 5 |

Mảng kết quả A :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

**Flash Sort**

* **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Gồm có 3 bước. Bước 1 là Phân lớp dữ liệu dựa trên giả thiết, bước 2 là Hoán vị toàn cục ( đưa các phần tử về lớp của chúng), bước 3 là Sắp xếp bố cục ( sắp xếp trong phạm vi từng lớp).
* **Mã giả:**

Begin (a[], n)

min = a[0], max = 0

m = 0.45\*n

A[m]

For i = 0 to m do

A[i] = 0

Endfor

For i = 1 to n do

If (a[i] > min)

min = a[i]

Endif

If (a[i] > a[max])

max = i

Endif

Endfor

If ( a[max] == min)

Endif

c = (m-1)/(a[max] – min)

For i = 0 to n do

k = c\*(a[i] – min)

A[k] = A[k] + 1

Endfor

For i = 1 to m do

A[i] = A[i] + A[i-1]

Endfor

swap(a[max], a[0])

nmove = 0

j = 0, t = m – 1, k = 0

flash

While (nmove < n – 1) do

While ( j > A[k] – 1) do

j = j + 1

k = c\*(a[j] – a[min])

Endwhile

flash = a[j]

If ( k < 0 ) Endif

While ( j != A[k]) do

k = c\*(flash – min)

hold = a[t = (A[k] = A[k] – 1)]

a[t] = flash

flash = hold

nmove = nmove + 1

Endwhile

Endwhile

insertionSort(a,n)

End.

* **Nhận xét:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự lộn xộn | Mảng có thứ tự đảo ngược |
| Độ phức tạp | O(n) | O(n) | O(n) |
| Stable | Không stable | | |

* **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 24 | 17 |

m = (int) 0.45 x 5 = 2

min = 1

max = 24

c = (m-1)/ (a[max] – min) =1/23

**B1**: Chia làm 2 lớp, lớp 0 gồm 3 phần tử [2, 5, 1], lớp 1 gồm [24, 17]

A = [3 2]  A = [3 5]

Đổi a[max] và a[0]:

Mảng a:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 24 | 5 | 1 | 2 | 17 |

flash = 24

Mảng A[1]-- : A[3 5]  A[3 4]

**B2**: nmove = 1

Đổi chỗ a[0] và a[4]

Mảng a:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | 5 | 1 | 2 | 24 |

flash = 17

Mảng A[0]--: A[3 4]  A[2 4]

**B3**: nmove = 2

Đổi chỗ a[0] và a[3]

Mảng a:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 1 | 17 | 24 |

flash = 2

Mảng A[0]-- : A[2 4]  A[1 4]

**B4**: nmove = 3

Đổi chỗ a[0] và a[2]

Mảng a:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 17 | 24 |

flash = 1

Mảng A[1]-- : A[1 4]  A[1 3]

**B5**: nmove =4

Đổi chỗ a[0] và a[1]

Mảng a:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 1 | 2 | 17 | 24 |

flash = 5

nmove = 4 = n – 1  Stop

**B6**: Phân lớp gồm lớp 0[ 5 1 2], lớp 1[ 17 24]

Dùng insertion sort để sắp xếp lại mảng này.

Mảng kết quả :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 17 | 24 |

# Bảng phân công công việc

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Người phụ trách** | **Mô tả nội dung công việc** | **Bắt đầu** | **Kết thúc** | **Kết quả mong đợi** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |