**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

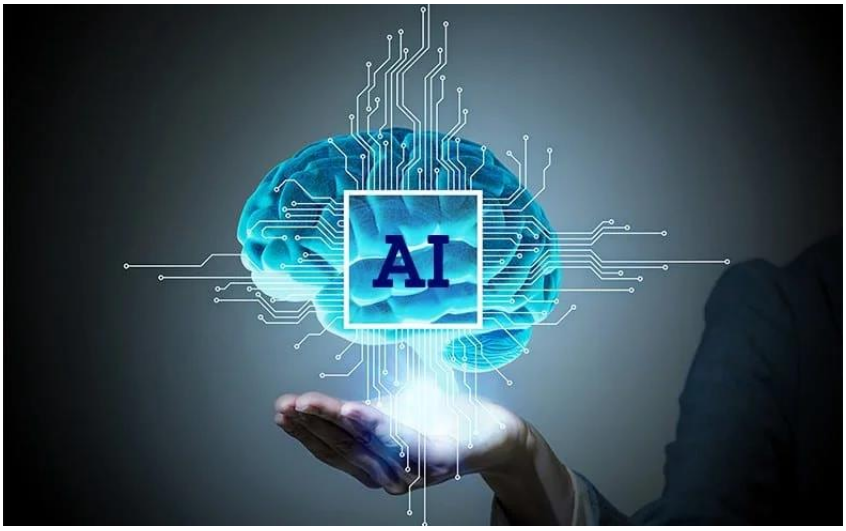
**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**KHOA TOÁN - TIN HỌC**

-----

# BÀI BÁO CÁO THỰC HÀNH TUẦN 7

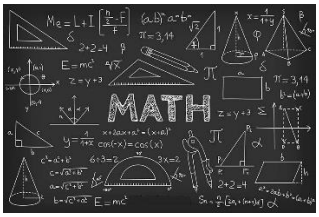



*MÔN HỌC:* **Phân Tích Thuật Toán**

*Sinh Viên:* **Trần Công Hiếu - 21110294**

*Lớp:* **21TTH**

**TP.HCM, ngày 26 tháng 05 năm 2024**




TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA TOÁN – TIN HỌC

# BÀI BÁO CÁO THỰC HÀNH TUẦN 7

***HK2 - NĂM HỌC: 2024-2025***

## MÔN: PHÂN TÍCH THUẬT TOÁN

## SINH VIÊN: TRẦN CÔNG HIẾU

## MSSV: 21110294

## LỚP: 21TTH

# NHẬN XÉT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.HCM, Ngày……Tháng……Năm 2021

Giảng viên Bộ môn

# Mục Lục

# Bài 1.

## 1.1. Trình bày đoạn mã.

**(File: bai1.py)**

Trong đoạn mã này, ta sẽ xét trường hợp mảng A có số phần tử là n = 1236, x = 0 và mảng A được tạo ngẫu nhiên có giá trị từ 0 đến 9. Các giá trị nếu không muốn tự cho trước mà tự nhập thì ta có dòng lệnh được đưa vào comment ngay phía dưới các giá trị mặc định đã đề cập

```
1   import numpy as np
2   n=1236
3   #n = int(input("Nhap n: "))
4   level = n//10
5   x=0
6   #x = int(input("Nhap x: "))
7   A = np.random.randint(0, 10, size=n)
8   count=0
9
10  #print(A)
11  #print(sorted(A))
12
```

“import numpy as np”: Gọi thư viện numpy để thao tác với mảng, tạo số ngẫu nhiên trong đoạn mã của chương trình và gọi tắt là “np”, điều này cho phép gọi các hàm và đối tượng từ thư viện numpy bằng cách sử dụng tiền tố “np”.

“n = 1236”: Khởi gián cho n bằng 1036, lúc sau khai báo mảng A sẽ có 1036 phần tử.

“level = n//10”: Tính giá trị level chính là điều kiện số lần xuất hiện cần ít nhất là bao nhiêu để x là phần tử lấn chiếm cấp độ 10. Và để lấy phần nguyên thì ta dùng toán tử “//”.

“x=0”: Gán giá trị x cần kiểm tra là 0. Lưu ý là có điều kiện x thuộc A, tuy nhiên ta dễ thấy nếu biến count tăng (count khác 0) hoặc x là phần tử lấn chiếm cấp độ 10 thì hiển nhiên x thuộc A, bởi chỉ khi x thuộc A thì mới thỏa so sánh “if(i==x):” như đoạn mã dưới. Còn trường hợp x có là phần tử trong A nhưng không là

phần tử lấn chiếm cấp độ 10 hay x không thuộc A hay không thì không quan trọng.

“A = np.random.randint(0, 10, size=n)”: Tạo mảng ngẫu nhiên A, n phần tử và có giá trị nằm trong đoạn [0, 9].

Có thể in ra mảng A trước và sau khi sắp xếp để quan sát mảng với kết quả.

```
13    for i in A:
14        if(i==x):
15            count+=1
16    print("-Count: ",count)
17    print("-Level: ",level)
18    if(count>=level):
19        print("=>",x,"la phan tu lan chiem cap do 10.")
20    else:
21        print("=>",x,"khong la phan tu lan chiem cap do 10.")
22
```

“for i in A:”: Tạo vòng lặp với i là giá trị từng phần tử trong mảng A.

“if(i==x): count+=1”: Xét điều kiện cho i, nếu i bằng x thì biến đếm count tăng thêm 1.

Kết thúc vòng lặp, ta có thể in thêm phần giá trị biến count và level để tự kiểm tra kết quả cuối cùng.

“if(count>=level):”: Xét điều kiện nếu biến đếm count tức số lần xuất hiện của x lớn hơn hoặc bằng level thì in ra thông báo x là phần tử lấn chiếm cấp độ 10.

Ngược lại, nếu biến đếm count nhỏ hơn level thì in thông báo x không là phần tử lấn chiếm cấp độ 10.

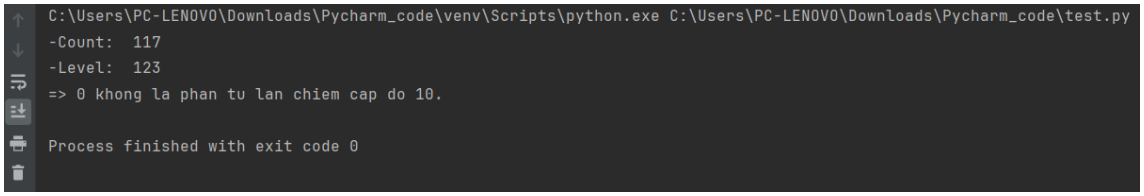
*** Kết quả:**

- Khi x là phần tử lấn chiếm cấp độ 10.

```
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test.py
-Count:  130
-Level:  123
=> 0 la phan tu lan chiem cap do 10.

Process finished with exit code 0
```

- Khi x không là phần tử lấn chiếm cấp độ 10.




## 1.2. Liệu tồn tại thuật toán có độ phức tạp O(n)?.

Như ta có thể thấy ở trên. Hoàn toàn có thể tìm ra phần tử x là phần tử lấn chiếm cấp độ 10 chỉ bằng 1 vòng lặp for để đếm số lần xuất hiện của x trong A. Nếu lớn hơn mức yêu cầu thì x là phần tử lấn chiếm cấp độ 10 và có thuật toán có độ phức tạp O(n).

# Bài 2.

## 2.1. Ý tưởng.

*- Bước 1:* Tạo 2 mảng A và B ngẫu nhiên được sắp xếp theo thứ tự tăng dần. Và 1 mảng khác lưu kết quả merge có thứ tự tăng dần từ 2 mảng A và B.

*- Bước 2:* - Dùng vòng lặp while với điều kiện dừng là có i, hoặc j lớn hơn hoặc bằng n. Với i, j lần lượt là index đại diện của mảng A, B.

- Trong vòng lặp, ta xét giá trị giữa A[i] và B[j]. Nếu giá trị nào nhỏ hơn thì gán vào mảng mới trước vì thứ tự trong mảng mới là nhỏ hơn thì hiển nhiên phải đứng trước.

- Vòng lặp kết thúc khi i hoặc j lớn hơn hoặc bằng n.

*- Bước 3:* Vì Bước 2 kết thúc chứng tỏ hoặc i hoặc j đã bằng n. Tương đương là còn i hoặc j chưa "chạm" tới n nên còn phần tử còn lại thuộc 1 trong 2 mảng chưa được gán vào mảng mới. Do đó, ta chỉ cần duyệt các phần tử còn thiếu đưa vào mảng mới.

## 2.2. Trình bày đoạn mã.

**(File: bai2.py)**

```
1   import numpy as np
2   n = 10
3   #n = int(input("Nhap n: "))
4
5   A = sorted(np.random.randint(0, 100, size=n))
6   B = sorted(np.random.randint(0, 100, size=n))
7
8   C = [0] * (2 * n)
9   i = j = k = 0
10
```

“import numpy as np”: Gọi thư viện numpy để thao tác với mảng, tạo số ngẫu nhiên trong đoạn mã của chương trình và gọi tắt là “np”, điều này cho phép gọi các hàm và đối tượng từ thư viện numpy bằng cách sử dụng tiền tố “np”.

“n=10”: Khởi tạo n bằng 10 là số phần tử của mảng A, B.

“A = sorted(np.random.randint(0, 100,  size=n)”: Tạo mảng A có n phần tử và giá trị nguyên ngẫu nhiên trong đoạn [0, 99] thông qua đoạn mã “np.random.randint()”. Sau đó dùng hàm sắp xếp có sẵn trong python để đưa A về mảng có thứ tự tăng dần.

“B = sorted(np.random.randint(0, 100,  size=n)”: Tương tự mảng A, ta cũng tạo mảng B có n phần tử và giá trị nguyên ngẫu nhiên trong đoạn [0, 99] thông qua đoạn mã “np.random.randint()”. Sau đó dùng hàm sắp xếp có sẵn trong python để đưa B về mảng có thứ tự tăng dần.

“C = [0] * (2 * n)”: Tạo mảng C có 2n phần tử mang giá trị 0 để lưu trữ kết quả khi merge 2 mảng A và B.

“i = j = k = 0”: Khởi tạo 3 biến đếm i, j, k bằng 0. Biến này để vào vòng lặp while duyệt i và j để so sánh giá trị của mảng A và B tương ứng với index i và j. Biến đếm k là để đếm số phần tử được gán vào C có đủ 2n phần tử hay không.

```
11  while i < n and j < n:
12      if A[i] <= B[j]:
13          C[k] = A[i]
14          i += 1
15      else:
16          C[k] = B[j]
17          j += 1
18      k += 1
19
20  while i < n:
21      C[k] = A[i]
22      i += 1
23      k += 1
24
25  while j < n:
26      C[k] = B[j]
27      j += 1
28      k += 1
29
30  print("- Ma tran A:\n",A)
31  print("- Ma tran B:\n",B)
32  print("- Ma tran merge A va B:\n",C)
33
```

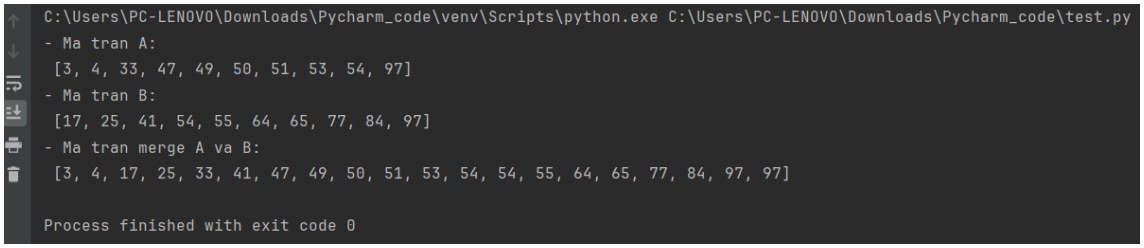
“while i<n and j<n”: Tạo vòng lặp while để kiểm tra đồng thời hoặc i hoặc j có nhỏ hơn n hay không. Nếu không thì ta so sánh giá trị của A[i] và B[j], phần tử nào bé hơn sẽ được đưa vào mảng C và biến đếm được cộng lên 1. Kết thúc vòng while này, sẽ tồn tại phần tử thuộc mảng A hoặc B còn dư lại hoặc trường hợp đẹp nhất là giá trị A và B trong C là so le nên C được hoàn tất merge từ A và B.

“while i<n:”: Nếu còn phần tử trong mảng A “xót” lại (vì chúng lớn nhất trong mảng C” thì ta duyệt từ vị trí thứ i đó đến hết mảng và gán vào C.

“while j<n:”: Tương tự, nếu còn phần tử trong mảng B “xót” lại (vì chúng lớn nhất trong mảng C” thì ta duyệt từ vị trí thứ j đó đến hết mảng và gán vào C.

Cuối cùng, ta lần lượt in mảng A, B và C ra để quan sát kết quả thu được.

## * Kết quả:




## * Nhận xét:

Sẽ còn có các trường hợp đặc biệt như A hay B lớn hơn hoặc nhỏ hơn hoàn toàn so với mảng còn lại. Hoặc mảng A và B có giá trị trong C là so le nhau. Nhưng dù ở trường hợp đặc biệt hay trường hợp thường thì ta cũng chỉ cần xét 2n phần tử. Cho nên độ phức tạp của thuật toán sẽ là O(n) và đây cũng chính là 1 thuật toán khả thi để merge hai mảng A và B thành mảng mới có 2n phần tử được sắp xếp theo thứ tự tăng dần.