Contents

[Re-arrange Array 3 2](#_Toc55250956)

[Sắp xếp các số nhỏ hơn hoặc bằng k đứng cạnh nhau: 2](#_Toc55250957)

[Prime 16 2](#_Toc55250958)

[Tìm các số có 3 ước số trong khoảng từ 1 đến n: 2](#_Toc55250959)

[Large Number 1 2](#_Toc55250960)

[Trừ hai số lớn: 2](#_Toc55250961)

[Re-arrange Array 7 3](#_Toc55250962)

[Nối các số cho trước thành số lớn nhất có thể: 3](#_Toc55250963)

[Range Query 8 3](#_Toc55250964)

[Tìm dãy con liên tục độ dài k có trung bình cộng max: 3](#_Toc55250965)

[Cách hiệu quả: 3](#_Toc55250966)

[Tìm tích lớn nhất của dãy con: 3](#_Toc55250967)

[Sắp xếp mảng a1 size m theo thứ tự của mảng a2 size n: 3](#_Toc55250968)

[String 5 3](#_Toc55250969)

[Sắp đặt các kí tự sao cho các kí tự kề nhau thì khác nhau 3](#_Toc55250970)

[Độ dài lớn nhất có cũng tổng của 2 mảng binary: 3](#_Toc55250971)

[Re-arrang Array 16 3](#_Toc55250972)

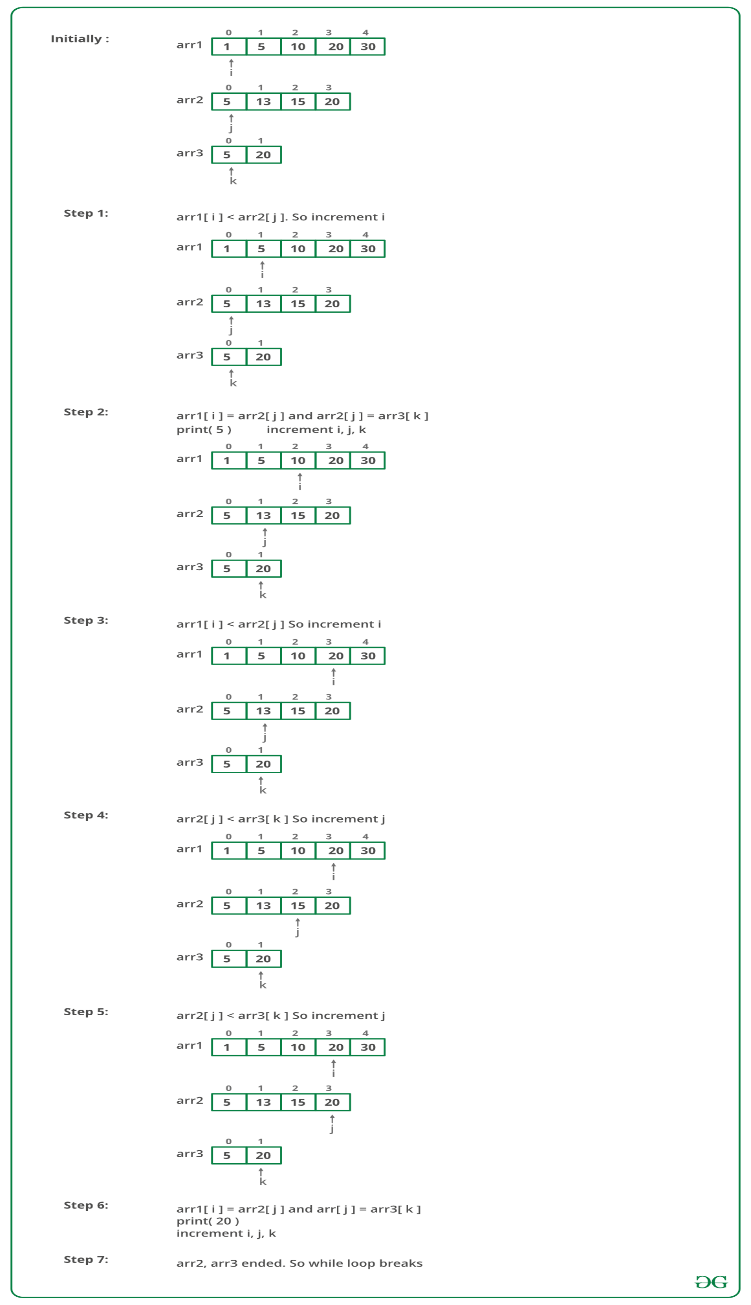
[Dùng 1 phép đổi chỗ duy nhất để tạo ra số lớn nhất nhỏ hơn n: 3](#_Toc55250973)

[Order Statistics 4 4](#_Toc55250974)

[Tìm tổng lớn nhất của mảng con mà không có 2 phần tử kề nhau ở mảng gốc: 4](#_Toc55250975)

[Searching 16 4](#_Toc55250976)

[Tìm các phần tử chung của 3 mảng theo thứ tự tăng dần: 4](#_Toc55250977)

[ 4](#_Toc55250978)

[String 11 5](#_Toc55250979)

[Tìm xâu con nhỏ nhất của s1(string) chứa tất cả các phần tử của s2(pattern): 5](#_Toc55250980)

# Re-arrange Array 3

## Sắp xếp các số nhỏ hơn hoặc bằng k đứng cạnh nhau:

* Đếm tất cả các phần tử nhỏ hơn bằng k=cnt;
* Sử dụng two pointer technique cho cửa sổ có độ dài cnt, mỗi lần theo dõi trong phạm vi này có bao nhiêu phần tử lớn hơn k, gọi số ptu là “bad”.
* Lặp lại bước 2 đến khi tìm được cửa sổ có độ dài cnt có số lượng “bad” là ít nhất =>Kết quả.

# Prime 16

## Tìm các số có 3 ước số trong khoảng từ 1 đến n:

* Số có 3 ước số là bình phương một số nguyên tố.
* =>Chỉ cần đến các số nguyên tố từ 1 đến căn n.

# Large Number 1

## Trừ hai số lớn:

* Check độ dài 2 xâu, đảm bảo độ dài xâu a lớn hơn xâu b.
  + Nếu độ dài bằng nhau thì so snahs trực tiếp 2 xâu.
* Đảo ngược 2 xâu, thêm số 0 vào cuối xâu b sao cho độ dài 2 xâu là bằng nhau.
* Tiến hành phép trừ như lớp 1.

# Re-arrange Array 7

## Nối các số cho trước thành số lớn nhất có thể:

* Ta sẽ định nghĩa lại hàm sort.
* Với 2 số X,Y, ta so sánh XY với YX, cái nào lớn hơn thì lấy.

# Range Query 8

## Tìm dãy con liên tục độ dài k có trung bình cộng max:

* Tạo một mảng có độ dài n(bằng mảng gốc) ,đặt là csum, để lưu trữ tổng tích lũy các phần tử trong mảng.
* Csum[i] lưu trữ giá trị tổng tử 0 đến i.
* Tìm tổng giữa 2 phần tử bằng cách trừ csum với chỉ số tương ứng.

### Cách hiệu quả:

* Tính tổng k phần tử đầu tiên,lưu vào sum, đặt max\_sum=sum;
* Tiếp tục tính từ k đến n-1, bằng cách trừ đi a[i-k[ từ sum và cộng a[i] để được tổng tiếp theo.
  + Nếu tổng mới lớn hơn max\_sum=>max\_sum=sum.

Re-arrang Array 12

## Tìm tích lớn nhất của dãy con:

* Làm chay thế cho nó vuông. Suy nghĩ mệt đầu.

Re-arrang Array 13

## Sắp xếp mảng a1 size m theo thứ tự của mảng a2 size n:

* Sắp xếp mảng a1
* Đặt x=a1[0]-1 để ko trùng với bất kì phần tử nào trong mảng
* Duyệt mảng a2, nếu gặp ptu nào bằng a1 thì in ra,đặt phần tử đó =x,dừng lại khi a1>a2
* Duyệt a1, gặp phần tử nào khác x thì in ra

# String 5

## Sắp đặt các kí tự sao cho các kí tự kề nhau thì khác nhau

* Lưu trữ số lần xuất hiện của mỗi ký tự vào 1 unordered\_map và so sánh số lần xuất hiện max của kí tự với hiệu của (độ dài xâu gốc và số lần xuất hiện max).
* Nếu số lần xuất hiện max nhỏ hơn hiệu thì có thể sắp xếp.

Range Query 12

## Độ dài lớn nhất có cũng tổng của 2 mảng binary:

* Duyệt tất cả các mảng con của 2 mảng cho trước.
* Tình tổng các mảng con:
  + Nếu tổng bằng nhau và chiều dài mảng con lớn hơn max\_length (initialize=0)=>update max\_length.

# Re-arrang Array 16

## Dùng 1 phép đổi chỗ duy nhất để tạo ra số lớn nhất nhỏ hơn n:

* Duyệt từ phải sang, tìm chữ số mà lớn hơn số liền phải nó, đặt chỉ số của em này là idx.
* Tìm phần tử ở bên phải của s[idx] mà có giá trị lớn nhất nhỏ hơn s[idx].
* Swap 2 em này đc số cần tìm.

# Order Statistics 4

## Tìm tổng lớn nhất của mảng con mà không có 2 phần tử kề nhau ở mảng gốc:

* Duyệt tất cả các phần tử và tính incl và excl, trong đó:
  + Incl: max sum bao gồm phần tử trước nó.
  + Excl: max sum ko bao gồm phần tử trước nó.
* Tổng max ko bao gồm phần tử hiện tại sẽ là max(incl,excl).
* Tổng max bao gổm phần tử hiện tại bằng excl + phần tử hiện tại (chỉ xét excl vì các phần tử ko đc liền nhau).
* Kết quả là max(incl,excl).

# Searching 16

## Tìm các phần tử chung của 3 mảng theo thứ tự tăng dần:

* Lập 1 mảng phụ lưu giá trị chung của mảng 1 và mảng 2.
* Tìm giá trị chung của mảng 3 và mảng phụ.

Better solution:

* Gọi phần tử đang đc xét ở mảng a1,a2,a3 lần lượt là x,y,z. Ta có các trường hợp sau:
  + Nếu x,y,z là giống nhau, ta in ra và chuyển chúng lên đầu của 3 mảng.
  + Nếu x<y, ta có thể chuyển x lên đầu mảng a1.
  + Nếu x>z và y>z chuyển z lên đầu mảng 3 vì z ko phải phần tử chung.

# https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190704124815/FindCommonElementsinThreeSortedArrays.png

# String 11

## Tìm xâu con nhỏ nhất của s1(string) chứa tất cả các phần tử của s2(pattern):

* Lưu sự xuất hiện của các phần tử của s2 vào một hash\_pat.
* So sánh các kí tự của s2 với các kí tự của s1, tức là tăng count nếu hai giá trị match.
* Nếu (count== length of s2) nghĩa là window đang xét là ok. Tuy nhiên window này chưa phải là sub string nhỏ nhất.
* Tìm cách giảm bớt các kí tự ở đầu của của sổ hiện tại nếu có thể.
* Update min\_length.
* In ra cửa sổ nhỏ nhất.

# Sorting 7

## Tìm dãy con liên tục dài nhất sao cho sau khi sắp xếp lại dãy này ta được mảng đã sắp xếp:

* Duyệt từ trái sang phải, tìm phần tử đầu tiên lớn hơn ptu kế tiếp, vị trí là start.
* Duyệt từ phải sang trái, tìm phần tử đầu tiên nhỏ hơn ptu kế tiếp, vị trí là end.
* Tìm max và min trong khoảng start và end.
* Kiểm tra từ 1 đến s-1 có ptu nào lớn hơn min, nếu có start = vi tri cua em no.
* Kiểm tra từ e+1 đến n xem có ptu nào nhỏ hơn max, nếu có start=vi tri cua em no.