Contents

[Re-arrange Array 3 1](#_Toc54086479)

[Sắp xếp các số nhỏ hơn hoặc bằng k đứng cạnh nhau: 1](#_Toc54086480)

[Prime 16 1](#_Toc54086481)

[Tìm các số có 3 ước số trong khoảng từ 1 đến n: 1](#_Toc54086482)

[Large Number 1 1](#_Toc54086483)

[Trừ hai số lớn: 1](#_Toc54086484)

[Re-arrange Array 7 1](#_Toc54086485)

[Nối các số cho trước thành số lớn nhất có thể: 1](#_Toc54086486)

[Range Query 8 2](#_Toc54086487)

[Tìm dãy con liên tục độ dài k có trung bình cộng max: 2](#_Toc54086488)

[Cách hiệu quả: 2](#_Toc54086489)

[Tìm tích lớn nhất của dãy con: 2](#_Toc54086490)

[Sắp xếp mảng a1 size m theo thứ tự của mảng a2 size n: 2](#_Toc54086491)

[String 5 2](#_Toc54086492)

[Sắp đặt các kí tự sao cho các kí tự kề nhau thì khác nhau 2](#_Toc54086493)

[Độ dài lớn nhất có cũng tổng của 2 mảng binary: 2](#_Toc54086494)

[Re-arrang Array 16 2](#_Toc54086495)

[Dùng 1 phép đổi chỗ duy nhất để tạo ra số lớn nhất nhỏ hơn n: 2](#_Toc54086496)

# 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.