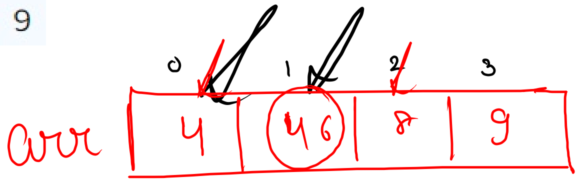
Form the largest number

```
4
4  46  8  9
```



ans = 98464

~~98644~~

$$\begin{array}{r} 46 \\ 4 \\ \hline 50 \end{array}$$

string

$\boxed{464}$

concatination

abc

abcd

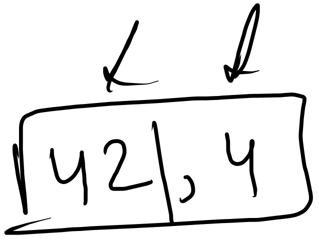
42

90
80
46
40

↓ing

442

424



int arr =

4	46	8	9

String arr1 =

"4"	"46"	"8"	"9"

logic

a = "4"

b = "46"

a+b → "446"

b+a → "464"

Concatinate

String arr1 =

"9"	"8"	"46"	"4"

String ans = 9 8 46 4

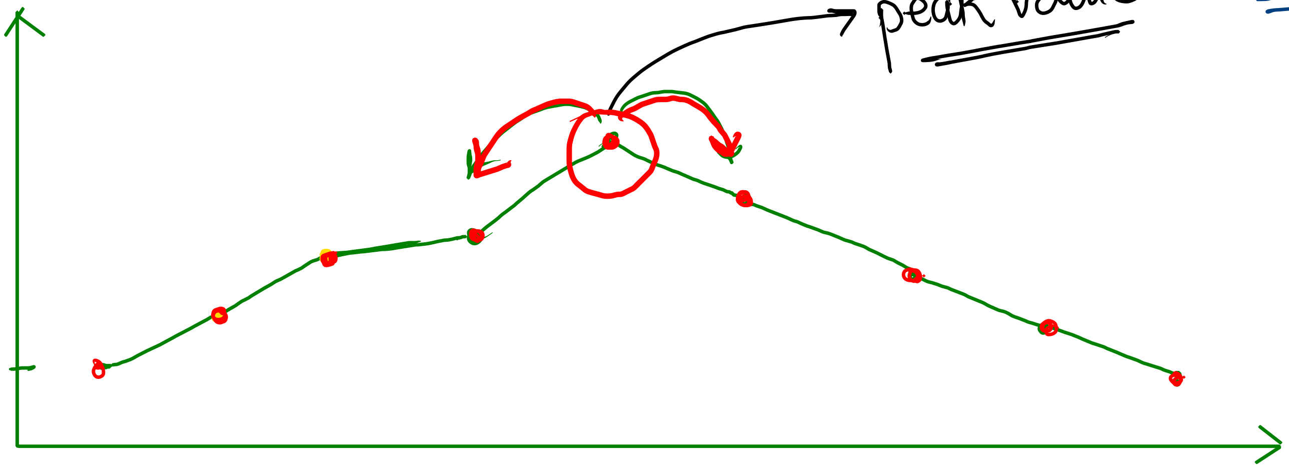
code

```java
public static String formLargestNum(int[] arr, int n) {
    String[] arr1 = new String[n];
    for (int i = 0; i < n; i++) {
        arr1[i] = String.valueOf( arr[i] );
    }

    Arrays.sort( arr1, ( a, b ) -> {
        String str1 = a + b;
        String str2 = b + a;

        return str2.compareTo(str1);    // decreasing
    } );

    String ans = "";
    for (int i = 0; i < n; i++) {
        ans += arr1[i];
    }

    return ans;
}
```

Peak Index in a Mountain Array 2

$$\begin{array}{ccccccccccc} & 0 & 1 & 2 & 3 & \underline{\underline{4}} & 5 & 6 & 7 & 8 & \\ arr = [& 1, & 3, & 5, & 6, & \underline{\underline{8}}, & 7, & 4, & 2, & 1 &] \end{array}$$

$\underline{\underline{ans = 4}}$



condition is, array contains only single value which satisfy

$$arr[i-1] < arr[i] > arr[i+1]$$

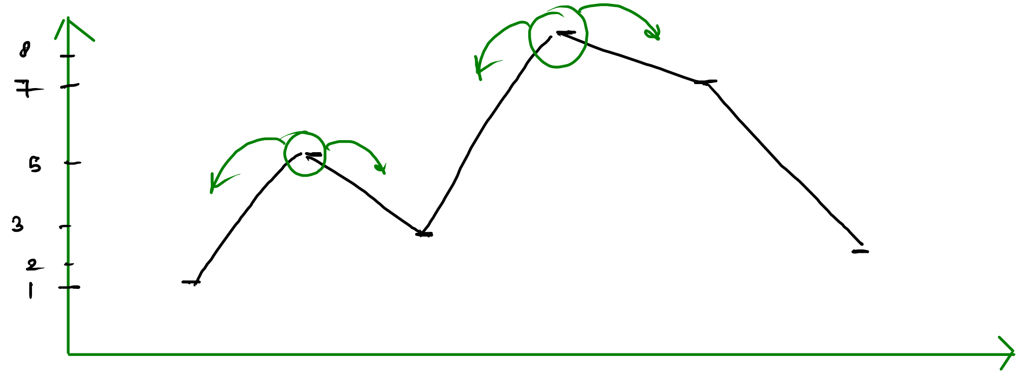
code

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    System.out.println(peakIndex(arr, n));
}

public static int peakIndex(int[] arr, int n) {
    for (int i = 1; i < n - 1; i++) {
        if ( arr[i] > arr[i - 1] && arr[i] > arr[i + 1] ) {
            return i;
        }
    }
    return -1;
}
```

T.C
linear
O(N)

Peak Elements

arr = [1 5 3 8 7 2]



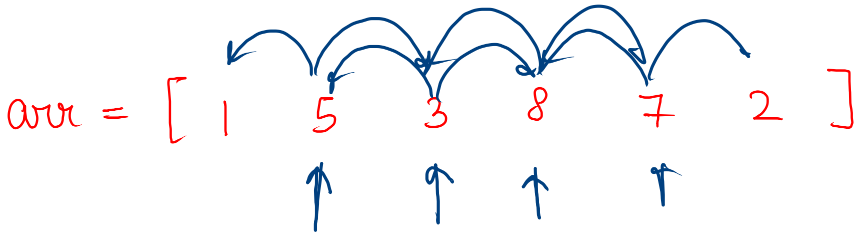
return :– 1) it provide some value back from where the f^n was called.

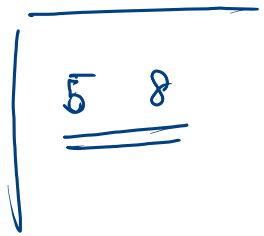
2) It destroys the function

break :- terminate the loop

code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    peakIndex(arr, n);
}

public static void peakIndex(int[] arr, int n) {
    for (int i = 1; i < n - 1; i++) {
        if ( arr[i] > arr[i - 1] && arr[i] > arr[i + 1] ) {
            System.out.print( arr[i] + " " );
        }
    }
}
```



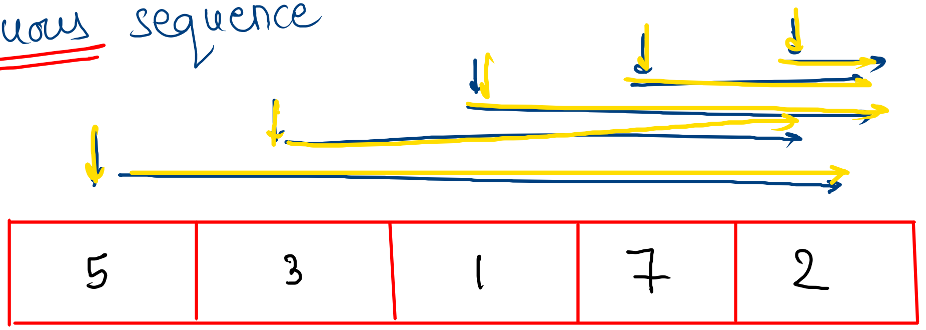


⇒ Subarrays / Subsets

↳ subarray is continuous sequence within our array

↳ order need to be preserved

arr =



subarrays :-

5
- 5
- 5 3
- 5 3 1
- 5 3 1 7
- 5 3 1 7 2

4
- 3
- 3 1
- 3 1 7
- 3 1 7 2

3
- 1
- 1 7
- 1 7 2

2
- 7
- 7 2

1
- 2

total elements

$= 5+4+3+2+1$

$= 15$

if arr size is n

total $\Rightarrow n+(n-1)+(n-2)+\dots+1$

$\Rightarrow n*(n+1)/2$

$\Rightarrow O(n*(n+1)/2) \cong O(N^2)$

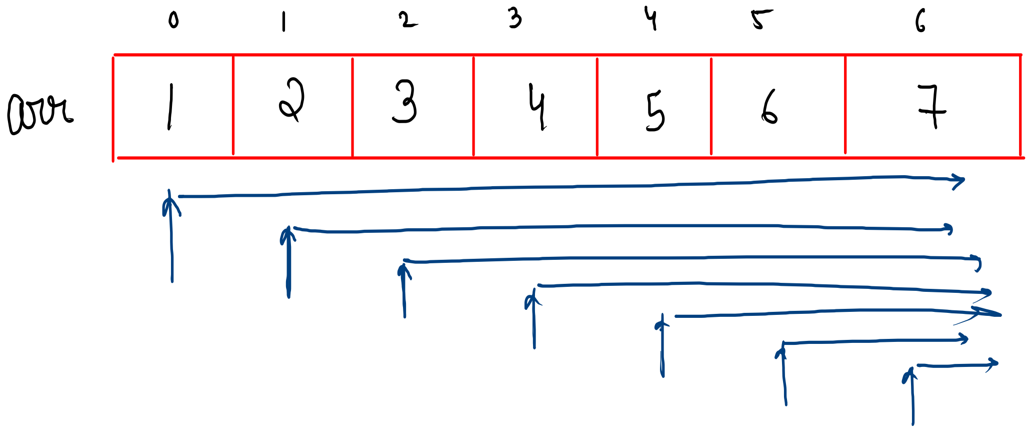
Imp

↳ 1 7 2

↳ 1 3 7 ✗

↳ 5 1 7 ✗

non-continuous



n = size

si → start idex

ei → end index

i=0, j=0 → n

i=1, j=1 → n

i=2, j=2 → n

i=3, j=3 → n

⋮

```
for( int i = 0 ; i< n ; i++) {
    for( int j = i ; j<n ; j++) {
        Syso( arr[j]+ "  ");
    }
    Sysoln();
}
```

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    subarrays(arr, n);
}

public static void subarrays(int[] arr, int n) {
    for (int i = 0; i < n; i++) {    // start index
        for (int j = i; j < n; j++) {  // end index
            print(arr, i, j);
        }
    }
}

public static void print(int[] arr, int si, int ei) {
    for (int i = si; i <= ei; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}
```

$O(N^2 * N)$

1 3

arr

3	1	2	4
0	1	2	3

i=0, j=0 → 3

i=0, j=1 → 3 1

i=0, j=2 → 3 1 2

i=0, j=3 → 3 1 2 4

i=0, j=4 ✗

i=1, j=1 → 1

i=1, j=2 → 1 2

$\boxed{i=1, j=3 \rightarrow 1\ 2\ 4}$

i=1, j=4 ✗

i=2, j=2 → 2

i=2, j=3 → 2 4

i=2, j=4 ✗

i=3, j=3 → 4

i=3, j=4 ✗

i=4 ✗

Sum Equals Zero

$$arr = [\overset{0}{5} \quad \underbrace{\overset{1}{-2} \quad \overset{2}{3} \quad \overset{3}{-1}} \quad \overset{4}{4}]$$, ans = true

find subarrays ↳

(5) 5

(3) 5 -2

(6) 5 -2 3

(5) 5 -2 3 -1

(9) 5 -2 3 -1 4

(-2) -2

(1) -2 3

return ⇒ (0) -2 3 -1

-2 3 -1 4

3

3 -1

3 -1 4

-1

-1 4

4

brute force

↳ most basic approch

code

```java
public static boolean subarrays(int[] arr, int n) {
    for (int i = 0; i < n; i++) {   // start index
        for (int j = i; j < n; j++) {  // end index
            int sum = findSum(arr, i, j);
            if (sum == 0) {
                return true;
            }
        }
    }
    return false;
}

public static int findSum(int[] arr, int si, int ei) {
    int sum = 0;
    for (int i = si; i <= ei; i++) {
        sum += arr[i];
    }
    return sum;
}
```

here, we can also use "Kadane's algo."