1. Diff k II

<0, 1, 2, 3, ...>

Given an array A of integers and another non-negative integer k, find if there exists 2 indices i and j such that A[i] - A[j] = k, i != j.

Given :- A = < > k >= 0

Output - 1 / 0

Ex - A = <1, 5, 3>
k = 2

1st approach → consider all pairs

- <1, 5> → 4 <1, 3> → 2
- <5, 3> → 2

TC: O(n^2)
SC: O(1)

2nd approach →

<1, _ >

-1 3

k = 2

<a, ... > search a+k & a-k

If we sort arr, search on right

<1, 3, 5>

search -1 and 3

TC: O(nlog n)
SC: O(n)

3rd approach → use a hashset

(a)

A = <1, 3, 5> k=2

1st 2nd

1 _

-1 3

Set

<1, 3, 5>

2nd A = <1, 3, 5>

TC : O(n) → O(1)

Step 1 : Insertion in set → n

Step 2 : Finding a+k & a-k → n

TC : O(n)
Going through arr 2 times

3rd (b) approach

A = <1, 5, 3>
K = 2

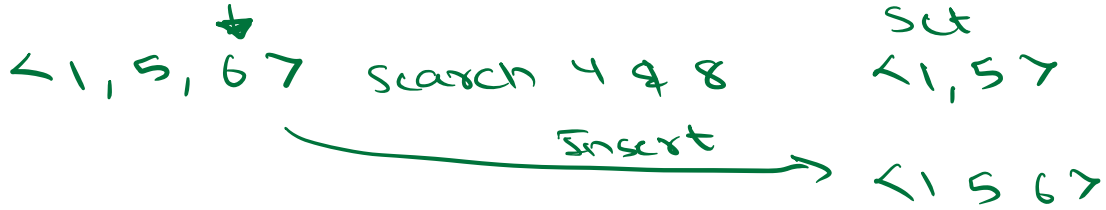
- <1> search -1 and 3 — Set <>
  Insert → <1>
- <1, 5> search 3 and 7 — <1>
  K = 2
  Insert → <1, 5>

- <1, 5, 3> search 1 and 5 — <1, 5>
  K = 2
  (3-2) (a-K)
  (3+2) (a+K)

Return 1 (3,1 and 3,5)

A = <1, 5, 6>

K = 2



Reached end of A, return 0

Code:

```
public int diffPossible(final int[] A,
                        int B) {

    Hashset<Integer> s = new Hashset<Integer>()
    for (int num : A) {

        if (s.contains(num - B) ||
            s.contains(num + B))
        return 1;
        s.add(num);


    }

    return 0;
}
```

TC: O(n)

Iterating through A once

SC: O(n) (set)

2. Largest Number <0, 1, 2, ...>

Given an array A of non-negative integers, arrange them such that they form the largest number.

Note - The result may be very large, so return a string instead of an integer.

$1 <= len(A) <= 100000$

$0 <= A[i] <= 2 \times 10^9$

Ex A = <4, 2, 9, 1>

Nos. → 4291, 1294, ...

Total nos. → 4! (n!)

Largest no. → 9421 A = <9, 4, 2, 1>

1. Sort → Descending
2. Largest digit → Left

Ex A = <3, 30, 34, 5, 9>

Nos. → 3303459, 3343059, ... (5!)

Sort in descending

A = <34, 30, 9, 5, 3>

No → 3430953 ✗ (Not largest)

Largest no → 9534330 Sort → ↓↓↓ ✗

A = <9, 5, 34, 3, 30> Largest digit → left ✓

A = <30, 9> → Input

Nos = 309 < 930 ✓

<9, 30> → output

x y
30 9

y x x y
9 30 > 30 9

(9 comes before 30)
y comes before x

→ _ 9 _ _ _ 30 _

2 [9 30] 5 > 2 [3 09] 5

29305 > 23095

A = <3, 30, 34, 5, 9>

- <3, 30>
  x y
  3 30 > 30 3
  x y y x
  x comes before y
  3 comes first
  _3 _ _ _30_

- <3, 34>
  3 34 < 34 3
  x y y x
  y comes before x
  34 comes 1st
  _34 _ _3 _ _30

- <3, 5>
  35 < 53
  x y y x
  5 comes before 3
  5 5
  ↓ ↓
  _34 _ _3 _ _30

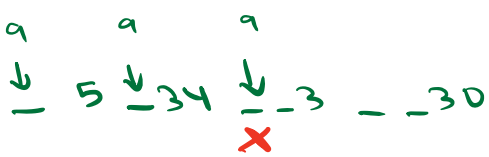
- <3, 9>
  39 < 93
  x y y x
  9 comes before 3
  9,5 9,5
  ↓ ↓
  _34 _ _3 _ _30

- <30, 34>
  30 34 < 34 30
  x y y x
  34 comes before 30
  9,5 9,5
  ↓ ↓
  _34 _ _3 _ _30

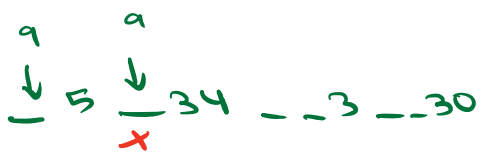
- <30,5> 305 < 530 (x y, y x) — 5 comes before 30

- <30,9> 309 < 930 — 9 comes

- <34,5> 345 < 543 (x y, y x) — 5 comes before 34



- <34,9> 349 < 943 (x y, y x) — 9 comes before 34



- <5,9> 59 < 95 (x y, y x) — 9 comes before 5

9_5_34_3_30

<9, 5, 34, 3, 30> → custom sort

Largest no → 9534330

Code :- A <3, 34, 30, 5, 9>

Node → any element in A

```
class Node implements Comparable<Node> {
    int number;
    public Node(int num) {
    this.number = num;
    }
```

```
@Override
public int compareTo(Node o) {
        //309
    String xy = String.valueOf(this.number) +
                String.valueOf(o.number)
        //930
    String yx = String.valueOf(o.number) +
                String.valueOf(this.number)

    return yx.compareTo(xy);
}
```

"930".compareTo("309")

<30,9>

a b

a.compareTo(b)

this → 30

b → 9

<x,y> → <y,x>

<9,307>

930 > 309

swap(30,9)

yx > xy

swap(x,y)

- public class Solution {
    public String largestNum(List<Integer> A){

```
StringBuffer strbuf = new StringBuffer()
Node num[];
int i=0;
num = new Node[A.size()];
```

// integers stored in nodes // <30,9>

```
for (int n : A)
{
    num[i] = new Node(n);
    i++;
}
// num[0] = new Node(30);

Arrays.sort(num);   // custom sort

for (Node n : num) {
if (n.number == 0 && strbuf.length() == 1 &&
strbuf.toString()[0] == '0')
    continue;
strbuf.append(n.number);
}
return strbuf.toString();
}
```

→ n

→ default constructor

Node → 30 this.number

call compareTo

→ n

→ largest no.

→ edge case

- A = <0, 0, 0, 0> code → 0000 ✗

  Largest No. → 0 ✓

  strbuf → empty

  A = <0, 0, 0, 0> "0"

  A = <0, 0, 0, 0> "0"

- A = <0, 30, 0> → <30, 0, 0>

  Largest No → 3000

TC O(nlogn) sort algo

compareTo → O(1)

SC O(n) → no. of nodes = A size

3. Subarray with given sum

Given an array of positive integers A and integer B, find and return first continuous subarray which adds to B. If answer doesn't exist, return an array with single element -1.

First sub-array means the sub-array for which starting index is minimum.

A = <2, 1, 3, 1, 5> (0 1 2 3 4)

B = 5

<1,3,1> (2 3 4) <5> (4)

output → <1,3,1>

min (2,4) = 2

1st Approach - Consider all subarrays

sum = 0

- 2 → sum = 2 (+1) → 2 ≠ 5, 2 < 5
- 2,1 → sum = 3 (+3) → 3 ≠ 5, 3 < 5
- 2,1,3 → sum = 6 → 6 ≠ 5, 6 > 5

- 2,1,3,1 ✗ (> B)

sum > B stop (i++)

i

- 1 (+3) → sum = 1 → 1 ≠ 5, 1 < 5
- 1,3 (+1) → sum = 4 → 4 ≠ 5, 4 < 5
- 1,3,1 → sum = 5 → 5 = 5

i j

return (1,3,1)

<i,j> subarray

TC $O(n^2)$ < i j

SC $O(1)$

A = <1,2,3,4>

B = 50

↓

worst case

A = < 2 , 1, 3, 4, 6, 6 > B = 12

sum = 0

· 2 (i, j) — sum = 2 — 2 < 12 (sum < B, j++)

+1

· 2, 1 (i, j) — sum = 3 — 3 < 12

+3

· 2, 1, 3 (i, j) — sum = 6 — 6 < 12

+4

· 2, 1, 3, 4 (i, j) — sum = 10 — 10 < 12

+6

· 2, 1, 3, 4, 6 (i, j) — sum = 16 — 16 > 12 ✗

stop (sum > B)

2, 1, 3, 4, 6, 6

-2

· 1, 3, 4, 6 (i, j) — sum = 14 — 14 > 12 ✗

stop

1, 3, 4, 6 ✗

1, 3, 4, 6, 6 ✗

1 ✗

1, 3 ✗

1, 3, 4 ✗

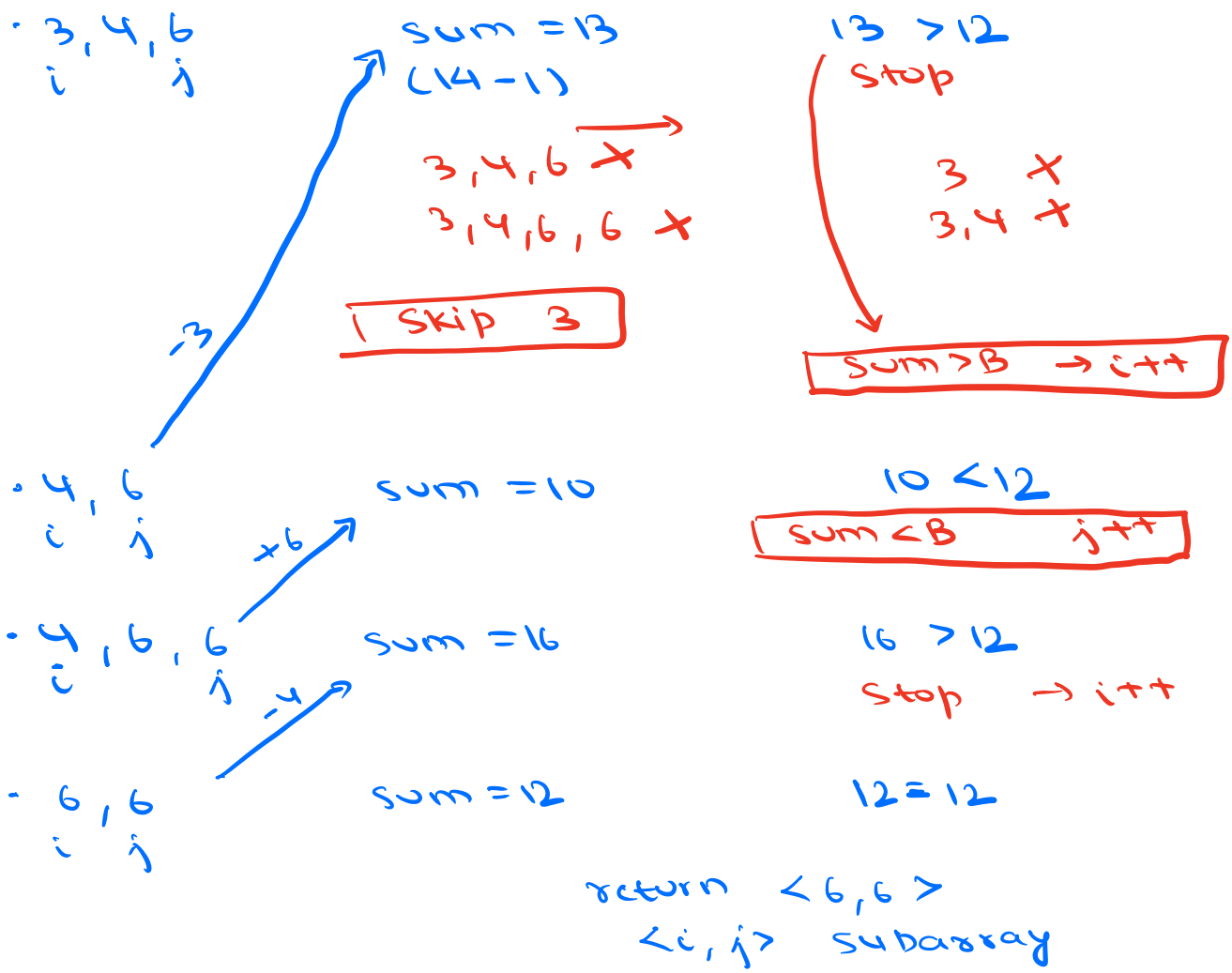
sum < 2, 1, 3, 4 > < 12 | x < 12

↓ -2

sum < 1, 3, 4 > < 12 | x - 2 < 12

Skip 1

sum > B i++



sum < B j++
sum = B < i, j >
sum > B i++

Code

```
public int[] solve(int[] A, int B) {
    long n = A.length
    int i=0, j, sum=0;
    for (j=0; j<n; j++) {
        sum += A[j];
        if (sum > B) {
            while (sum > B) {
                sum -= A[i];
                i++;
            }
        }
        else if (sum == B) {
            int[] ans = new int[j-i+1];
            for (int k=i; k<=j; k++) {
                ans[k-i] = A[k];
            }
            return ans;
        }
    }
    int[] ans = new int[1];
    ans[0] = -1;
    return ans;
}
```

TC O(n)

2 opr per element

① addition

② removal

2n opr

SC O(1)

4. Sort by Color

Given an array with N objects colored red, white or blue, sort them so that objects of the same color are adjacent, with colors in order red, white & blue.
We will use the integers 0, 1, 2 to represent red, white and blue respectively.
Note - using the library sort function is not allowed.

R, W, B
[0 1 2] → output

Sort 0, 1, 2

A = <0, 1, 2, 0, 1, 2>

1st approach → count 0s, 1s and 2s

Step 1:- c0 = 2, c1 = 2, c2 = 2

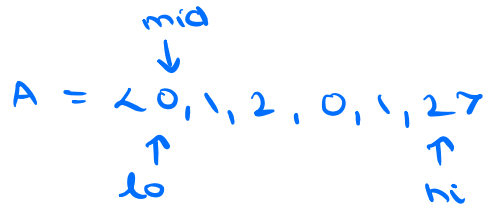
Step 2:- <0 1 2 0 1 2>
<0 0 1 1 2 2>

TC O(N)
SC O(1)
2 iterations

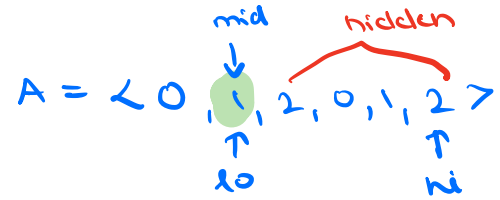
2nd approach →

0 | 1 | 2
Left | Mid | Right

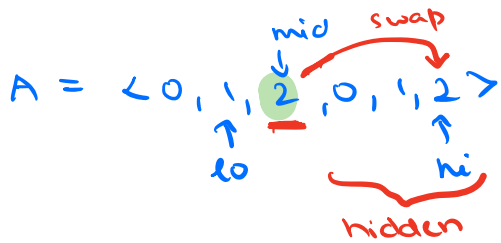
0 0 1 1 2 2 2 2 2 2 2
Mid
pivot
← 0 to left
2 to right →



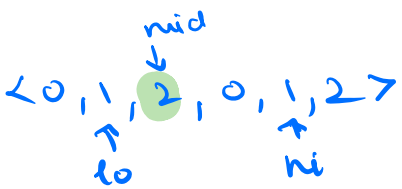
lo → 0 is placed
hi → 2 is placed
mid → array iteration

- 

1 starts where 0 ends
mid++ (move to next element)

- 

don't know A[hi]
swap (A[mid], A[hi])



hi → ind where 2 should be placed
hi - 1
mid not incremented → bcoz it is not processed

- 

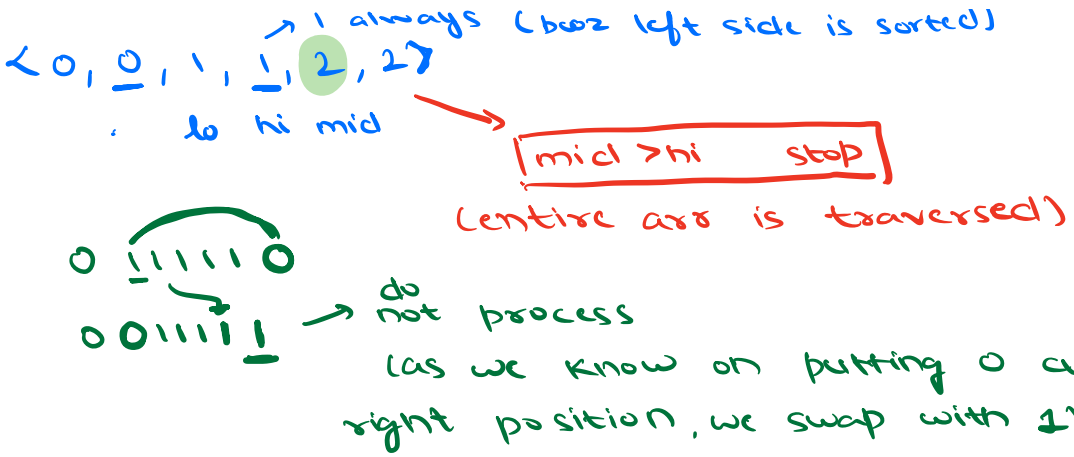
swap (A[mid], A[hi])
hi = hi - 1

- 

mid++

- 

swap (A[lo], A[mid])
lo++, mid++



Code

```
public ArrayList<Integer> sortCol
    (ArrayList<Integer> A) {

    int lo=0, hi=A.size()-1;
    for(int mid=0; mid<=hi; ){

        if(A.get(mid)==0){
            int temp = A.get(lo);
            A.set(lo,0);
            A.set(mid,temp);
            lo++;
            mid++;
        }
        else if(A.get(mid)==2){
            int temp = A.get(hi);
            A.set(hi,2);
            A.set(mid,temp);
            hi--;
        }
        else                    // A[mid]=1
            mid++;
    }
```

1 iteration

TC O(n)

SC O(1)

return Aij