Binary Search - 3

1.Problem Statement - Painter's Partition Problem – Given time only

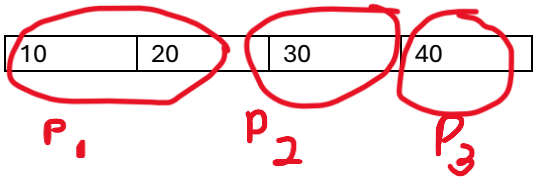
Given an array of N size. This array represents lengths of N boards. You need to paint all the boards using painters. [L1, L2, L3, L4] = [2, 3, 5, 1].

1. It takes 1 unit of time to paint 1 unit of length
2. Each board will not be divided among painters
3. A painter will only paint consecutive boards

Given the time you have to complete the job. (Time is fixed). Find out how many painters will be needed.

Example: A = | 10 | 20 | 30 | 40 | Time = 100; ans =1

1.A= T = | 10 | 20 | 30 | 40 | 50 unit of time; ans = 3



2.A = | 10 | 20 | 30 | 40 | T = 30 units; ans = Not possible

IDEA: Keep assigning boards to curr painter until workload <=T, if not then introduce a another painter. We want painter to work at its true potential → then only I will get min no. of painters required

Pseudocode

```
int minPainters(arr, T){
    int cnt = 1;                        //count of painter- for now
    int curr =0;                //curr – current workload of painter; currently painter doing 0 amount of work
    for(int i =0; i < A.length; i++){
        if(A[i] > T) return -1;
        if(curr +A[i] <=T){             //assign this workload to my current painter
            curr += A[i];
         }else{
            cnt++;                      //If workload of current painter > time; then assign new painter and new
            curr = A[i];                painter has workload starting from A[i]
    }
    return cnt;
 }
```

TC = O(N)

SC = O(1)

2.Problem Statement - Painter's Partition Problem – Given time and #of painters

Given the time you have to complete the job. Given #ofpainters. Find if these painters can complete the job in the given time.

1.Example: A =

10	20	30	40

T = 50, Painter = 5; ans = True

2. A=

10	20	30	40

T=50, P=2, ans = False

3. T =50, P=3, ans = True

Pseudocode

```
boolean isPossible(arr, T){
    int cnt =1;
    int curr =0;
    for(int i =0; i< A.length; i++){
        if(A[i] > T) return -1;
        if(curr + A[i] <= T){
            curr += A[i];
        }else{
            cnt++;
            curr = A[i];
        }
        if(P >= cnt) return true;          // P > than min #of painters required (cnt) return true
        else return False;
```

3.Problem Statement - Painter's Partition Problem – Given P & T – Find min T?

Given 2 integers A and B and an array of integers C of size N. Element C[i] represents the length of ith board. You have to paint all N boards [C0, C1, C2, C3 … CN-1]. There are A painters available and each of them takes B units of time to paint 1 unit of the board.

Calculate and return the minimum **time** required to paint all boards under the constraints that any painter will only paint contiguous sections of the board.

NOTE:
1. 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.
2. A painter will only paint contiguous boards. This means a configuration where painter 1 paints boards 1 and 3 but not 2 is invalid.
Return the ans % 10000003.

Problem Constraints

$1 <= A <= 1000$

$1 <= B <= 10^6$

$1 <= N <= 10^5$

$1 <= C[i] <= 10^6$

Output Format

Return minimum time required to paint all boards under the constraints that any painter will only paint contiguous sections of board % 10000003.

Example Input:

A = 10

B = 1

C = [1, 8, 11, 3]

Example Output: 11

Each block is painted by a painter so, Painter 1 paints block 1, painter 2 paints block 2, painter 3 paints block 3 and painter 4 paints block 4, time taken = max(1, 8, 11, 3) = 11

ans = 11 % 10000003

1.Example A = P | 20 | 10 | 30 | 40 | =1; ans =(20+10+30+40) = 100

2. A= [20, 10, 30, 40] P = 2; T=? ; ans =60

A = | 20 | 10 | 30 | 40 | P1 = 20; P2 = 80; **T = 80**

A = | 20 | 10 | 30 | 40 | P1 = 30; P2 = 70; **T = 70**

A = | 20 | 10 | 30 | 40 | P1 = 60; P2 = 40; **T = 60** *// 60 is the min time required given 2 painters*

How to understand that we can apply binary search on this question?

1. T =0 T = 50 T=100

If 50 is potential answer → update ans

go left (maybe 47, 48, 49 is possible)

If 50 is not possible then go right

2.So for a midpoint, we can go either left or right → so binary search is possible and can be applied on ans.

T = 0 1 2..... mid T = 100

False F F F F F ....F F T T T T T T ....T T T T

after sometime it will be possible. Here our goal is to find out 1st True

3.When we encounter a True → go left → to get best potential ans

When we encounter a False → go right

Whenever this kind of distribution is there → we have to apply binary search

Actual Code

l = max of the array

r = sum of all the elements in the array

```
public class Solution {
  public int paint(int A, int B, int[] C) {
    long l =0;// max element of an would be the min time required → in an array of [2, 5, 6, 7] → atleast 7 units of time is required
    long r = 0;//sum of all elements// max time required → l & r is the search space → my answer will lie in between l & r only
    long ans = 0;
    for(int i =0; i<C.length; i++){
      l = Math.max(C[i], l);
    }
    for(int i=0; i<C.length; i++){
      r +=C[i];
    }
    while( l <=r){
      long mid = l + (r-l)/2;
      boolean status = isPossible(C, mid, A);
      if(status == false){
        l = mid+1;
      }else{
        ans = mid;
        r = mid-1;
      }
    }
    return (int)((ans*B) % 10000003);
  }
  private boolean isPossible(int []A, long t, int P){
    int cnt =1;
    long curr=0;
    for(int i =0; i<A.length; i++){
      if(A[i] >t) return false;
      if(curr + A[i] <=t){
        curr += A[i];
      }else{
        cnt+=1;
        curr = A[i];
      }

    }
    if(P>=cnt){
      return true;
    }else{
      return false;
    }
  }

}
```

1.Our search Space: (max time – min time)

(sum of array – max array)

2. so in binary search, time complexity is log(search space) and not log(n)

TC = O(N * log(sum of array – max array) *//N is iteration taken by isPossible code*

SC = O(1)

Why ans * B?

- **ans:** The value of ans is the **minimum maximum time** in which the painters can paint all boards if each painter can paint at most ans units of time.
- **B:** The parameter B represents the time taken per unit of painting. To convert the number of units into the actual time required for painting, we multiply ans (units of time) by B (time per unit).

Dry run

A = [20, 10, 30, 40] P=3

l = 40 m = 70 r = 100

in 70 time, 3 painters can do the job → so updating ans to 70 and go left → r = mid-1

l = 40 m = 55 r = 70

can 3 painters do the job in T = 55 → Yes → update ans = 55 → go left

l = 40 m = 47 r = 54

yes → ans = 47 → go left

l = 40 m = 43 r = 46

yes → ans = 43 → go left

l = 40 m = 41 r = 42

yes → ans = 42 → go left

l = 40 m = 40 r = 40

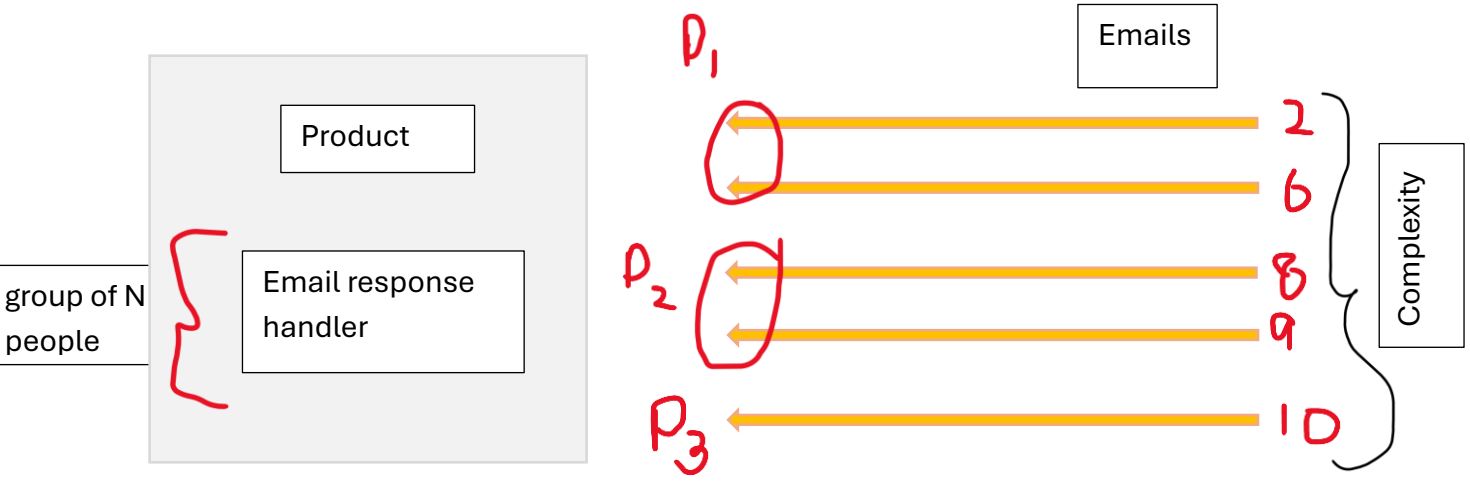
yes → ans = 40 → go left

l = 40 m = STOP coz l>r r = 39

One more Example of binary Search:

If you are working on a product and you are receiving too many emails for the project and you need email response handler. Lets say these are N people. Now you have to distribute these emails into the group of N people in a continuous manner. Each email have some complexity associated with them – meaning how much time it will take to resolve this email/ticket.

Now you have to distribute this job in such a manner that everyone will get the same amount of work and will get to go home early and together. So whosoever gets the max amount of work, you need to minimize that.



4.Problem Statement - Aggressive cows

Farmer John has built a new long barn with N stalls. Given an array of integers A of size N where each element of the array represents the location of the stall and an integer B which represents the number of cows.

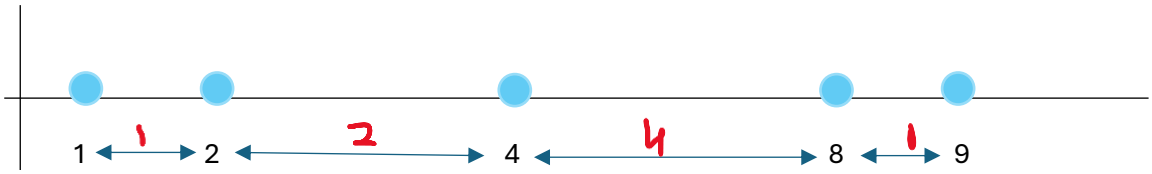
His cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, John wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Problem Constraints
2 <= N <= 100000
0 <= A[i] <= 10^9
2 <= B <= N

Output Format: Return the largest minimum distance possible among the cows.

Input 1:
A = [1, 2, 3, 4, 5]
B = 3
Input 2:
A = [1, 2]
B = 2
Output 1:
2
Output 2:
1
Explanation 1:
John can assign the stalls at location 1, 3 and 5 to the 3 cows respectively. So the minimum distance will be 2.
Explanation 2:
The minimum distance will be 1.

There are M stalls on X-axis and N cows where m>=n and stalls are given in a sorted array



Stalls are where you can place your cows. These cows don't like each other so they will be angry if you place them near to each other. The more near they are placed, more angry they would be.
Your goal is to maximize the min distance between any 2 cows in your arrangements. Closest cows should be as far as possible.

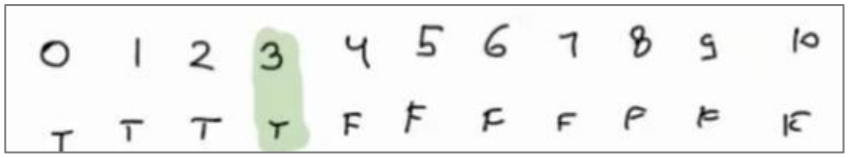
Idea -1 – wrong method

If we use strategy where if we have cows = 3 and 10 is the range, we can do (10-0)/4 = 2.5. this will fail when the stalls are arranged as below:
[1, 2, 3, 4, 5, 100]
Here 100/3 = 33 for cows = 10 → with this idea you will want 33 distance between the cows which is wrong because the best idea possible is 4 distance and 95 distance. so ans =4. This is greedy idea which doesnot work.

Idea -2 – Binary search

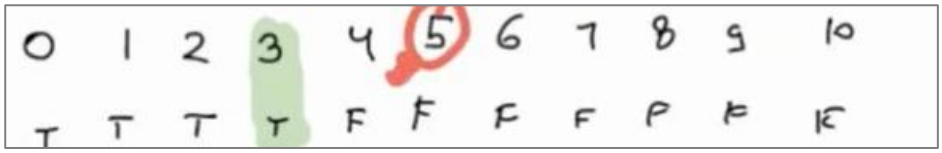
Lets assume max distance between the cows is 3 → definitely I can have 2 distance between the cows or 1 distance between the cows or 0 distance between the cows.

But if the max min distance possible is 3, then I cannot 4 distance between all my cows or 5 or 6…

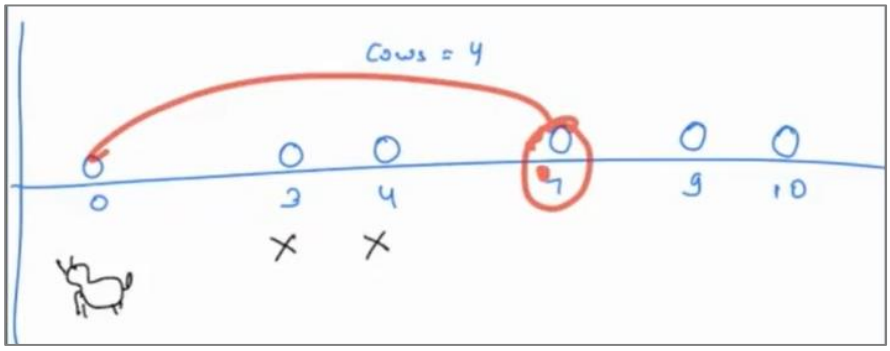


So if there is this distribution that means we can apply binary search and your goal is to find out the last true.

Also, if find out whether a particular distance such as 5, is possible between all my cows

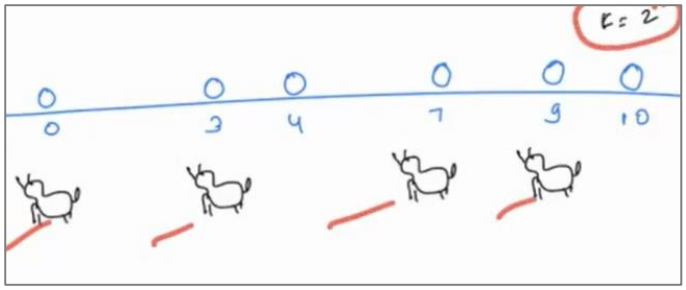


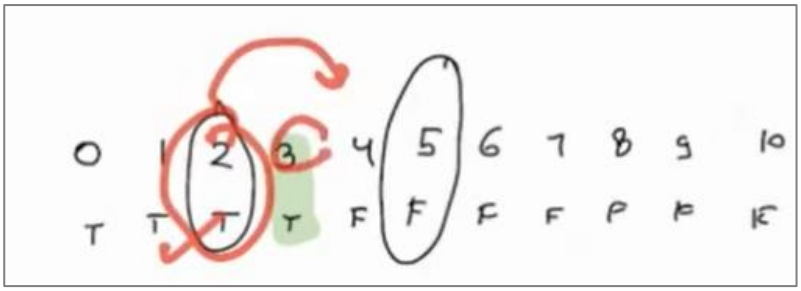
If distance = 5 and want to check whether I can place my cows at a 5 unit of distance. So I will place 1st cow at 0 and then 2nd cow at 7 index → since (0- 7) = 7 >5.



After that I cannot place my cow at 9th or 10th index since the distance between 7 to 9 or 7 to 10 is less than 5. so 5 distance is not possible for all 4 cows(only 2 cows were placed correctly). → the we have to the left side.

Now use distance = 2. Here I was able to place all my 4 cows





So distance 2 is True but I'll go right to find out my last True.

Actual Code

```
public class Solution {
  public int solve(int[] A, int B) {
    Arrays.sort(A);  // Sort stall locations

    int l = 1;  // Minimum possible distance
    int r = A[A.length - 1] - A[0];  // Maximum possible distance
    int ans = 0;

    while (l <= r) {
      int m = l + (r - l) / 2;

      if (isPossible(A, B, m)) {
        ans = m;  // Update answer
        l = m + 1;  // Try for a larger distance
      } else {
        r = m - 1;  // Reduce the search space
      }
    }

    return ans;
  }

  private boolean isPossible(int[] A, int B, int m) {
    int count = 1;  // Place the first cow in the first stall
    int lastPos = A[0];  // The position of the last cow placed

    for (int i = 1; i < A.length; i++) {
      if (A[i] - lastPos >= m) {
        count++;
        lastPos = A[i];  // Place another cow
      }
      if (count >= B) {  // Successfully placed all cows
        return true;
      }
    }
    return false;  // Couldn't place all cows with minimum distance m
  }
}
```

1.Problem Statement - Painter's Partition ProblemSolved

Problem Description

Given 2 integers **A** and **B** and an array of integers **C** of size **N**. Element **C[i]** represents the length of **i^{th}** board.

You have to paint all **N** boards **[C_0, C_1, C_2, C_3 ... C_{N-1}]**. There are **A** painters available and each of them takes **B** units of time to paint **1 unit** of the board.

Calculate and return the minimum time required to paint all boards under the constraints that **any painter will only paint contiguous sections of the board.**

NOTE:

1. 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.

2. A painter will only paint contiguous boards. This means a configuration where painter 1 paints boards 1 and 3 but not 2 is invalid.

Return the **ans % 10000003**.

Problem Constraints

1 <= A <= 1000

1 <= B <= 10^6

1 <= N <= 10^5

1 <= C[i] <= 10^6

Input Format

The first argument given is the integer A.

The second argument given is the integer B.

The third argument given is the integer array C.

Output Format

Return minimum time required to paint all boards under the constraints that any painter will only paint contiguous sections of board % 10000003.

Example Input

Input 1:

A = 2

B = 5

C = [1, 10]

Input 2:

A = 10

B = 1

C = [1, 8, 11, 3]

Example Output

Output 1: 50

Output 2: 11

Example Explanation

Explanation 1:

Possibility 1:- One painter paints both blocks, time taken = 55 units.

Possibility 2:- Painter 1 paints block 1, painter 2 paints block 2, time take = max(5, 50) = 50

There are no other distinct ways to paint boards.

ans = 50 % 10000003

Explanation 2:

Each block is painted by a painter so, Painter 1 paints block 1, painter 2 paints block 2, painter 3 paints block 3
and painter 4 paints block 4, time taken = max(1, 8, 11, 3) = 11
ans = 11 % 10000003

Actual Code

```
public class Solution {
  public int paint(int A, int B, int[] C) {
    long l =0, r = 0, ans = 0;
    for(int i =0; i<C.length; i++){
      l = Math.max(C[i], l);
    }
    for(int i=0; i<C.length; i++){
      r +=C[i];
    }
    while( l <=r){
      long mid = l + (r-l)/2;
      boolean status = ispossible(C, mid, A);
      if(status == false){
        l = mid+1;
      }else{
        ans = mid;
        r = mid-1;
      }
    }
    return (int)((ans*B) % 10000003);
  }
  private boolean ispossible(int []A, long mid, int P){
    int cnt =1;
    long curr=0;
    for(int i =0; i<A.length; i++){
      if(A[i] >mid) return false;
      if(curr + A[i] <=mid){
        curr += A[i];
      }else{
        cnt+=1;
        curr = A[i];
      }

    }
    if(P>=cnt){
      return true;
    }else{
      return false;
    }
  }

}
```

2. Problem Statement - Allocate Books

Problem Description

Given an array of integers **A** of size **N** and an integer **B**.

The College library has **N** books. The **ith** book has **A[i]** number of pages.

You have to allocate books to **B** number of students so that the maximum number of pages allocated to a student is minimum.

A book will be allocated to **exactly one student**.

Each student has to be allocated **at least one book**.

Allotment should be in **contiguous order**, for example: A student cannot be allocated book 1 and book 3, skipping book 2.

Calculate and return that **minimum** possible number.

NOTE: Return **-1** if a valid assignment is not possible.

Problem Constraints

$1 <= N <= 10^5$

$1 <= A[i], B <= 10^5$

Input Format

The first argument given is the integer array A.

The second argument given is the integer B.

Output Format

Return that minimum possible number.

Example Input

Input 1:

A = [12, 34, 67, 90]

B = 2

Input 2:

A = [12, 15, 78]

B = 4

Example Output

Output 1:

113

Output 2:

-1

Example Explanation

Explanation 1:

There are two students. Books can be distributed in following fashion :

1) [12] and [34, 67, 90]
   Max number of pages is allocated to student 2 with 34 + 67 + 90 = 191 pages
2) [12, 34] and [67, 90]
   Max number of pages is allocated to student 2 with 67 + 90 = 157 pages
3) [12, 34, 67] and [90]
   Max number of pages is allocated to student 1 with 12 + 34 + 67 = 113 pages
   Of the 3 cases, Option 3 has the minimum pages = 113.

Explanation 2:

Each student has to be allocated at least one book.

But the Total number of books is **less than** the number of students.
Thus each student cannot be allocated to atleast one book.
Therefore, the result is **-1**.

Actual Code

```
public class Solution {
  public int books(ArrayList<Integer> A, int B) {
    int l = 0, r= 0, ans =-1;
    if(B > A.size()) return -1;
    for(int i =0; i<A.size(); i++){
      l = Math.max(l, A.get(i));
      r +=A.get(i);
    }
    while(l<=r){
      int mid = l +(r-l)/2;
      boolean status = isPossible(A, B, mid);          // B = #of students; mid = min#of pages allocated
      if(status == false){

        l = mid+1;                                     //moving l towards books having more #of pages since the array is sorted
      }else{
        ans = mid;
        r = mid-1;
      }
    }
    return ans;
  }
  private boolean isPossible(ArrayList<Integer>A, int B, int mid){
    int cnt = 1;                                       //cnt = Initial #of students
    int curr =0;                                       //curr = Initial #of pages
    for(int i =0; i<A.size(); i++){
      if(A.get(i)>mid) return false;
      if((curr + A.get(i)) <= mid){
        curr +=A.get(i);
      }else{
        cnt++;
        curr = A.get(i);
      }
    }
    if(B >=cnt){
      return true;
    }
    return false;
  }
}
```

3.Problem Statement - Special Integer

Problem Description

Given an array of integers **A** and an integer **B**, find and return the maximum value K such that there is no subarray in A of size K with the sum of elements greater than B.

Problem Constraints

1 <= |A| <= 100000

1 <= A[i] <= 10^9

1 <= B <= 10^9

Input Format

The first argument given is the integer array A.

The second argument given is integer B.

Output Format

Return the maximum value of K (sub array length).

Example Input

Input 1:

A = [1, 2, 3, 4, 5]

B = 10

Input 2:

A = [5, 17, 100, 11]

B = 130

Example Output

Output 1:

2

Output 2:

3

Example Explanation

Explanation 1:

For K = 5, There are subarrays [1, 2, 3, 4, 5] which has a sum > B

For K = 4, There are subarrays [1, 2, 3, 4], [2, 3, 4, 5] which has a sum > B

For K = 3, There is a subarray [3, 4, 5] which has a sum > B

For K = 2, There were no subarray which has a sum > B.

Constraints are satisfied for maximal value of 2.

Explanation 2:

For K = 4, There are subarrays [5, 17, 100, 11] which has a sum > B

For K = 3, There were no subarray which has a sum > B.

Constraints are satisfied for maximal value of 3.

Actual Code

```
public class Solution {
  public int solve(ArrayList<Integer> A, int B) {
    int l = 1, r = A.size(), ans =0;

    while(l<=r){
      int mid = l+(r-l)/2;
```

```
        if(isPossible(A, B, mid)){
            l = mid +1;
            ans = mid;
        }else{
            r = mid-1;
        }
    }
    return ans;
  }
  private boolean isPossible(ArrayList<Integer>A, int B, int mid){
    int k =mid;
    long sum=0;
    for(int i = 0; i<k; i++){
      sum+=A.get(i);
    }
    if(sum > B) return false;
    for(int i =k; i<A.size(); i++){
      sum += A.get(i) - A.get(i-k);   //sliding window
      if(sum > B) return false;
    }
    return true;
  }

}
```