Competitive Coding @ Berkeley

Decal Lecture #5 - Binary Search The Answer

Welcome Back!

- Today's topic will be binary search and some clever ways we can use it
- As always, the problemset is open and we will cover the first two problems
    - Feel free to give those a look

Announcements

- ICPC Regionals happened last weekend!

ICPC Results

Berkeley as a whole had an extremely impressive performance, with all six teams getting in the top 15, and the top 4 teams getting in the top 6. First UC Berkeley win since 2017!

Rank	Name	Solved	Time	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Golden State Geeks (UC Berkeley) (California - D1)	9	1304	1/110	1/242	1/18	2/53	2/174	1/68	0/--	1/191	0/--	1/11	6/297	5/--	0/--
2	Cardinal (Stanford) (California - D1)	8	1235	1/116	0/--	1/26	3/219	3/295	2/36	0/--	4/95	0/--	1/24	1/264	0/--	0/--
3	Burden of Dreams (UC Berkeley) (California - D1)	8	1319	7/196	3/242	1/10	1/168	0/--	2/37	0/--	2/269	0/--	1/14	1/183	0/--	0/--
4	Berkeley Blue (UC Berkeley) (California - D1)	7	1049	1/71	2/--	1/25	0/--	2/240	4/220	1/184	0/--	0/--	2/49	1/160	0/--	0/--
5	1-gon di-gon dra-gon (U of Washington) (Washington - D1)	7	1057	1/169	4/102	1/18	3/218	0/--	4/81	0/--	2/278	0/--	1/11	1/--	0/--	0/--
6	Berkeley Radicals (UC Berkeley) (California - D1)	7	1065	1/102	0/--	1/24	0/--	2/290	1/67	0/--	2/279	0/--	1/24	3/199	1/--	0/--
7	simpl. lia. auto. (U of Washington) (Washington - D1)	7	1125	1/50	0/--	1/69	4/--	0/--	4/97	0/--	4/175	0/--	1/11	6/277	5/146	0/--
8	conqueror_of_pacnw (Stanford) (California - D1)	6	798	2/126	2/283	1/36	0/--	0/--	1/81	0/--	1/206	0/--	1/26	0/--	0/--	0/--
9	Cash Cow Heresy (UC Davis) (California - D1)	6	813	0/--	1/203	1/23	0/--	1/102	4/107	0/--	2/287	0/--	1/11	0/--	0/--	0/--
10	DA Flint (De Anza) (California - D1)	6	833	1/152	0/--	1/28	2/213	0/--	2/113	0/--	1/221	0/--	2/46	7/--	0/--	0/--

Motivating Problem

- We have a **sorted list** ***a*** of ***unique integers*** and want to be able to find the position of a specific element ***x***
    - If ***x*** does not appear, return ***-1***

Motivating Problem

- We have a **sorted list** ***a*** of ***unique integers*** and want to be able to find the position of a specific element ***x***
    - If ***x*** does not appear, return ***-1***
- Easiest solution:
    - Iterate through all indices and return the current index ***i*** if a_i == ***x***
    - Return -1 if we never found it

Motivating Problem

- We have a **sorted list** ***a*** of ***unique integers*** and want to be able to find the position of a specific element ***x***
  - If ***x*** does not appear, return ***-1***
- Easiest solution:
  - Iterate through all indices and return the current index ***i*** if a_i == ***x***
  - Return -1 if we never found it
  - O(***n***) runtime where ***n*** is the number of elements of ***a***
  - **Does not take advantage of the fact that our array is sorted!**

Can we do better?

- Let’s say we want to find the word “program” in a dictionary, how would we do it?

Can we do better?

- Let's say we want to find the word "program" in a dictionary, how would we do it?
- Our O(n) solution would tell us to start from the **first word** on the **first page** and move on **one word at a time**

Can we do better?

- Let's say we want to find the word "program" in a dictionary, how would we do it?
- Our O(n) solution would tell us to start from the **first word** on the **first page** and move on **one word at a time**
- Instead, we go to the middle of the dictionary and see if "program" is on the left or right half
    - Then we go to the middle of that section and repeat until we find the word!

You invented binary search!

- Finding a word in a dictionary is basically just **binary search** on a book
- **Binary search** is an **efficient way to find a target in a sorted list**
    - At each step, we eliminate half of the search space giving us a runtime of **O(log n)**

Binary Search In Code

- We will track two pointers that mark the endpoints of where the word can be
    - At first, the left pointer ***l*** is on the first element and the right pointer ***r*** on the last element

Binary Search In Code

- We will track two pointers that mark the endpoints of where the word can be
  - At first, the left pointer ***l*** is on the first element and the right pointer ***r*** on the last element
- Calculate ***m***, the midpoint as ***m = (l + r) / 2***
- Compare our desired element ***x*** to the midpoint ***m***

Binary Search In Code

- We will track two pointers that mark the endpoints of where the word can be
  - At first, the left pointer ***l*** is on the first element and the right pointer ***r*** on the last element
- Calculate ***m***, the midpoint as ***m = (l + r) / 2***
- Compare our desired element ***x*** to the midpoint ***m***
  - If ***x < m***, then we know **x** is in the left half
    - Set ***r*** to ***m - 1***
  - If ***x > m***, then we know that ***x*** is in the right half
    - Set ***l*** to ***m + 1***
  - Otherwise is ***x == m***, we've found it!

What if the target doesn't exist in the list?

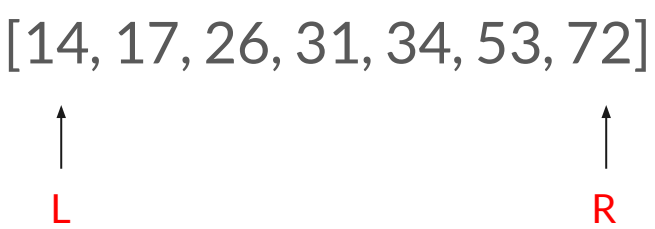
- If the target does not exist in the list, the two pointers ***l*** and ***r*** will eventually "cross" each other
  - This means ***l*** will be on the right of ***r***

What if the target doesn't exist in the list?

- If the target does not exist in the list, the two pointers ***l*** and ***r*** will eventually "cross" each other
  - This means ***l*** will be on the right of ***r***
- We want to continue the search while ***l <= r***
  - If the target was never found and we exit our condition ***while l <= r***, then it doesn't exist

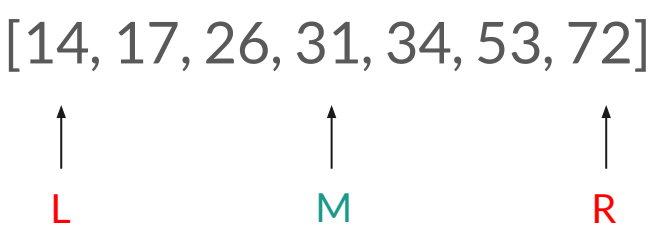
Binary Search Example

- We want to find 26



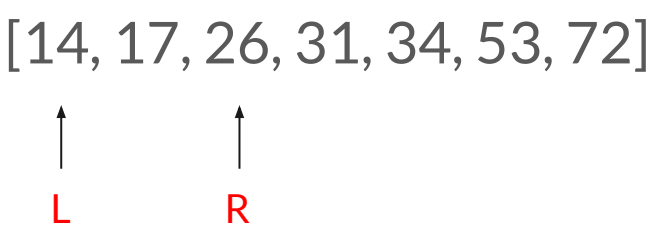
Binary Search Example

- We want to find 26



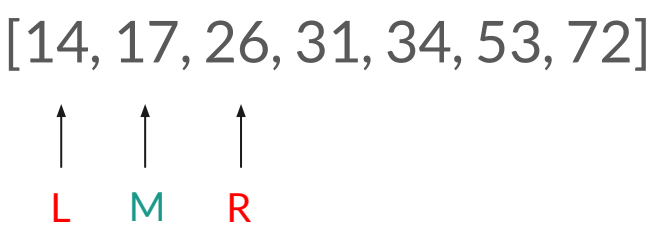
Binary Search Example

- We want to find 26



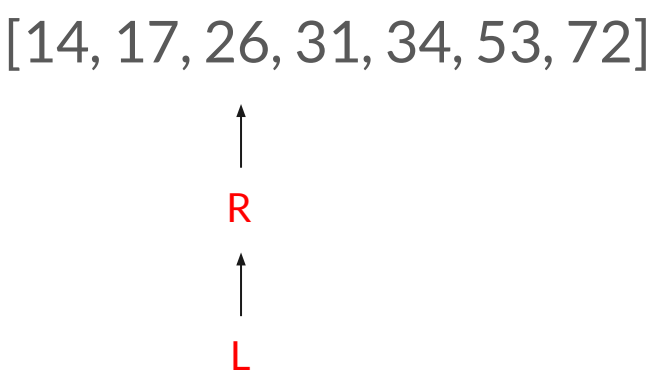
Binary Search Example

- We want to find 26



Binary Search Example

- We want to find 26



Binary Search Example

- We want to find 26

[14, 17, 26, 31, 34, 53, 72]

R

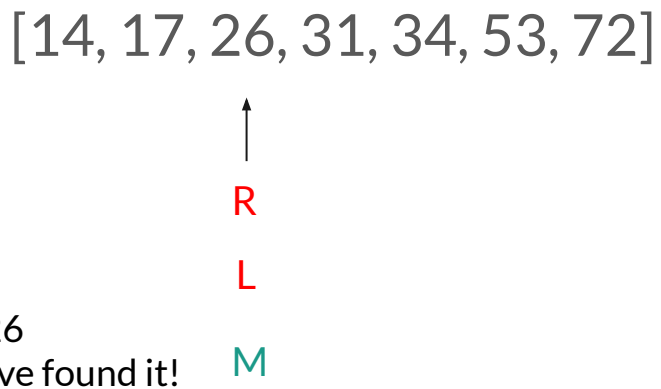
L

M

- All pointers are on 26
- M is on 26, so we have found it!

Binary Search Example

- We want to find 26



- All pointers are on 26
- M is on 26, so we have found it!
- If we had 25 here instead of 26…
  - The left pointer would move over one spot to the right
  - The binary search would terminate since the left and right “crossed”

Binary Searching A “Monotonic Function”

- A **monotonic increasing function** is a function that is always nondecreasing on its domain
  - For all input x, **f(x+1) ≥ f(x)**

Binary Searching A “Monotonic Function”

- A **monotonic increasing function** is a function that is always nondecreasing on its domain
  - For all input x, **f(x+1) ≥ f(x)**
- For the sake of binary searching the answer, we will only care about functions that return **boolean values**
  - **f(x) will return either a 0 or 1**

Binary Searching A “Monotonic Function”

- We can apply binary search to find the **smallest** (or sometimes greatest) **values that satisfy a given condition**
- Problem: Find the value of the first element in a list that is greater than a given value ***x***
    - In other words, we want the first index that satisfies $a_i > x$

Binary Searching A “Monotonic Function”

- Problem: Find the value of the first element in a sorted list that is greater than a given value x
  - In other words, we want the first index that satisfies $a_i > x$
- Example $x = 11$ and $a = [2, 5, 6, 9, 10, 11, 12, 15, 19, 20, 51]$
- Construct an array b such that
  - $b_i = 1$ if $a_i > x$
  - $b_i = 0$ otherwise

Binary Searching A “Monotonic Function”

- Problem: Find the value of the first element in a sorted list that is greater than a given value ***x***
  - In other words, we want the first index that satisfies $a_i > x$
- Example ***x*** = 11 and ***a*** = **[2, 5, 6, 9, 10, 11, 12, 15, 19, 20, 51]**
- Construct an array ***b*** such that
  - $b_i = 1$ if $a_i > x$
  - $b_i = 0$ otherwise
- ***a*** = `[2 , 5 , 6 , 9 , 10, 11, 12, 15, 19, 20, 51]`
- ***b*** = `[0 , 0 , 0 , 0,  0,  0,  1,  1,  1,  1,  1]`

Binary Searching A “Monotonic Function”

- *x = 11*
- *a* = [2 , 5 , 6 , 9 , 10, 11, 12, 15, 19, 20, 51]
- *b* = [0 , 0 , 0 , 0, 0, 0, 1, 1, 1, 1, 1]
- We can binary search to find the first value that is true for the desired condition
- Calculate our midpoint index ***m*** as ***m = (l + r) / 2*** just like before
- Keep a global answer variable ***ans*** to save the best candidate throughout the binary search

Binary Searching A “Monotonic Function”

- ***x = 11***
- ***a*** = [2 , 5 , 6 , 9 , 10, 11, 12, 15, 19, 20, 51]
- ***b*** = [0 , 0 , 0 , 0, 0, 0, 1, 1, 1, 1, 1]
- We can binary search to find the first value that is true for the desired condition
- Calculate our midpoint index ***m*** as ***m = (l + r) / 2*** just like before
- Keep a global answer variable ***ans*** to save the best candidate throughout the binary search
    - If index ***m*** is **True**, assign ***ans*** to ***m*** and set ***R*** to ***m-1*** because it’s possible there is a smaller **True** index!
    - If index ***m*** is **False**, set ***L*** to ***m+1*** because we need to look right to get a **True** index

Useful Binary Search Functions

- Upper_bound: returns the **first index** i such that $a_i > x$
- Lower_bound: returns the **first index** i such that $a_i \geq x$

Useful Binary Search Functions

- Upper_bound: returns the **first index** ***i*** such that $a_i > x$
- Lower_bound: returns the **first index** ***i*** such that $a_i \geq x$
- # of elements in an array equal to ***x***
  - upper_bound(x) - lower_bound(x)
- How to find last index ***i*** such that $a_i \leq x$?
  - https://cses.fi/problemset/task/1091/

Useful Binary Search Functions

- Upper_bound: returns the **first index** i such that $a_i > x$
- Lower_bound: returns the **first index** i such that $a_i \geq x$
- # of elements in an array equal to x
  - upper_bound(x) - lower_bound(x)
- How to find last index i such that $a_i \leq x$?
  - https://cses.fi/problemset/task/1091/
  - Upper_bound(x) - 1
- Lower bound doesn't work. Let a person's max ticket price is 5
  - a = [4, 5, 6]
  - a = [4, 6]

Ready to use implementations?

- Yes!
- In Python
  - **bisect.bisect_left** (basically lower_bound)
  - **bisect.bisect_right** (basically upper_bound)
- In C++
  - **std::upper_bound**
  - **std::lower_bound**

Binary Searching For An Answer

- Many answers that we need to find in problems are actually monotonic!
- https://codeforces.com/problemset/problem/279/B

When Valera has got some free time, he goes to the library to read some books. Today he's got t free minutes to read. That's why Valera took n books in the library and for each book he estimated the time he is going to need to read it. Let's number the books by integers from 1 to n. Valera needs a_i minutes to read the i-th book.

Valera decided to choose an arbitrary book with number i and read the books one by one, starting from this book. In other words, he will first read book number i, then book number $i + 1$, then book number $i + 2$ and so on. He continues the process until he either runs out of the free time or finishes reading the n-th book. Valera reads each book up to the end, that is, he doesn't start reading the book if he doesn't have enough free time to finish reading it.

Print the maximum number of books Valera can read.

Input

The first line contains two integers n and t ($1 \le n \le 10^5$; $1 \le t \le 10^9$) — the number of books and the number of free minutes Valera's got. The second line contains a sequence of n integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^4$), where number a_i shows the number of minutes that the boy needs to read the i-th book.

Output

Print a single integer — the maximum number of books Valera can read.

Examples

input	Copy
4 5 3 1 2 1	
output	Copy
3	

input	Copy
3 3 2 2 3	
output	Copy
1	

Binary Searching For An Answer

- Many answers that we need to find in problems are actually monotonic!
- https://codeforces.com/problemset/problem/279/B
- Easiest solution is to check all number of books he could read from 0 to ***n***
    - For all lengths, check each segment of books of that length

Binary Searching For An Answer

- Many answers that we need to find in problems are actually monotonic!
- https://codeforces.com/problemset/problem/279/B
- Easiest solution is to check all number of books he could read from 0 to ***n***
  - For all lengths, check each segment of books of that length
  - Far too slow, $O(n^3)$ time complexity
    - We have to try all segments lengths 0 to ***n***
    - For each segment length to check, we're checking around ***n*** segments
    - Calculating sum on a segment is also O(n) time

Binary Searching For An Answer

- Remember that we can actually find any range sum in O(1) with some precalculation!
- We will make a prefix sum array in O(n) time first

$$A = [3, 1, 4, 1, 5, 9, 2, 6]$$

$$P = [3, 4, 8, 9, 14, 23, 25, 31]$$

Binary Searching For An Answer

- Remember that we can actually find any range sum in O(1) with some precalculation!
- We will make a prefix sum array in O(n) time first
- This brings our time complexity down to $O(n^2)$ which is still too slow

Binary Searching For An Answer

- Remember that we can actually find any range sum in O(1) with some precalculation!
- We will make a prefix sum array in O(n) time first
- This brings our time complexity down to $O(n^2)$ which is still too slow
- Observe that the number of books able to be read is a monotonic function!
    - If we can read 5 consecutive books, we should also be able to read any amount less than 5
    - If we cannot read 5 consecutive books, there's no way we could read any amount higher

Binary Searching For An Answer

- Rather than trying all book segment lengths from 0 to ***n***
  - We can binary search over the interval [0, ***n***]

Binary Searching For An Answer

- Rather than trying all book segment lengths from 0 to ***n***
  - We can binary search over the interval [0, ***n***]
- Imagine it in terms of our boolean array where b_i is **True** if we can read ***i*** books
- *a* = `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
- *b* = `[T, T, T, T, T, F, F, F, F, F]`
  - We're now looking for the last index that returns **True**

Binary Searching For An Answer

- Rather than trying all book segment lengths from 0 to ***n***
  - We can binary search over the interval [0, ***n***]
- Very similar as before, keep a global answer variable ***ans***
- Calculate our midpoint index ***m*** as ***m = (l + r) / 2***
- Keep a global answer variable ***ans*** to save the best candidate throughout the binary search

Binary Searching For An Answer

- Rather than trying all book segment lengths from 0 to ***n***
  - We can binary search over the interval [0, ***n***]
- Very similar as before, keep a global answer variable ***ans***
- Calculate our midpoint index ***m*** as ***m = (l + r) / 2***
- Keep a global answer variable ***ans*** to save the best candidate throughout the binary search
  - If index ***m*** is **True**, assign ***ans*** to ***m*** and set ***L*** to ***m+1*** because it's possible we can read more books
  - If index ***m*** is **False**, set ***R*** to ***m-1*** because we need to try reading less books

```python
n, t = [int(num) for num in input().split()]
a = [int(num) for num in input().split()]
p = [0]
running_sum = 0
for i in range(n):
    running_sum += a[i]
    p.append(running_sum)

l, r = 0, n
def valid(m):
    ok = 0
    for i in range(m, n+1):
        if(p[i] - p[i-m] <= t):
            ok = 1
    return ok

ans = -1
while l <= r:
    m = (l+r)//2
    if valid(m):
        ans = m
        l = m + 1
    else:
        r = m - 1
print(ans)
```

Binary Search The Answer

- To wrap it up…
  - Always consider if your problem is monotonic!
  - This observation alone can turn a brute force solution into a working one

Binary Search The Answer

- To wrap it up…
  - Always consider if your problem is monotonic!
  - This observation alone can turn a brute force solution into a working one
  - Although binary search can be the intended solution on its own, you can very frequently create **alternative O(n log n) solutions with binary search**

Maximum Median

You are given an array a of n integers, where n is odd. You can make the following operation with it:

- Choose one of the elements of the array (for example a_i) and increase it by 1 (that is, replace it with $a_i + 1$).

You want to make the median of the array the largest possible using at most k operations.

The median of the odd-sized array is the middle element after the array is sorted in non-decreasing order. For example, the median of the array $[1, 5, 2, 3, 5]$ is 3.

Input

The first line contains two integers n and k ($1 \le n \le 2 \cdot 10^5$, n is odd, $1 \le k \le 10^9$) — the number of elements in the array and the largest number of operations you can make.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

Output

Print a single integer — the maximum possible median after the operations.

Examples

input	Copy
3 2 1 3 5	

output	Copy
5	

Maximum Median

- If we can figure out whether or not it's possible to make the median of the array X for a given X, we can solve the problem using binary search
- We can greedily find whether or not a given X is possible, by increasing every element the second half of the array up to X, and seeing if this can be done with K operations
- This is monotonic, because if a given X is possible, X - 1 must also be possible (unless X = median of original array), and if a given X is impossible, X + 1 must also be impossible, so we can binary search over this.

Cellular Network

You are given n points on the straight line — the positions (x-coordinates) of the cities and m points on the same line — the positions (x-coordinates) of the cellular towers. All towers work in the same way — they provide cellular network for all cities, which are located at the distance which is no more than r from this tower.

Your task is to find minimal r that each city has been provided by cellular network, i.e. for each city there is at least one cellular tower at the distance which is no more than r.

If $r = 0$ then a tower provides cellular network only for the point where it is located. One tower can provide cellular network for any number of cities, but all these cities must be at the distance which is no more than r from this tower.

Examples

input	Copy
3 2 -2 2 4 -3 0	
output	Copy
4	

Cellular Network

- Similarly to the maximum median problem, if we can check whether or not a given R is possible, we can solve the problem by binary searching for this value of R
- We can check whether or not an R is possible using a two-pointers approach, with both pointers going to the right, after sorting the lists of cities and towers in ascending order.
- This function will also be monotonic, because if a given R is possible, R + 1 is also possible, and if a given R is impossible, R - 1 is impossible.

Random Tip For C++ Users

- **THIS ONLY APPLIES TO C++**
- If you need to use lower_bound or upper_bound on a hashset, do not do it like this!
  - i = lower_bound(set.begin(), set.end(), x)
  - This will take O(n) time rather than O(log n) because std::lower_bound is not meant to handle hashset iterators (or hashmap)
- Instead, these data structures have their own implementations
- For set<int> s or map<int, int> s
  - s.lower_bound(x)

ICPC Regionals Resolver

- https://www.youtube.com/watch?v=JurNfJLX-lI

Thanks for Coming!

Special Calforces round right after this!

Participate in our Lockout Tournament next weekend