① Search in Nearly Sorted Array.

	10	20	30	40	50	60	70	80		

for k=1 → Nearly Sorted array looks like

	20	10	40	30	50	60	70	80	90	k=1
idx	0	1	2	3	4	5	6	7	8	target = 80

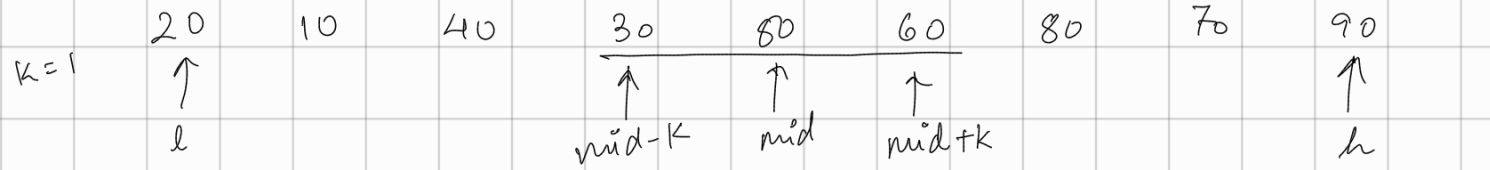
low = 0, high = 8, mid = low + (high - low)/2;
mid = 4

check for element in [mid-k, mid+k]
if found, return its index
if not, update low & high
here 80 > max [mid-k, mid+k] (60)
so update low = mid + k + 1

now low = 6, high = 8, mid = 7
check in [mid - k, mid + k]
now at mid - 1 = 80
So return mid-1 & found

Pseudo code.

low = 0, high = n-1
while (l <= h) {
if (check from [mid-k, mid+k] == true)
found
else update with
h = mid-k-1 or l = mid+k+1
accordingly

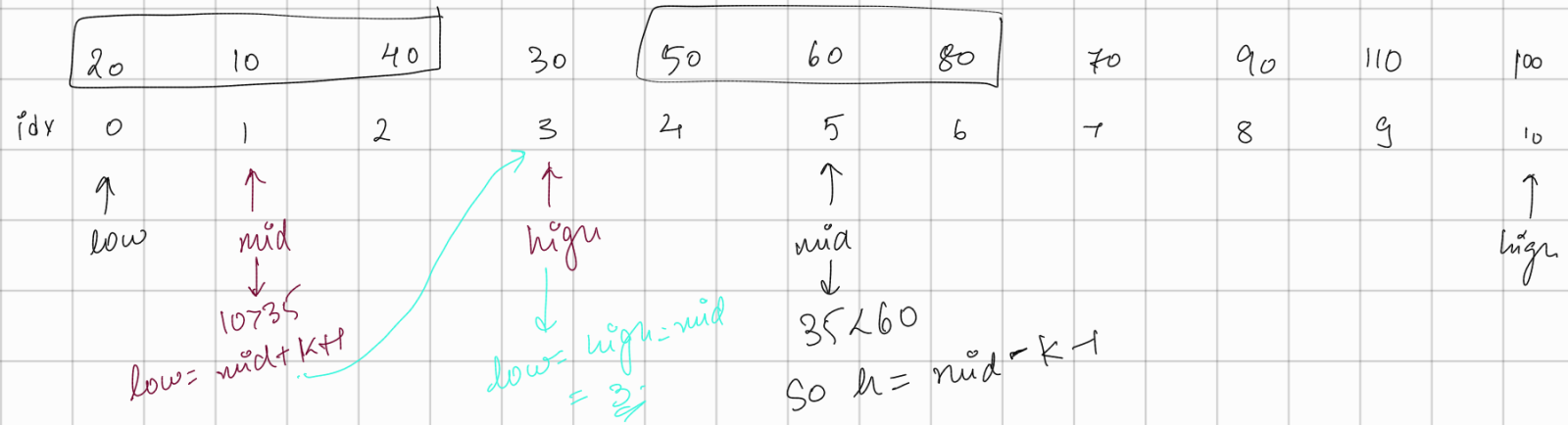


target = 20

low	high	mid	mid-k	mid+k	found?
0	8	4	3	5	X
0	2	1	0	2	✓

Time Complexity ⇒ $O((2K+1) \times \log_2 n)$ ← better than linear

target = 35 , k = 1



CODE:

```
int low=0, high= arr.length-1;
while (low<= high){
    int mid= low + (high-low)/2;
    if (arr[mid]== K)
        return mid;
    else if ( mid>0 && arr[mid-1]==K)
        return mid-1;
    else if ( mid<n-1 && arr[mid+1]== k)
        return mid+1;
    else if ( arr[mid]<K)  high= mid+2;
    else    low= mid-2;
}
```

return -1;

(2) Jump Search (b/w Linear & Binary)

target = 140

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160

window1 : K>10, K>60 X X ✓ → here apply linear search

Take a window of some size & check if element can be in that window if it can exist, linear search in window else change window

for window size → k & no. of elements = n

Best Case: O(1) → O(k) ; first window

Worst Case: O(windows + window size) = O(⌈n/k⌉ + k)

time to skip all (n/k) windows.

to check in a window → constant work

ceil: ⌈ ⌉

To get best time complexity: take $k = \sqrt{N}$

Worst Case: $\frac{N}{\sqrt{N}} + \sqrt{N} = 2\sqrt{N}$

∴ Worst case ⇒ $O(\sqrt{N})$

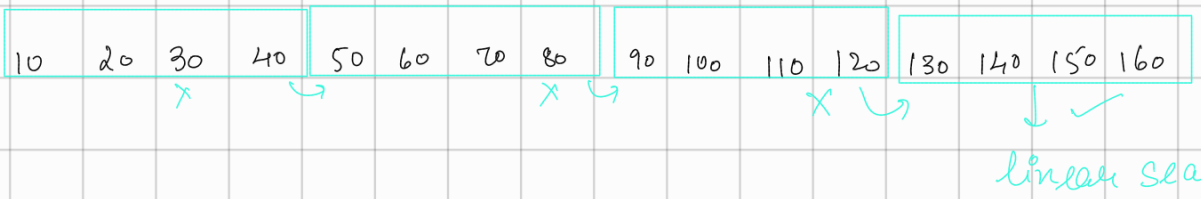
for global minima

To minimize $f(k) = (N/k) + (k) \rightarrow f'(N) = 0$

$\frac{df}{dk} \Rightarrow \frac{-N}{k^2} + 1 = 0 \Rightarrow k^2 = N \rightarrow \boxed{k = \sqrt{N}}$

So for Jump Search, use window size = $\sqrt{N}$

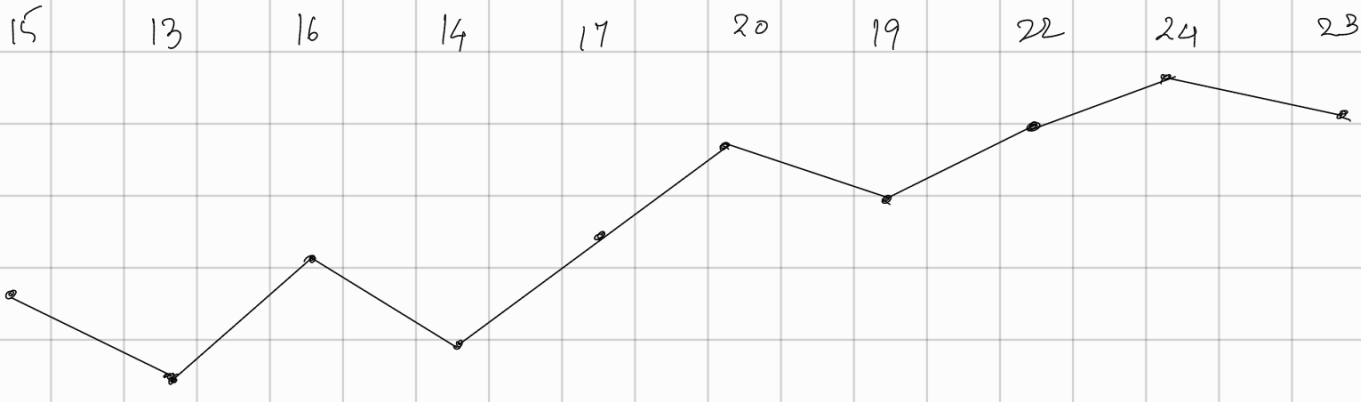
target = 140



③ Step Search. { Element / value → Property }

{ Index → Property }

Eg; K = 3. target = 28 abs(arr[i] − arr[i−1]) <= 3 ∀ i ∈ [2, N]

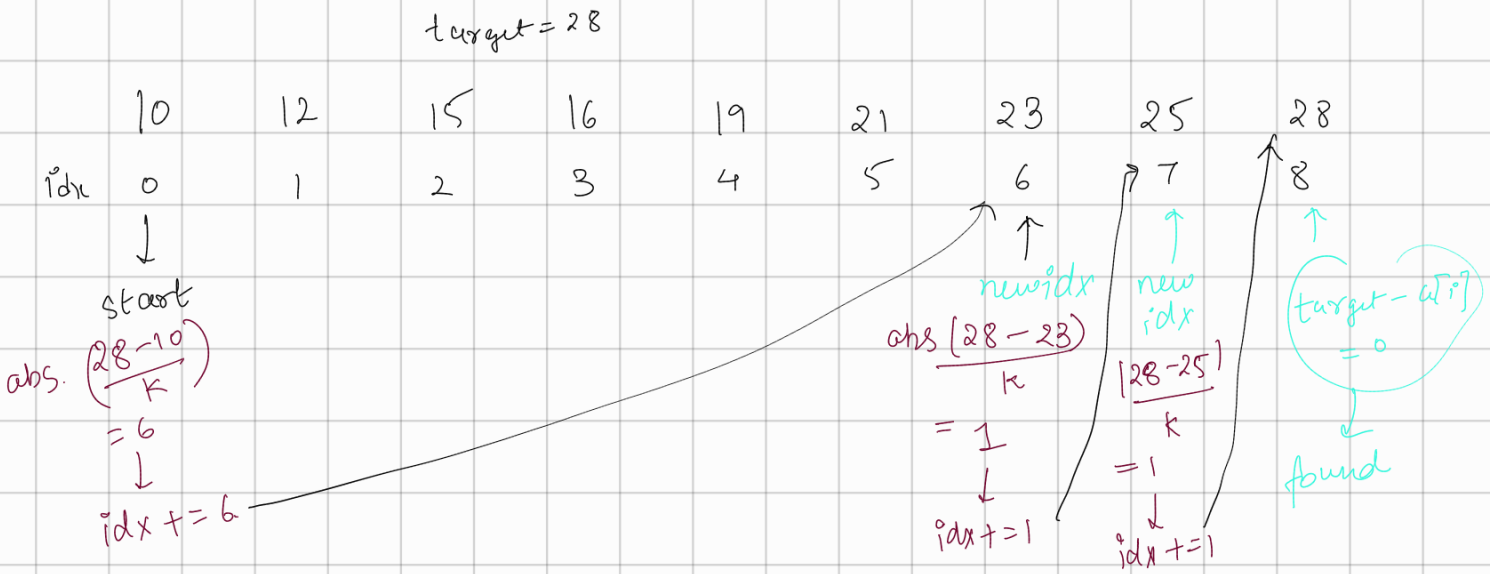


to search 28 start from 10 & move $\left(\frac{28-10}{K}\right)$ steps ahead. because we cannot get 28 for this amount of steps any way.

In given eg: (28 − 10)/3 = 6 → So skip (6 − 1) = 5 steps

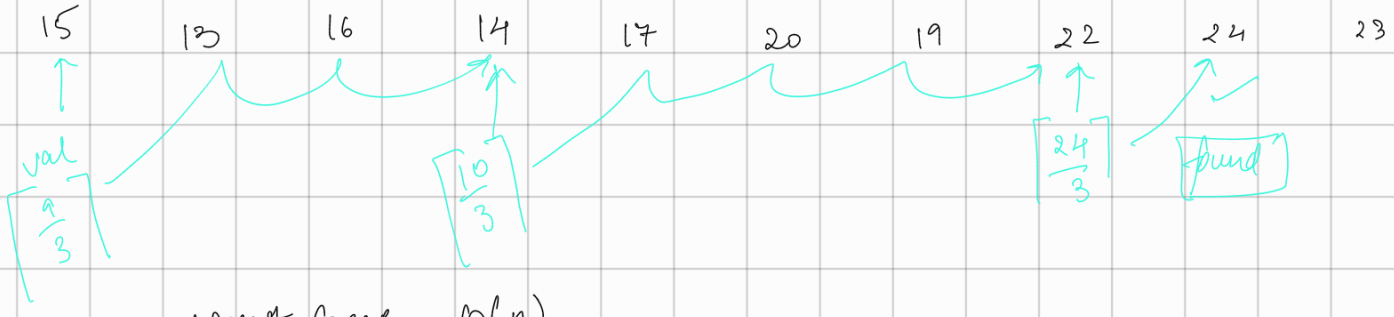
Jump over 6 steps.

Corner case:



$val = \frac{abs(target - a[i])}{k}$

target = 24



Worst Case: O(n)
Best Case: O(1)
Average Case: O(N/2) (maybe)

(4) Unbounded Binary search. / Infinite Binary Search

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 ... ∞

assume target = 21

Cannot use direct binary search because high is not defined

Use approach similar to
fenwick tree
binary lifting

fix low = 0 , high = low + 1
Search from low to high

if not found then update high = 2 × high
and low = high + 1,

if not found still, low = high+1 & high = high × 2
again search in [low, high]

.
.
.

upto when the element is found or low > element.

whenever a[low] <= target and a[high] >= target
apply normal BS.

$O(\log_2 N) < O(\log_3 N)$

So doubling gives the best complexity

Recursive BS: TC: $O(\log_2 N)$ & SC: $O(\log_2 N)$

∴ CODE:

```
binSearchInfinite(int[] arr, int low, int high, int target) {

    if (target >= arr[low] && target <= arr[high])
        return binarySearch(arr, low, high, target);
    else
        return binarySearchInfinite(arr, high+1, 2*high, target);
}
```

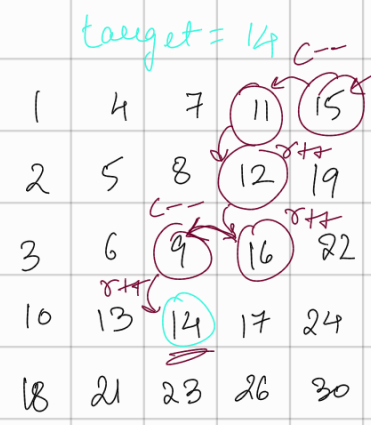
doubling the window size will take least time as compared to least size window

⑤ Search in a 2D Matrix II

Brute force : BS on each row. $O(n \log_2 m)$

Approach : Stair case search → $O(n+m)$

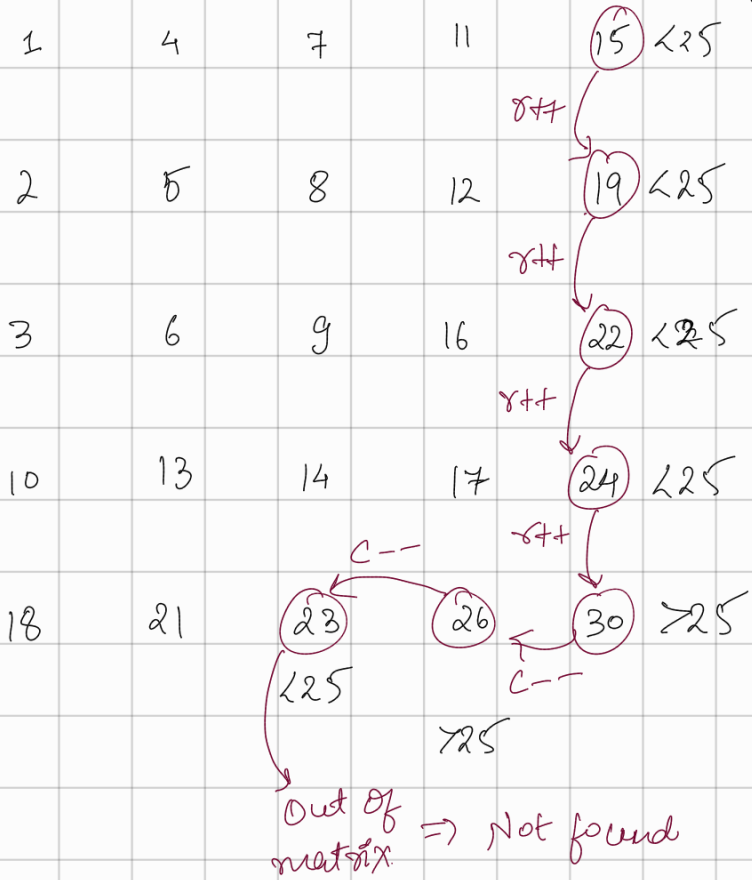
Divide & Conquer

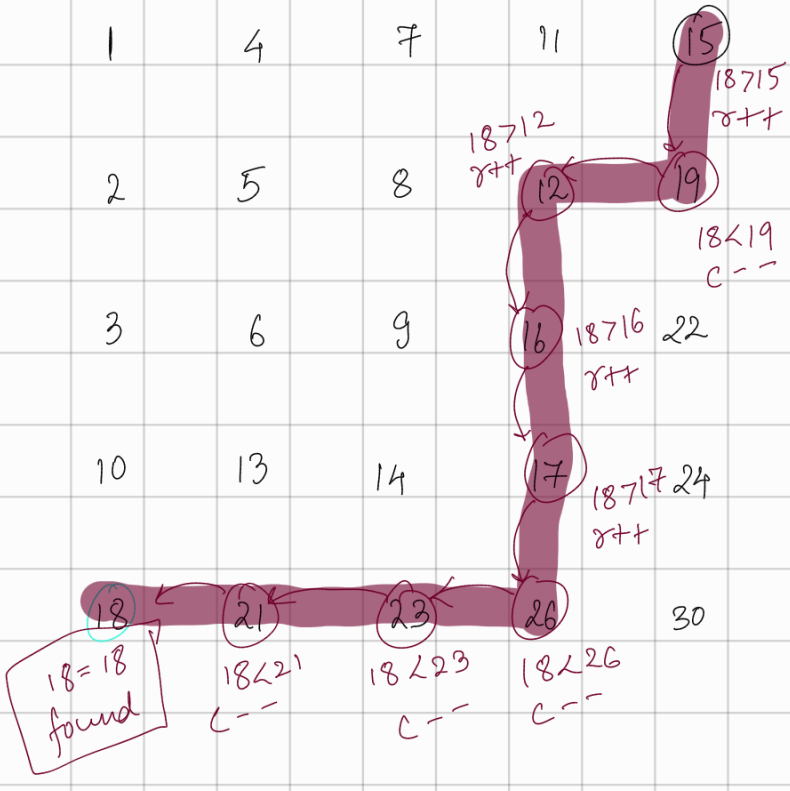


start = 15 , r = 0, c = 4

```
if (target > start){
    c--;
}else if (target < start)
    r++;
}else {
    found at [r][c];
}
update start = mat[r][c];
```

after updating





Start Top Right

target = 18

if ele < target
r++
else if ele > target
c--

Unsuccessful search : out of matrix

Worst case: O(m+n)

Looks like a staircase, so it is called a staircase approach

Every iteration, we reduce search space by one row or one column

∴ CODE:

```
int row=0, col = mat[0].length-1;
while (row < matr.length && col >=0){
    if (matrix[row][col] == target) return true;
    if (matrix[row][col] > target) col--;
    else row++;
}
return false;
```

(★) Minimize max distance to gas station

	3		6		12		19		33		44		67		72		89		95
		3		6		7		14		11		23		5		17		6	

Greedily, before & after as is not useful.

Place first in between max distance (Here 23) → 44 "New" 67

So Now 44 12 56 11 67

Next: b/w 72 & 89 at → $\left(\frac{72+89}{2}\right) = 80$,

So Now 72 8 80 9 89

low = 3 , high = 23

Binary search on ans over

if K gas stations can be placed optimally
then true else false