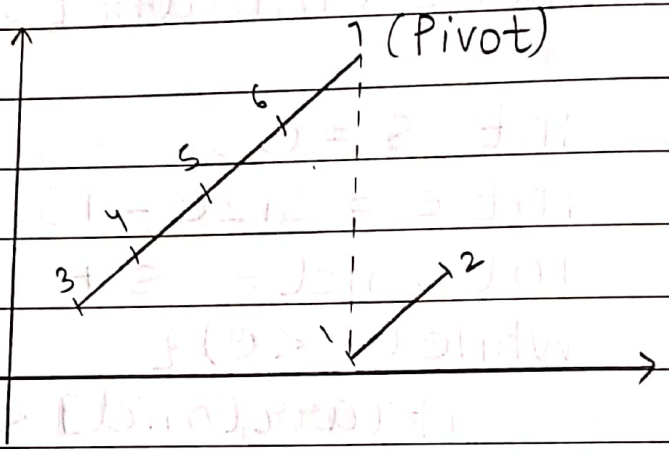
18/02/2023

Q1 Find pivot element in an array

i/p →	3	4	5	6	7	1	2



O/p → 7 or 1

Order breaks at the pivot element. At some platform 7 is pivot & at some 1 is the pivot element. Here we will consider 7 is the pivot element

The element at which the monotonic order breaks is the pivot element.

Brute force approach can be the linear search, but this approach has time complexity of O(n) but can we do in O(log n).

Here our code can get stuck & hence we can explicitly handle these 2 cases. or numbers

Algorithm

1) Start = 0
end = size - 1 = 7 - 1 = 6
mid = $\frac{0+6}{2}$ = 3
arr[mid] = 6

Suppose that our mid comes at 4th index, then arr[mid] = 7 & arr[mid+1] = 1

```
if (arr[mid] > arr[mid+1]){
        return mid;
}
if (arr[mid-1] > arr[mid]){
        return mid-1;
}
```

→ These are the 2 cases which we are explicitly handling.

Also we have to make sure mid+1 & mid-1 is a valid index.

2) Now only 2 conditions are left i.e. to search in right part or left part. We know that pivot is the maximum element. Compare with starting element.

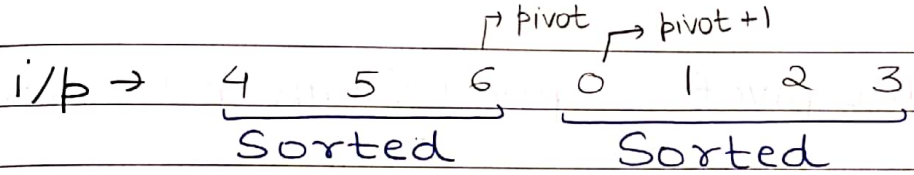
```
if (arr [s] >= arr [mid]) {
        e = mid - 1; // Left part
}
else if (arr [s] < arr [mid]) {
        s = mid + 1; // Right part
}
```

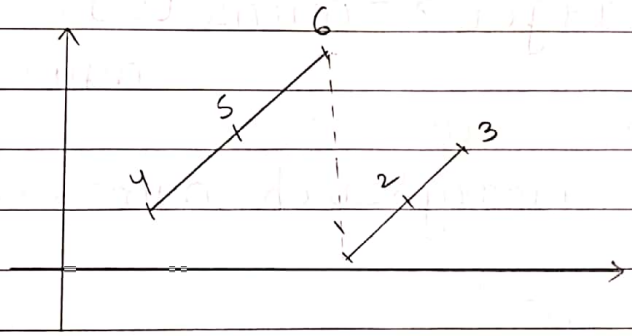
Code

```
int pivotElement (vector <int> arr) {

    int s = 0;
    int e = arr.size() - 1;
    int mid = s + (e-s)/2;

    while (s < e) {
                ↑ valid index
      if (mid+1 < arr.size() && arr [mid]
                                > arr [mid+1])
          return mid;
      }
                ↑ valid index
      if (mid-1 >= 0 && arr [mid-1] > arr [mid])
      {
        return mid-1;
      }
      if (arr [s] >= arr [mid]) {
            e = mid-1;
      }
```

```
else if (arr[s] < arr[mid]) {
        s = mid ;  // As in while we
                   have used s<e
}
mid = s + (e-s)/2;
}
return s ;  → For single element in array
}           & not gone in while loop.
```

Q2 Search in rotated sorted array.





Suppose that we need to find 2 in the array with the help of binary search.

Algorithm

1) Find pivot element which is 6 here. Now compare key = 2 which we need to compare with pivot.

2 < 6, hence we need to search for the element in sorted array - 2

4 5 6 | 0 1 2 3

array - 1 | array - 2

2) We can reuse the code of pivot element & then decide in which array we need to search the element & once we get to know the array, then simply apply the binary search.

Code

```
int search (vector<int>& nums, int t){

    int pivot = pivotElement (nums);

    if (target >= nums [0] && target <=
                        nums [pivot]){
  // Search in left array
  int ans = binarySearch (nums, target, 0,
                        pivot);

  return ans;
}
    // Valid index
if (pivot +1 < nums.size() && target
                        >= nums [pivot+1] &&
                    target <= nums [nums.size()
                        -1]){
    // Search in right array
    int ans = binarySearch (nums, target,
                        pivot+1, nums.size()-1);

    return ans;
}
return -1;
}
```

The question now comes to our mind that what is rotation in the array.

0 1 2 3 4 5 6 → sorted array

1st rotation

6 0 1 2 3 4 5

2nd rotation

5 6 0 1 2 3 4

3rd rotation

4 5 6 0 1 2 3

We can do further rotations on the array.

Note → Other way is that we can apply binary search on both arrays & the one who is returning index != -1 will be the answer.

Q3 Square root of a number using binary search

i/p → 12
o/p → 3 (only integer part)

Square root of a number n will lie within 0 to n. This is known as the search space

n = 10

Search Space: 0 1 2 3 4 5 6 7 8 9 10

s → 0

mid → 5

e → 10

1) start = 0
end = size - 1 = 10
mid = 5

arr [mid] → 5
5 * 5 = 25
25 > target = 10
So we need to search in left part.

2) start = 0
end = 4
mid = 2

arr [mid] = 2
2 x 2 = 4
4 < target = 10
Store the answer and then search in the right part of array.

3) start = 3
end = 4
mid = $\frac{3+4}{2}$ = 3
arr [mid] = 3

3 * 3 = 9
9 < 10
Store answer & search in right part.

4) start = 4 } Both becomes equal.
end = 4 }

Code

```
int squareRoot (int n) {
    int ans = -1
    int s = 0;
    int e = n;
    int mid = s + (e-s)/2;
    int target = n;
    while (s <= e) {

        if (mid * mid == target) {
            return mid;
        }
        if (mid * mid > target) {
            e = mid - 1;
        }
        else {
            ans = mid;
            s = mid + 1;
        }
        mid = s + (e - s)/2;
    }
    return ans;
}
```

Note → We are storing the ans = mid because it might happen that if we go to right & we won't get the answer, then we get the ans = -1 only which is wrong answer.

Now we need to find the decimal part.

ans = 3

3.1 → 3.1 * 3.1 <= 10 } Now check
3.2 → 3.1 * 3.1 > 10
3.3
3.4
⋮
3.9

ans = 3.1

3.11 3.11 * 3.11 <= 10
3.12 3.12 * 3.12 <= 10
3.13 ⋮
⋮ 3.16 * 3.16 <= 10 ans = 3.16
3.17 * 3.17 > 10
3.19

Additional code

```
double ans = squareRoot(10);
int precision = 3;
double step = 0.1;

for (int i=0; i<precision; i++){
    for (double j=ans; j*j<=n; j=j+step)
        ans = j;
    }
    step = step/10;
}
cout << ans << endl;
```

Q4 Binary Search in 2D matrix

i/p →

1	2	3	4	5
6	7	8	9	10
11	12	(13)	14	15
16	17	18	19	20
21	22	23	24	25

n → no. of rows
m → no. of columns

Algorithm

start = 0

end = n * m - 1 = 5 * 5 - 1 = 25 - 1 = 24

$$mid = \frac{start + end}{2} = \frac{0 + 24}{2} = 12$$

arr [ ] [ ]
↳ row Index ↳ col Index

rowIndex → mid / cols } Formulae
colIndex → mid % cols } (cols → m)

element = arr [rowIndex] [colIndex];

if (element == target) return true;
if (element > target) e = mid - 1; Left part
else s = mid + 1; Right part

Code

```
bool binarySearch (int arr [][5], int r,
                   int c, int t) {

int s = 0;
int e = r * c - 1;
```

```
int mid = s + (e-s)/2;

while(s<=e){

    int rowIndex = mid/cols;
    int colIndex = mid%cols;
    int element = arr[rowIndex][colIndex];
    if(element == t){
        return true;
    }

    if(element < t){
        s = mid+1; // left part
    }
    else{
        e = mid-1; // Right part
    }
}
return false;
}
```