BRAC UNIVERSITY

Inspiring Excellence

CSE221

Algorithms: *Ternary Search*

Ternary Search

- A divide and conquer algorithm
- Very similar to binary search except that the array is broken down into 3 portions
- There are 2 “mids” and we check for the item, ***k*** inside both of them
- If we find then bingo! Else we check if k < first mid or k > second mid or k > first mid and k < second mid? If the first condition is true we look into the rightmost portion of the array. If the second condition is true we look into the leftmost portion of the array else we look into the centre portion.
- Note that the red marked conditions are true if and only if the array is sorted.

Ternary Search

Pseudo code (Iterative)

```
boolean binarySearch(A[], l , r, item){
        while (l <=r){
                mid1 = l + (r - l)/3;
                mid2 = r - (r - l)/3
                if (A[mid1]==item || A[mid2]==item){
                        return true;
                }else{
                        if (item<A[mid1]){
                                r = mid1-1;
                        }elseif (item>A[mid2]){
                                l = mid2+1;
                        }else{
                                l =mid1+1; r = mid2-1;
                        }
                }
        }
        return false;
}
```

l must always be less or equal to r for the loop to run

Find the mid1 and mid2 index

Check if item matches, return true if does else next step.

Check if the item is smaller, if yes shift the search range to leftmost portion by moving r

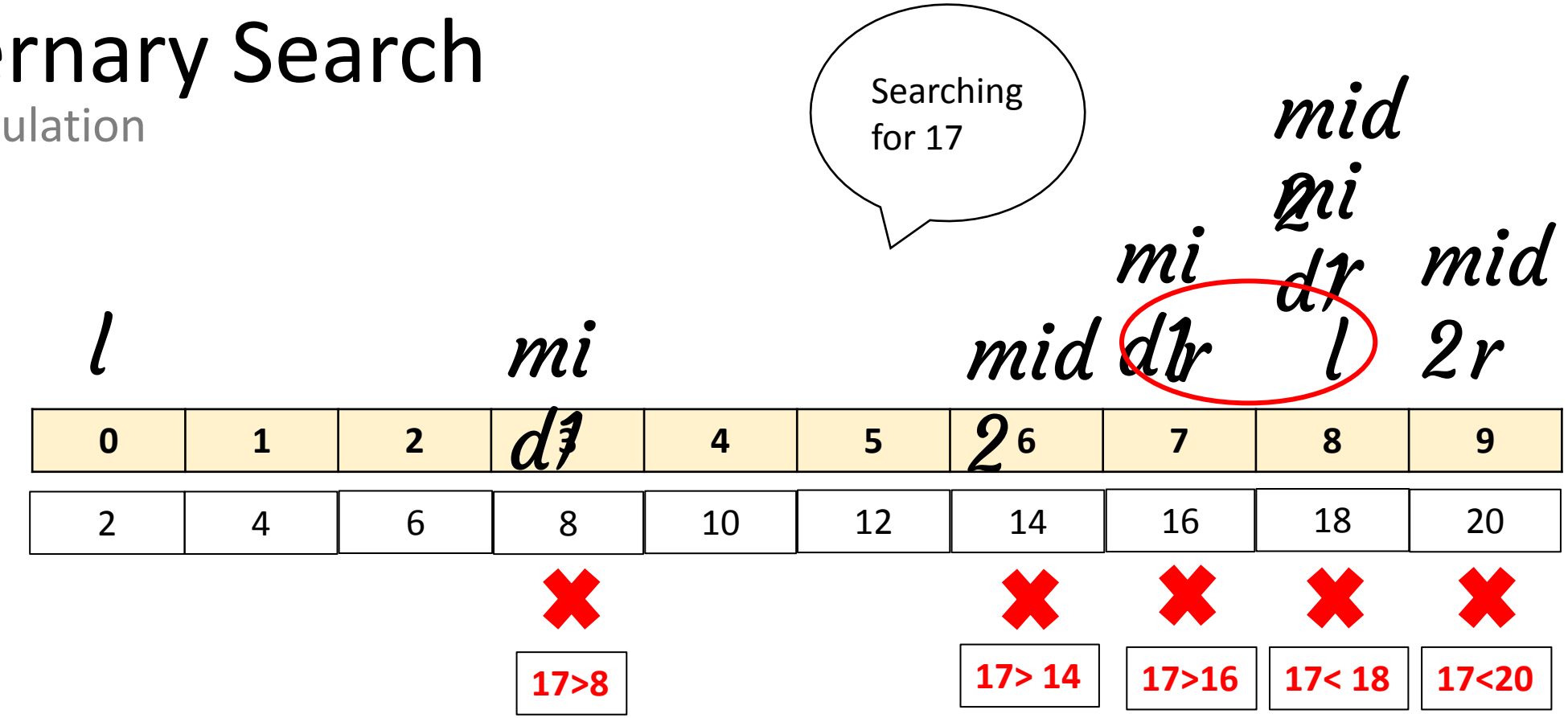
If the above condition is dissatisfied, shift the search range to rightmost portion by moving *l*

Else shift the search domain to the middle portion

The program will reach this line after while loop is complete. False is returned.

Ternary Search

Simulation



Ternary Search

Time Complexity (Iterative)

We have already learnt or at least got a good idea to find the time complexity of divide and conquer problems from the binary search slide. Refer to the algorithm and simulation of ternary search, you must get the concept that this and the binary search are almost the same saving the fact that at each step the problem size (range) is reducing by factor of 3.

Initially the domain of search was the entire array of length ***n***. At each step the search space was getting smaller by factor of 3 until 1 or 0. Find the running is same as binary search.

Problem Size	Step No.
n	0
n/3	1
n/9	2
n/27	3
….	….
1	k

Binary Search

Time Complexity (Iterative) Contd.

Problem Size	Step No.	Work done at each step
n	0	1
n/3	1	1
n/9	2	1
n/27	3	1
….	….	1
1	k	1

It took ***k*** steps for the searching to end and work done at each step was constant. We need to find ***k*** in terms of ***n.*** If you notice the divisors of the problem size (marked red) are all powers of 3 and we can use the step numbers as exponents. Therefore each problem size can be written as,

n/3^(step no.)

The last line can be written an, ***1 = n/3^k***.

If we solve it, we will find that $k = \log_3 n$

Therefore the time complexity is $\log_3 n$ x ***1***, which eventually is O($\log_3 n$)

Ternary Search

Recursive

BRAC UNIVERSITY
Inspiring Excellence

1. Find the recurrence equation of ternary search.
2. Solve the above equation to derive the big O of ternary search. [Draw the tree]
3. What do you think the running time would be if find the number in the very first loop?