Tuesday, 16 August 2022 9:05 PM

Searching

Q. Given an array of integers[A], check if it is present in it or not?

→ go to each and every element of A and check if it is equal to x.

Lookup: is present (v, A) {
for i = 0 → n-1 {
if (A[i] == v) {
return true
}
}
return false;
}
→ O(n)

Given: array has none indexing
reversed
come approach: → O(n)

Binary Search

Lec of min?
$\frac{n}{2^k} = \text{large}$
$\frac{n}{2^k} = 1$
$\frac{n}{2^k} = 2$ → log

1. Search space
2. Get mid
3. Take decision based on the middle value

Given sorted integer array [ascending]

$x = 16$

$x > 16$ → right
$x < 16$ → left

[10 | 14 | 16 | 18 | 20]

search space

$mid = \frac{0+4}{2} = 2$

$x = 16$

$mid = \frac{0+1}{2} = \frac{1}{2} = 0$

two right

$mid = \frac{1+1}{2} = 2$

if (arr[mid] == x)
return true

[2 3 4 5 5 6 7 8]

Pseudocode (Iterative Implementation)

L = 0
h = arr.length - 1
while (L <= h) {
int mid = $\frac{L+h}{2}$
if (arr[mid] == x) {
return mid;
}
if (arr[mid] < x) {
L = mid + 1;
} else {
h = mid - 1;
}
}
return -1;

Time Complexity

$T(n) = T(\frac{n}{2}) + c$ — (1)
$T(\frac{n}{2}) = T(\frac{n}{4}) + c$ — (2)

Substitute (2) in (1),
$T(n) = T(\frac{n}{4}) + 2c$ — (3)
$T(\frac{n}{4}) = T(\frac{n}{8}) + c$ — (4)
$T(n) = T(\frac{n}{8}) + 3c$ — (4)

$T(n) = T(\frac{n}{2^k}) + kc$

At some point, we get stuck
we reach only 1 element
$\frac{n}{2^k} = 1$
$n = 2^k$
Taking log on both sides
$\log_2 n = \log_2 2^k$
$\log_2 n = k$
$\log_2 2^k = k$
$\log_2 2^k = k$

$T(n) = T(\frac{n}{2^k}) + \log_2 n \cdot c$
$T(n) = T(\frac{n}{2}) + \log_2 n \cdot c$
$T(n) = T(1) + \log_2 n \cdot c$ → constant
$T(n) = \log_2 n$

TC → O($\log_2$)
SC → O(1)

→ Recursive Implementation

binarySearch (int[] A, int mid, int left, int r) {
if (mid > left) return -1;
int mid = (mid + left)/2;
if (x == A[mid]) return mid;
else if (x < A[mid])
return binarySearch (A, mid, mid-1, x);
else
return binarySearch (A, mid+1, left, x);
}

TC → O($\log_2$)
SC → O($\log_2$)

Conditions for applying binary search:

Array / list should be monotonic function.

Monotonicity

The monotonicity of a function tells us if it is increasing or decreasing

Monotonic function

→ functions is strictly on the entire domain, it is increasing or decreasing on the entire domain.

$y(x) = x^3$

Strictly increasing

Strictly decreasing

Non-monotonic function

A non-monotonic function is a function that is increasing or decreasing in different intervals of the domain

$y(x) = x^2$

Q. Given a sorted array, find the first occurrence of x.

2 3 5 8 8 10 12

x = 8

ans → 3

L = 0
h = n-1
ans = -1
while (L <= h) {
int mid = (L + (h-L))/2;
if (a[mid] > key) {
h = mid - 1;
}
else
L = mid + 1;
}
if (ans == -1) { print (not found) }
else return ans;

Q. → F F F T T T T

find index of first true

ans = mid;

FF TT

Q. Given a sorted array, find the first value $\geq x$.

$\frac{n+1}{2} = \frac{11}{2}$ → 5.5 → 6

2 3 5 8 8 10 12

ans → 10

L = 0
h = n-1
ans = -1
while (L <= h) {
int mid = $\frac{L+h}{2}$
if (a[mid] $\geq$ key) {
h = mid - 1;
ans = mid;
}
else
L = mid + 1;
}
return ans;

Upper bound and lower bound of binary search

→ lower bound: the smallest element greater than or equal to the element

Eg 1 2 3 4 5 5 5 6 7 9
Lb
≥ 5 → 5
UB

Eg 1 2 3 4 5 5 5 6 7 9
≥ 5 → 6
LB
UB

Eg 1 2 3 4 5 5 5 6 7 9
≥ 5 → 6
UB
LB

Frequency of a element → O($\log_2$)

if (upper bound - lower bound) > 0
→ True

Trie

primitive data types

① Array:- binarySearch(int, key)
→ if element present → True
→ element × → (-(insertion point) - 1)

point at which the key would be inserted in the array.

② Collections:- binarySearch( ) ;