Tuesday, 22 June, 2021 8:25 PM

Searching

Q. given an array of integers [A]

check if (x) is present in

[2 3 1 5 6]

Random is present (x, A) {
for i -> A length
if (A[i] == x)
return true;
}

}

array has some ordering [sorted]

some optimal (?)

Binary search

→ Elements [ordered] [sorted] x = 16 given

→ [1 3 5 16 18] → [1 3 5 16 18]

mid

(16 → higher) ××

low high

[1 3 5 16 18]

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

$\frac{1+4}{2} = \frac{5}{2} = 2.5$

$\frac{3+4}{2} = \frac{7}{2} = 3.5$

→ 16

mid

low high

pseudocode

L = 0
h = size - length - 1

while (L <= h) {
int mid = L + (h - L) / 2;
if (arr[mid] == x)
return mid;
if (arr[mid] > x) {
h = mid - 1;
}
else {
L = mid + 1;
}
}
return -1;

Time complexity

T(n) = T(n/2) + c

T(n) = T(n/2) + c — (1)

T(n/2) = T(n/4) + c — (2)

(2) in (1),

T(n) = T(n/4) + c + c — (3)

T(n/4) = T(n/8) + c — (4)

T(n) = T(n/8) + 3c — (5)

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

At some point, n/2^k diminishes, now

Reach only 1 element.

$T\left(\frac{n}{2^k}\right) = T(1)$

$\frac{n}{2^k} = 1$

$T(n) = T\left(\frac{n}{2^k}\right) + kc$ → (6)

$n = 2^k$

Take $\log_2$ on both sides

$\log_2 n = \log_2 2^k$

$\log_2 n = k$

$T(n) = T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \cdot c$

$T(n) = T\left(\frac{n}{n}\right) + \log_2 n \cdot c$

$T(n) = T(1) + \log_2 n \cdot c$ — constants

$T(n) = \log_2 n$

TC → O(log n)
SC → O(1)

Recursive implementation:

binary search (int [] A, int low, int high, int x) {
if (low > high) return -1;
int mid = (low + high)/2;
if (x == A[mid]) return mid;
else if (x < A[mid])
return binary search (A, low, mid-1, x);
else
return binary search (A, mid+1, high, x);
}

}

TC → O(log n)
SC → O(log n)

Condition for applying binary search:

Array [(1)] should be monotonic function

Monotonicity

The monotonicity of a function refers to if a function is increasing or decreasing

Monotonic functions

→ functions is increasing or its value remains or decreasing on the entire domain.

y(x) = x²

Not monotonic functions

A non-monotonic function is a function that is increasing or decreasing on different intervals of domain.

f(x) = x³

→ find first value 1 in 1?

Q. given a sorted array, find the first value of x

[1 1 1 2 2 2 3 3 3]

x = 2

mid = 4

high

mid = 2

low

L = 0
h = n-1
mid = -1
while (L <= h) {
int mid = L + (L-h)/2;
if (a[mid] == x)
mid = mid;
h = mid - 1;
else
L = mid + 1;
}
return mid;

Q. find first value x in

[1 3 5 6 8]

x = 3

mid = 2

low

high

L = 0
h = n-1
mid = -1
while (L <= h) {
if (a[mid] == x)
mid = mid;
else
L = mid + 1;
}
return mid;

Upper bound and lower bound of a Binary search

→ bound is defined positions where the value x would be inserted without breaking the ordering.

lower bound

Eg: 1 2 3 5 7 10 Bound 6 9 7

6 → 5 → 7

upper bound

lower bound

Eg: 1 2 3 5 7 10 Bound 6 9 7

6 → 5

lower bound

Eg: 1 2 3 5 7 10 Bound 6 9 7

6 → 7

upper bound

Frequency of a element

① lower bound
② upper bound
③ (upper bound - lower bound) → freq

Step

Binary search (arr, key)

→ if element present → index

→ element x → (-(insertion point) - 1)

point at which the key would be inserted into the array

Collections. Binary Search()

Cost

1. binary search (int pos, int pos, int, num)

→ indexes

2. lower bound (int pos, int pos, int, num)

→ no occurrences → position is not unique ie.

→ occurrences → return position

→ multiple occurrences → return first position

3. upper bound (int pos, int pos, int, num)

Sequential / linear

1 → 1
2 → 2
3 → 3
4 → 4
5 → 5

upper bound
lower bound
upper bound

→ position → 5 → comparisons

Total (n)
Average → (n+1)/2
→ n+1/2

→ O(n)

1 1 1 1 1 1 1 1 1

linear search

Q. find 2 in

1 2 3 4 5 6 7 8 9

→ 2