Case Analysis

Mohammed Rizin
Unemployed

March 21, 2025

1 Linear Search Algorithm

Algorithm 1 Linear Search Algorithm

```
1: procedure LinearSearch(A, n, x)
2:     for i ← 0 to n − 1 do
3:         if A[i] = x then
4:             Return i
5:         end if
6:     end for
7:     Return −1
8: end procedure
```

1.1 Best, Worst, Average Time

Suppose we have an array:

8	6	12	5	9	7	4	3	16	18
0	1	2	3	4	5	6	7	8	9

$\underbrace{}_{key}$

Figure 1: Searching for key = 3 after 7 steps

8	6	12	5	9	7	4	3	16	18
0	1	2	3	4	5	6	7	8	9

Not Found!

Figure 2: Searching for key = 0 after 10 steps

Best Case $B(n) \implies$ Searching key element is present at index 0. Then it will take $O(1)$.
Worst Case $W(n) \implies$ Either the element is absent or its at last index. Then it will take $O(n)$.
Average Case $\implies \frac{\text{all possible cases time}}{\text{no. of cases}}$

Most of the cases, finding the average case time is not possible. But most of the cases it would be equal to Worst case time.

To calculate :

$$AvgTime = \frac{\sum_{i=0}^{n} \text{time taken to find } i^{th} \text{ index element}}{n}$$
$$AvgTime = \frac{1+2+3+\cdots+n}{n}$$
$$AvgTime = \frac{\frac{n(n+1)}{2}}{n}$$
$$AvgTime = \frac{(n+1)}{2} \simeq O(n) \cong W(n)$$

All these complexities are based on comparison operations you are doing. In average case, we considered every single possibility and then took the mean of it.

1.2 Asymptotic Notations for Linear Search

As mentioned earlier, Cases are no where related to notations. Notations apply to functions. But cases are related to Analysis of that function.

$$B(n) = 1$$
$$B(n) = O(n)$$
$$B(n) = \Omega(n)$$
$$B(n) = \Theta(n)$$

$$W(n) = n$$
$$W(n) = O(n)$$
$$W(n) = \Omega(n)$$
$$W(n) = \Theta(n)$$

Don't get confused with $O(n)$ is for worst case scenario and $\Omega(n)$ is for best case scenario.

2 Binary Search Tree

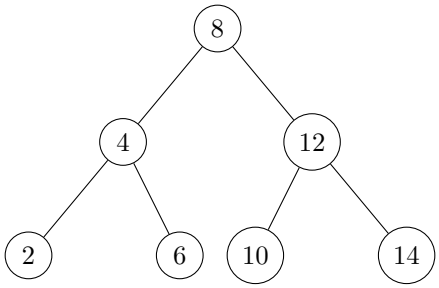

Figure 3: Balanced Binary Search Tree: Minimum Heights

Binary Search Trees are used for searching. All the elements in the right is greater than element in the left. If we are searching for smaller element, then we goto left hand side.

Suppose we are looking for element 2: We check if $2 \geq 8?No$, then go right.
We check if $2 \geq 4?No$, then go right.
if $2 \geq 2?Yes$, We found the element

Let us check how many comparison we have done. Generally the depth of a binary is $\log n$. So the number of comparisons are $\log n$ i.e. $\boxed{O(\log n)}$.

2.1 Analyze Best Case, Worst Case, Avg Case

Best Case Time $B(n) \implies$ Searching for the element at the root. This will take $O(1)$. Worst Case Time $W(n) \implies$ Searching for the element from below of the tree, i.e leaf. This will take $O(logn)$, i.e. depends on the height of the tree.

We will not find $AvgTime$ in this case. This is not the only binary search tree possible for the given elements.

2.2 Skewed Binary Search Tree

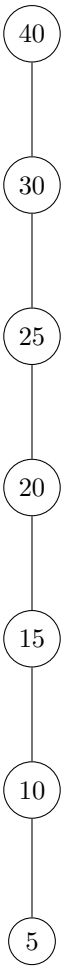

Figure 4: Left Skewed Binary Search Tree: Maximum height

2.2.1 Analyze Best Case, Worst Case, Avg Case

Best Case Time $B(n) \implies$ Searching for the element at the root. This will take $O(1)$.
Worst Case Time $W(n) \implies$ Searching for the element from below of the tree, i.e leaf. This will take $O(n)$, i.e. depends on the height of the tree.

2.3 Conclusion: Binary Search Tree

$\boxed{\min W(n) = \log n}$ $\boxed{\max W(n) = n}$

Worst case means maximum and its common for people to confuse how minimum of $W(n)$. It depends on the type of tree or problem we are solving.