VE281
Data Structures and Algorithms

Average-Case Time Complexity of BST

Learning Objectives:

- Know the average-case time complexity of search, insertion, and removal operations for a binary search tree

Which Statements Are Correct?

- Suppose the **depth (height)** of a binary search tree is h. Consider the time complexity for a **successful** search.
  A. In the worst case, the complexity is $O(h)$
  B. In the average case, the complexity is $O(h)$
- Suppose the **number of nodes** of a binary search tree is n. Consider the time complexity for a **successful** search.
  C. In the worst case, the complexity is $O(n)$
  D. In the worst case, the complexity is $O(\log n)$

How about average-case time complexity for a **successful** search in terms of the number of nodes n?

Average Case Analysis

- If the successful search reaches a node at depth d, the number of nodes visited is $d + 1$.
  - The complexity is $\Theta(d)$.

- Assume that it is equally likely for the object of the search to appear in any node of the search tree. The average complexity is $\Theta(\bar{d})$
  - $\bar{d}$ is the average depth of the nodes in a given tree

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$$

Internal Path Length

- $\sum_{i=1}^{n} d_i$ is called **internal path length**.
- To get the average case complexity, we need to get the **average** of $\sum_{i=1}^{n} d_i$ for all trees of n nodes.
- Define the **average internal path length** of a tree containing n nodes as $I(n)$.
  - $I(1) = 0$.
- For a tree of n nodes, suppose it has l nodes in its left subtree.
  - The number of nodes in its right subtree is $n - 1 - l$.
  - The average internal path length for such a tree is
$$T(n;l) = I(l) + I(n-1-l) + n - 1$$
- $I(n)$ is average of $T(n;l)$ over $l = 0,1,\ldots,n-1$.

Internal Path Length

- Assume all insertion sequences of n keys $k_1 < \cdots < k_n$ are equally likely.
  - The first key inserted being any k_l are equally likely.
- Note: If first key inserted is k_{l+1}, the left subtree has l nodes.
- Claim: All left subtree sizes are equally likely.
- Therefore, we have

$$I(n) = \frac{1}{n}\sum_{l=0}^{n-1} T(n;l)$$

$$= \frac{1}{n}\sum_{l=0}^{n-1} [I(l) + I(n-1-l) + n - 1]$$

$$= \frac{2}{n}\sum_{l=0}^{n-1} I(l) + (n-1)$$

Solving the Recursion

$$I(n) = \frac{2}{n}\sum_{l=0}^{n-1} I(l) + (n-1)$$

replace n with $n-1$

$$I(n-1) = \frac{2}{n-1}\sum_{l=0}^{n-2} I(l) + (n-2)$$

$$\sum_{l=0}^{n-2} I(l) = \frac{(n-1)[I(n-1)-(n-2)]}{2}$$

Solving the Recursion

$$I(n) = \frac{2}{n}\sum_{l=0}^{n-1} I(l) + (n-1) \qquad \sum_{l=0}^{n-2} I(l) = \frac{(n-1)[I(n-1)-(n-2)]}{2}$$

$$I(n) = \frac{n+1}{n} I(n-1) + \frac{2(n-1)}{n}$$

$$\frac{I(n)}{n+1} = \frac{I(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \quad \leq \frac{I(n-1)}{n} + \frac{2}{n}$$

Solving the Recursion

$$\frac{I(n)}{n+1} \le \frac{I(n-1)}{n} + \frac{2}{n}$$

$$\frac{I(n)}{n+1} \le \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \cdots + \frac{2}{2} + \frac{I(1)}{2}$$

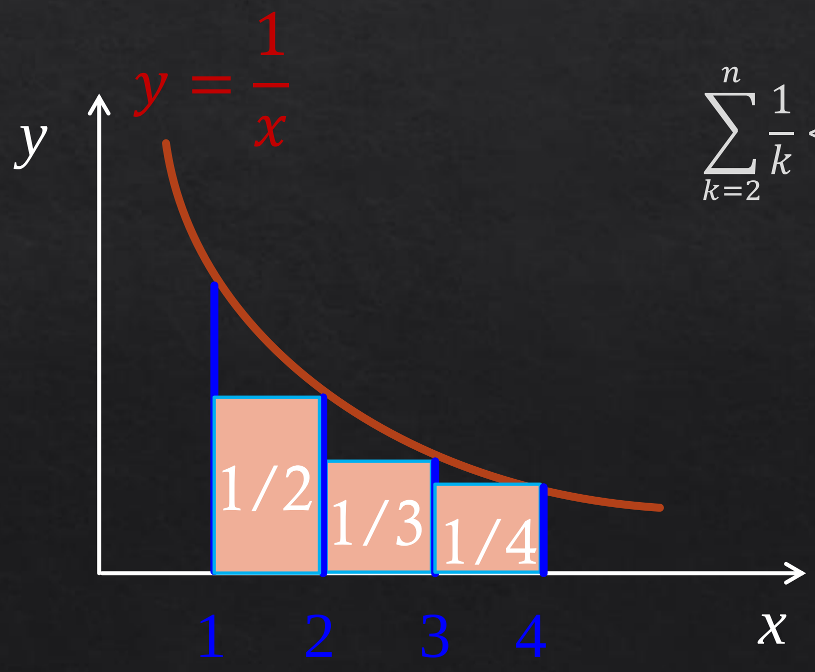
$$I(1) = 0$$

$$\frac{I(n)}{n+1} \le 2\sum_{k=2}^{n} \frac{1}{k}$$

Note: $\sum_{k=2}^{n} \frac{1}{k} < \ln n$

Proof of the Claim

- Claim: $\sum_{k=2}^{n} \frac{1}{k} < \ln n$



$$\sum_{k=2}^{n} \frac{1}{k} < \int_{1}^{n} \frac{1}{x} dx \;\; = \ln n$$

Average Case Analysis Conclusion

- What we get so far:

$$\frac{I(n)}{n+1} \le 2\sum_{k=2}^{n}\frac{1}{k} < 2\, ln\, n$$

- Thus, we have

$$I(n) = O(n \log n)$$

- Thus, the average complexity for a successful search is

$$\Theta\left(\frac{1}{n}I(n)\right) = O(\log n)$$

Average Case Time Complexity

- It can also be shown that given n nodes, the average-case time complexity for an **unsuccessful search** is $O(\log n)$.
- Given n nodes, the average-case time complexities for search, insertion, and removal are all $O(\log n)$.
  - Insertion and removal include "search".

	Search	**Insert**	**Remove**
Linked List	$O(n)$	$O(n)$	$O(n)$
Sorted Array	$O(\log n)$	$O(n)$	$O(n)$
Hash Table	$O(1)$	$O(1)$	$O(1)$
BST	$O(\log n)$	$O(\log n)$	$O(\log n)$

So, why we use BST, not hash table?