3 Trees: traversal and analysis of standard search trees
Binary search trees

- Binary trees for storing sets of keys, such that the operations are supported:
  - find
  - insert
  - delete

- Search tree property:
  all keys in the left subtree of a node $p$ are smaller than the key of $p$, and the key of $p$ is smaller than all keys in the right subtree of $p$.

- Implementation:
Standard binary search trees (8)

- Tree structure depends on the order of insertions into the initially empty tree
- Height can increase linearly, but it can also be in $O(\log n)$, more precisely $\lceil \log_2 (n+1) \rceil$. 

![Tree structure example](image)

Insert 5
Traversals of the nodes of a tree

- for output
- for calculating the sum, average, number of keys ...
- for changing the structure

Most important traversal orders:

- **preorder** = NLR (Node-Left-Right)
  first visit the root, then recursively the left and right subtree (if existent)
- **postorder** = LRN
- **inorder** = LNR
- the mirror image versions of 1-3
Preorder traversal is recursively defined as follows:

- **Traversal** of all nodes of a binary tree with root \( p \) in preorder:
  - visit \( p \),
  - traverse the left subtree of \( p \) in preorder,
  - traverse the right subtree of \( p \) in preorder.
Preorder implementation

// Preorder Node-Left-Right
void preOrder (){
    preOrder(root);
    System.out.println ();
}
void preOrder(SearchNode n){
    if (n == null) return;
    System.out.print (n.content+" ");
    preOrder(n.left);
    preOrder(n.right);
}

// Postorder Left-Right-Node
void postOrder(){
    postOrder(root);
    System.out.println ();
}
// ...
The traversal order is: first the left subtree, then the root, then the right subtree:

```java
// Inorder Left-Node-Right
void inOrder(){
    inOrder(root);
    System.out.println();
}
void inOrder(SearchNode n){
    if (n == null) return;
    inOrder(n.left);
    System.out.print (n.content+" ");
    inOrder(n.right);
}
```
Example

Preorder:  
17, 11, 7, 14, 12, 22
Postorder:  
7, 12, 14, 11, 22, 17
Inorder:  
7, 11, 12, 14, 17, 22
Sorting with standard search trees

Idea: Create a search tree for the input sequence and output the keys by an inorder traversal.

Remark: Depending on the input sequence, the search tree may degenerate.

Complexity: Depends on internal path length

Worst case: Sorted input: \( \Omega(n^2) \) steps.

Best case: We get a complete search tree of minimal height of about \( \log n \). Then \( n \) insertions and outputs are possible in time \( O(n \log n) \).

Average case: ?
Analysis of search trees

Two possible approaches to determine the internal path length:

1. **Random tree analysis**, i.e. average over all possible permutations of keys to be inserted (into the initially empty tree).

2. **Shape analysis**, i.e. average over all structurally different trees with \( n \) keys.

Difference of the expected values for the internal path:

1. \( \approx 1.386 \cdot n \log_2 n - 0.846 \cdot n + O(\log n) \)

2. \( \approx n \cdot \sqrt{\pi n} + O(n) \)
Reason for the difference

Random tree analysis counts more balanced trees more often.
Internal path length

Internal path length $I$: measure for judging the quality of a search tree $t$.

Recursive definition:

1. If $t$ is empty, then $I(t) = 0$

2. For a tree $t$ with left subtree $t_l$ and right subtree $t_r$:

\[
I(t) = I(t_l) + I(t_r) + \#nodes \ in \ t
\]

Apparently:

\[
I(t) = \sum_p (\text{depth}(p) + 1)
\]

$p$ internal node of $t$
Average search path length

For a tree $t$ the average search path length is defined by:

$$D(t) := I(t)/n, n = \# \text{ nodes in } t.$$ 

Question: What is the size of $D(t)$ in the

- best
- worst
- average

case for a tree $t$ with $n$ internal nodes?
Internal path: best case

We obtain a complete binary tree
Internal path: worst case
Random trees

- Without loss of generality, let \( \{1, \ldots, n\} \) be the keys to be inserted.
- Let \( s_1, \ldots, s_n \) be a random permutation of these keys.
- Hence, the probability that \( s_1 \) has the value \( k \), \( P(s_1=k) = 1/n \).
- If \( k \) is the first key, \( k \) will be stored in the root.
- Then the left subtree contains \( k-1 \) elements (the keys 1, \( \ldots, k-1 \)) and the right subtree contains \( n-k \) elements (the keys \( k+1, \ldots, n \)).
Expected internal path length

- \( EI(n) \) : Expectation for the internal path length of a randomly generated binary search tree with \( n \) nodes.

- Apparently we have:

\[
\begin{align*}
EI(0) & = 0 \\
EI(1) & = 1 \\
EI(n) & = \frac{1}{n} \sum_{j=1}^{n} (EI(j-1) + EI(n-k) + n) \\
& = n + \frac{1}{n} \left( \sum_{k=1}^{n} EI(k-1) + \sum_{k=1}^{n} EI(n-k) \right)
\end{align*}
\]

- Claim: \( EI(n) \approx 1.386n \log_2 n - 0.846n + O(\log n) \).
Observation

- Search, insertion and deletion of a key in a randomly generated binary search tree with \( n \) keys can be done, on average, in \( O(\log_2 n) \) steps.

- In the worst case, average cost for search, insertion and deletion can be linear in the number of items.

- One can show that the average distance of a node from the root in a randomly generated tree is only about 40% above the optimal value.

- However, by the restriction to the symmetrical successor, the behaviour becomes worse.

- If \( n^2 \) update operations are carried out in a randomly generated search tree with \( n \) keys, the expected average search path is only \( \Theta(\sqrt{n}) \).
Typical binary tree for a random sequence of keys
Resulting binary tree after $n^2$ updates