Splay trees
Deliverables

Sequential Access Property
Dynamic Finger Property
Working Set Property
Splay Tree definition
Operations on splay trees
Sequential Access Property

Sequential access is the property of a group of elements, that they must be accessed one after the other in a specific sequence.

Sequential Access Property: We can do inorder traversal of a BST in $O(1)$ amortized time. Let the numbers of the set $\{1, 2, \ldots, n\}$ be keys stored in a binary search tree. Let $\{x_1, x_2, \ldots, x_m\}$ be keys of an access pattern $x_i$.

It is known that some access patterns are very easy. For example if $x_i = k$ for all $i$, then placing $k$ at the root will be $O(1)$ access time. Similarly inorder access pattern will also give us constant amortized time.
let k be a key which appears in access sequence with probability \( p_k \). Then per operation cost of access is \( O(\sum p_i( \lg(1/p_i)) \)

This is because an element which appears with probability \( p_i \) can be stored at height \( \lg p_i \). This is strict lower bound for static sets.
Dynamic Finger Property

Dynamic finger property: if last access was to key $x_{i-1}$ and the present access is to key $x_i$, then the cost of the present access is $O(\lg |x_i - (x_{i-1})|)$

This property performs well on access patterns which exhibit spatial locality.
Working Set Property

Working Set property: Let $t_j$ the number of distinct keys between the present operation and the last time the same key was accessed. Let $t_j$ be the number of distinct keys in $\{x_i..x_j\}$ then the cost of accessing $x_j$ is $O(\log t_j + 2)$

Data structures with this property perform well on access patterns which exhibit temporal locality.
Unified Property: Let $t_{ij}$ be the number of distinct keys in 
$\{x_i, ..., x_j\}$ then the cost of accessing $x_j$ is $O(lg \min_i (|x_i - x_j| + t_{ij} + 2))$

It says that access to keys close to keys which have been recently accessed is cheap.
Splay trees

Splay Trees are Binary Search Trees

Searching in a Splay Tree: Starts the Same as in a BST

Splay trees are much simpler than AVL or Red Black trees. Their average performance is comparable and they provide additional properties not found in AVL or RB trees. It has working set and dynamic finger property.

Splay tree is a binary search tree where a node is splayed after it is accessed (for a search or update)

Deepest internal node accessed is splayed
Splay trees-complexity

Splaying costs $O(h)$, where $h$ is height of the tree – which is still $O(n)$ worst-case.

$O(h)$ rotations, each of which is $O(1)$.

Thus, amortized cost of any splay operation is $O(\log n)$.

In fact, the analysis goes through for any reasonable definition of rank($x$).

Thus splay trees can adapt to perform searches on frequently-requested items much faster than $O(\log n)$. 

Splay algorithm

All normal operations on a binary search tree are combined with one basic operation, called splaying. Splaying the tree for a certain element rearranges the tree so that the element is placed at the root of the tree.

One way to do this is to first perform a standard binary tree search for the element in question, and then use tree rotations in a specific fashion to bring the element to the top.

Alternatively, a top-down algorithm can combine the search and the tree reorganization into a single phase.
### Which nodes are splayed

<table>
<thead>
<tr>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>findElement</code></td>
<td>if key found, use that node. If key not found, use parent of ending external node</td>
</tr>
<tr>
<td><code>insertElement</code></td>
<td>Use the new node containing the item inserted</td>
</tr>
<tr>
<td><code>removeElement</code></td>
<td>Use the parent of the internal node that was actually removed from the tree (the parent of the node that the removed item was swapped with)</td>
</tr>
</tbody>
</table>
Splaying

When a node $x$ is accessed, a splay operation is performed on $x$ to move it to the root. To perform a splay operation we carry out a sequence of splay steps, each of which moves $x$ closer to the root. By performing a splay operation on the node of interest after every access, the recently accessed nodes are kept near the root and the tree remains roughly balanced, so that we achieve the desired amortized time bounds.

Each particular step depends on three factors:

Whether $x$ is the left or right child of its parent node, $y$,

whether $y$ is the root or not, and if not

whether $y$ is the left or right child of its parent, $g$ (the grandparent of $x$).
Splaying:

- "x is a left-left grandchild" means x is a left child of its parent, which is itself a left child of its parent.
- p is x's parent; g is p's parent.

1. Start with node x.
2. Is x the root?
   - Yes: Stop.
   - No: Is x a child of the root?
     - Yes: Zig
     - No: Is x the left child of the root?
       - Yes: Right-rotate about the root
       - No: Zig

3. Zig-Zig
   - Right-rotate about g, right-rotate about p

4. Zig-Zig
   - Left-rotate about g, left-rotate about p

5. Zig-Zag
   - Left-rotate about p, right-rotate about g

6. Zig-Zag
   - Right-rotate about p, left-rotate about g
Zig
Zig-Zag
Zig-Zig
X=(8,N) Before rotating

x is the right child of its parent, which is the left child of the grandparent
left-rotate around p, then right-rotate around g
After First Rotation

2.
(after first rotation)
After Second Rotation

3.
(after second rotation)

x is not yet the root, so we splay again
Before Rotation

Now $x$ is the left child of the root right-rotate around root.

1. (before applying rotation)
Last step

2.
(after rotation)

\[ x \rightarrow (8, N) \]

\[ (7, T) \]

\[ (1, Q) \]

\[ (7, P) \]

\[ (5, H) \]

\[ (2, R) \]

\[ (1, C) \]

\[ (5, J) \]

\[ (6, Y) \]

\[ (10, A) \]

\[ (14, J) \]

\[ (10, U) \]

\[ (35, R) \]

\[ (37, P) \]

\[ (21, O) \]

\[ (36, L) \]

\[ (40, X) \]

\[ x \text{ is the root, so stop} \]
Example Insert 0, 1, .. 8
After find 0
After find 1
After Insert 9
Deleting Node 5 Searching 5

![Diagram of a tree with nodes and edges]
Deleting Node 5 Removing Root
Deleting Node 5 Searching Largest
Deleting Node 5 Linking new root
Questions, Comments and Suggestions
Question 1

Splaying the tree for a certain element rearranges the tree so that the element is placed at the ____ of the tree.

A) Leaves
B) Root
C) Top Half
D) Bottom Half
Question 2

Amortized or Randomized access cost of splay tree is
A) O(n)
B) O(nlogn)
C) O(logn)
D) O(n²)
Question 3

A splay tree can be worse than a static tree in terms of any operation by a
A) logn factor
B) n factor
C) Constant Factor
D) $\lg^2 n$