מבני נתונים

עץ חיפוש בינארי

Tzachi (Isaac) Rosen

Dynamic Sets

- Elements have a **key** and **satellite data**
- **Dynamic sets** support queries such as:
  - **Search(S, k)**
  - **Minimum(S)**
  - **Maximum(S)**
  - **Successor(S, x)**
  - **Predecessor(S, x)**
  - **Insert(S, x)**
  - **Delete(S, x)**

Binary Search Trees

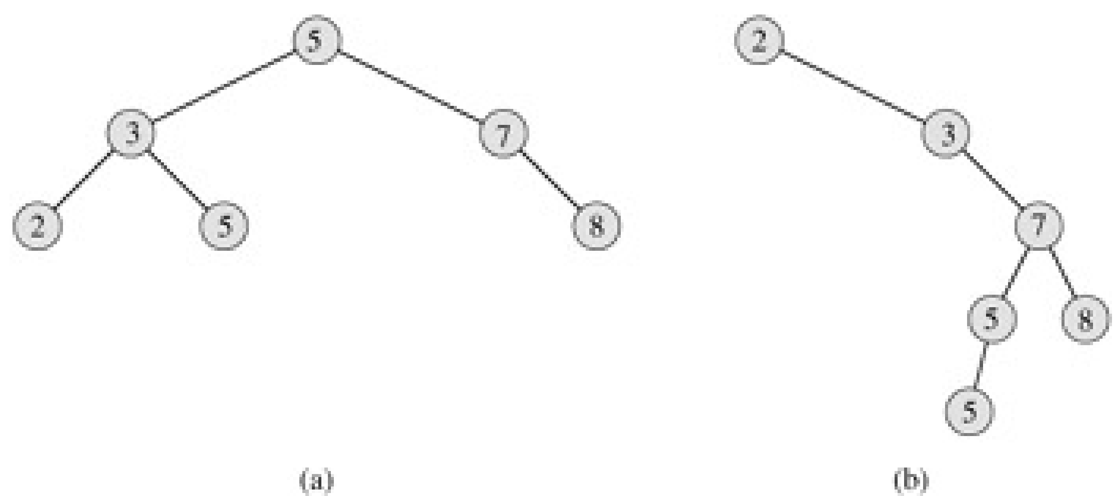
- **root**[T] points to the root of tree T
- In addition to satellite data, elements have:
  - **key:** an identifying field inducing a total ordering
  - **left**: pointer to a left child (may be null)
  - **right**: pointer to a right child (may be null)
  - **p**: pointer to a parent node (null for root)

Binary Search Trees

- ***Binary Search tree property***:

  key[y] $\leq$ key[x] < key[z], for any nodes x, y and z, such that y in left sub tree of x and z in right sub tree of x

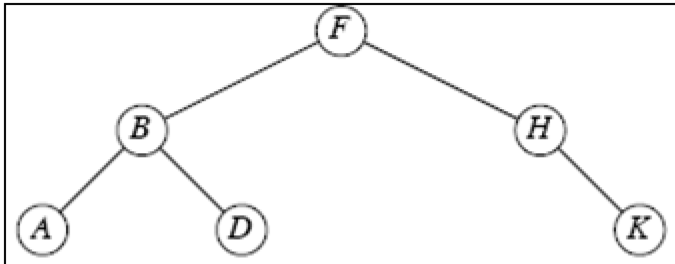
- Example:



Tree Walk

- **Inorder tree walk**:
  - walk left
  - visit root
  - walk right
- **Preorder tree walk**:
  - visit root
  - walk left
  - walk right
- **Postorder tree walk**:
  - walk left
  - walk right
  - visit root

```
inorder (node x)
  if (x ≠ null) then
    inorder (left[x])
    visit (x)
    inorder (right[x])
```

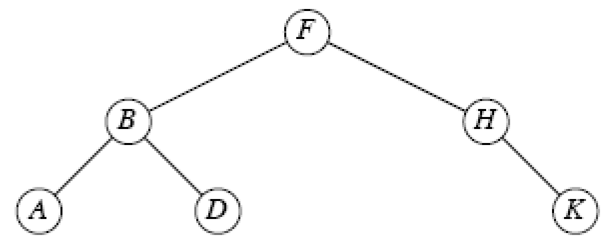


- Inorder: A B D F H K
- Preorder: F B A D H K
- Postoreder: A D B K H F

- **Complexity**: O(n), where n is the number of nodes of the tree

Tree Search

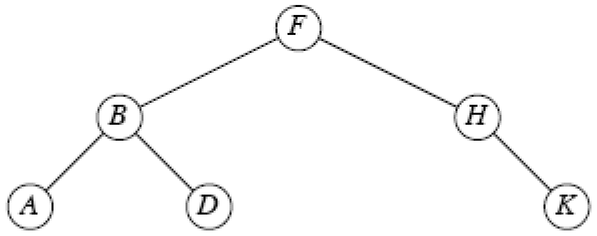
```
search (node x, key k)
    if (x = null | k = key[x]) do
        return x;
    if (k < key[x]) then
        return search(left[x], k)
    else
        return search(right[x], k)
```



- **Complexity**: O(h), where h is the height of the tree

Iterative Tree Search

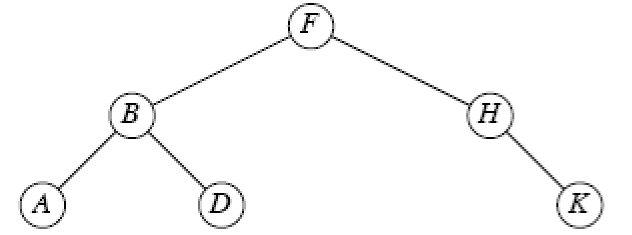
```
search(node x, key k)
    while (x != null & k != key[x]) do
        if (k < key[x]) then
            x = left[x]
        else
            x = right[x]
    return x
```



Minimum, Maximum, Successor and Predecessor

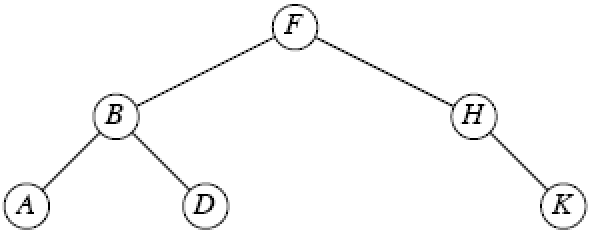
- **Minimum**:
  return leftmost node in tree or **null**
- **Successor** (node x):
  if (x has a right subtree) **then**
  successor is minimum node in right subtree
  else
  successor is first ancestor of x whose left child is also ancestor of x
- **Maximum**:
  return rightmost node in tree or **null**
- **Predecessor**:
  – symmetric to successor
- **Complexity**: O(h), where h is the height of the tree

```
successor (node x)
   if (right[x] ≠ null) then
      return minimum (right[x])
   y  = p[x]
   while (y ≠ null & x = right[y]) do
      x = y
      y = p[y]
   return y
```



Tree Walk

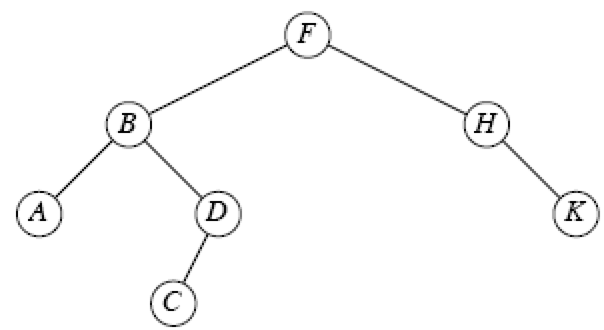
```
walk (node x)
    x = minimum(x)
    while (x != null) do
        visit(x)
        x = successor(x)
```



- **Complexity**: O(n), where n is the number of nodes of the tree

Insertion

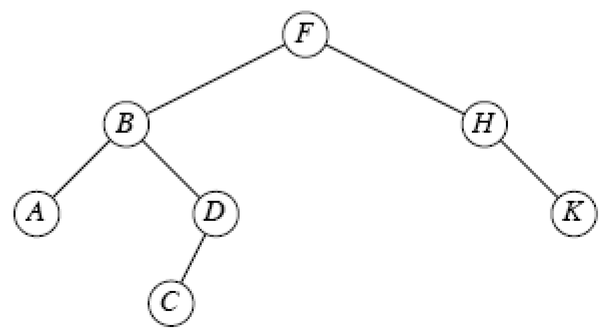
- Adds an element x to the tree so that the binary search tree property continues to hold
- The basic algorithm
  - Like the search procedure above
  - Insert x in place of null
  - Use a “trailing pointer” to keep track of where you came from (like inserting into singly linked list)



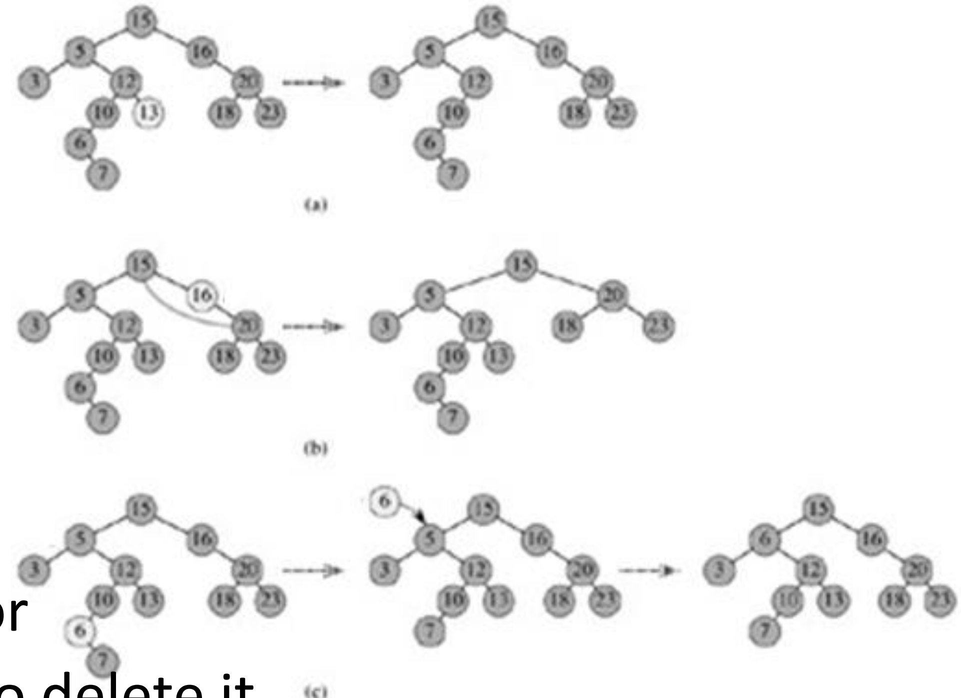
- **Complexity**: O(h), where h is the height of the tree

Insertion

```
insert (tree T, node z)
   y = null
   x = root[T ]
   while (x ≠ null) do
      y = x
      if (key[z] < key[x]) then
         x = left[x]
      else
         x = right[x]
   p[z] = y
   if (y = null) then // T was empty
      root[T ] = z
   else
      if (key[z] < key[y]) then
         left[y] = z
      else
         right[y] = z
```



Deletion



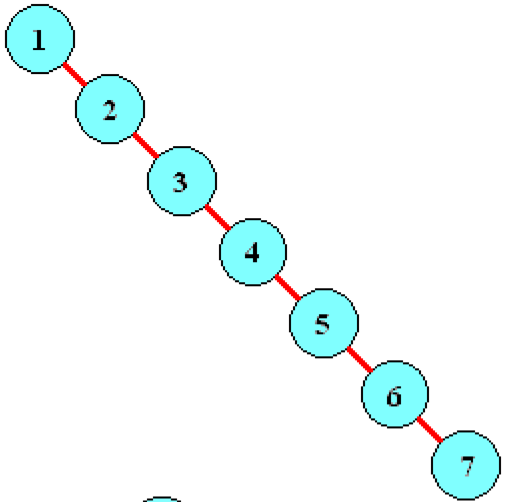
- 3 cases:
  - x has no children:
    - Remove x
  - x has one child:
    - Splice out x
  - x has two children:
    - Swap x with successor
    - Perform case 1 or 2 to delete it
- **Complexity**: O(h), where h is the height of the tree

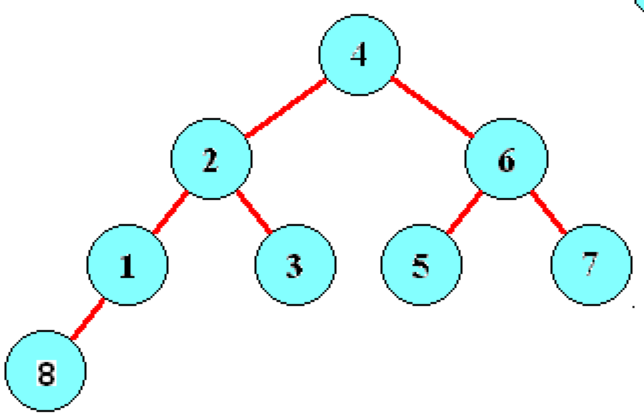
Deletion

```
delete (tree T, node z)
    // Determine which node y to splice out.
    if (left[z] = null | right[z] = null) then y = z else y = successor(z)
    // x is set preferably to a non-null child of y.
    if (left[y] ≠ null) then x = left[y] else x = right[y]
    // y is removed from the tree by manipulating pointers of p[y] and x.
    if (x ≠ null) then p[x] = p[y]
    if (p[y] = null) then
        root[T] = x
    else if (y = left[p[y]]) then left[p[y]] = x else right[p[y]] = x
    // If it was z's successor that was spliced out, copy its data into z.
    if (y ≠ z) then key[z] = key[y] and copy y's satellite data into z
```

Summary

- All the **basic operations** in **O(h) time**, where h is the height of the tree.
- **Worst case** height n - 1.
  - Degenerate tree
- **Best case** height lg n.
  - Compressed tree of size n
    - 2^i nodes at each level i
    - except maybe at last level.
  - Hence, $2^h \leq n$.
  - Hence, $h \leq \lg n$





Randomly built binary search trees

- The **average height** is much closer to the best case.
- Little is known about the average height when **both insertion and deletion** are used.
- **Randomly Built Binary Search Tree**
  – Keys inserting in **random order** into an initially empty tree.
  – Each of the n! **permutations** of the input keys is **equally likely**.

Randomly built binary search trees

- Theorem:
  - The **average depth** of a node in a randomly built binary search tree is O(logn).
- Proof:
  - Define **the internal depth** of a tree T to be the sum, over all nodes x in T, of the depth of x.
  - Let P(n) denotes the internal depth of a randomly built binary search tree T with n nodes.
  - We will show that P(n) = O(nlogn).
  - Thus, since T has n nodes, the average depth of a node in T will be P(n)/n = O(logn).

Randomly built binary search trees

- Notice:
  - The tree T will have one node as a root.
  - Thus, there will be n - 1 nodes distributed among the left and right sub-trees.
  - This distribution is random, and follows the pattern:

Left	Right
0	$n-1$
1	$n-2$
$\vdots$	$\vdots$
$n-2$	1
$n-1$	0

Randomly built binary search trees

- Since there are n total possible distributions, and they are all equally likely, we get:

$$
\begin{aligned}
P(n) &= \frac{1}{n}\sum_{i=0}^{n-1}(P(i)+P(n-i-1)+n-1) \\
&= \frac{1}{n}(\sum_{i=0}^{n-1}P(i)+\sum_{i=0}^{n-1}P(n-i-1)+\sum_{i=0}^{n-1}(n-1)) \\
&= \frac{1}{n}(\sum_{i=0}^{n-1}P(i)+\sum_{j=0}^{n-1}P(j)+n(n-1)) \\
&= \frac{2}{n}\sum_{i=0}^{n-1}P(i)+n-1 \\
&= \frac{2}{n}\sum_{i=0}^{n-1}P(i)+\Theta(n)
\end{aligned}
$$

Randomly built binary search trees

- Solve the recurrence $P(n) = \frac{2}{n}\sum_{q=1}^{n-1} P(q) + \Theta(n)$ by the substitution method.
- Guess: $P(n) \le a\,n\,\lg n + b$
- Substitute:

$$
\begin{aligned}
P(n) &= \frac{2}{n}\sum_{q=1}^{n-1} P(q) + \Theta(n) \\
&\le \frac{2}{n}\sum_{q=1}^{n-1} (aq\lg q + b) + \Theta(n) \\
&= \frac{2a}{n}\sum_{q=1}^{n-1} q\lg q + \frac{2b}{n}(n-1) + \Theta(n)
\end{aligned}
$$

Randomly built binary search trees

- Splitting the sum into two parts:

$$
\begin{aligned}
\sum_{q=1}^{n-1} q \lg q &= \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \lg q + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \lg q \\
&\leq \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \lg \frac{n}{2} + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \lg n \\
&= \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q(\lg n - 1) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \lg n \\
&= (\lg n - 1) \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + \lg n \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \\
&= \lg n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \\
&\leq \lg n \frac{n(n-1)}{2} - \frac{(n/2 - 1)n/2}{2} \\
&\leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2, \text{ if } n \geq 2
\end{aligned}
$$

Randomly built binary search trees

- Back to P(n), we get:

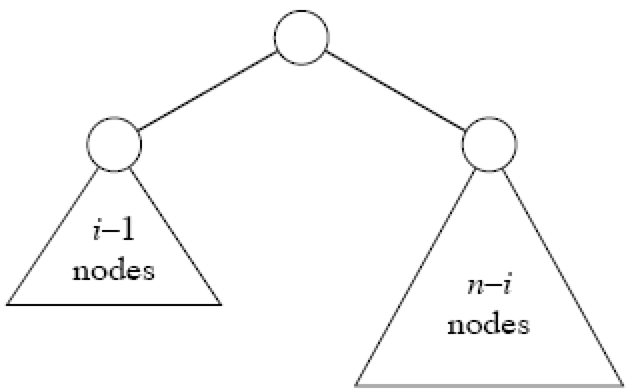
$$
\begin{aligned}
P(n) &\le \frac{2a}{n}\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + \frac{2b}{n}(n-1) + \Theta(n) \\
&\le an \lg n - \frac{a}{4}n + 2b + \Theta(n) \\
&= an \lg n + b + \left(\Theta(n) + b - \frac{a}{4}n\right) \\
&\le an \lg n + b
\end{aligned}
$$

Randomly built binary search trees

- **Theorem**:
  - The **average height** of a randomly-built binary search tree of n distinct keys is O(lg n)
- **Proof**:
  - Define the following random variables:
    - Xn - The **height** of a randomly built binary search tree on n keys.
    - $Yn = 2^{Xn}$ - The **exponential height.**
    - Rn - The **rank** of the root within the set of n keys used to build the binary search tree
      - Equally likely to be any element of {1, 2, . . . , n}.
      - If Rn = i , then
        - » Left subtree is a randomly-built binary search tree on i − 1 keys.
        - » Right subtree is a randomly-built binary search tree on n − i keys.
  - We will show that E[Yn] is polynomial in n, which will imply that E[Xn] = O(lg n).

Randomly built binary search trees

- Formula for Yn:
  - Knowing Rn = i



  - Yn = 2 · max(Yi−1, Yn−i) .
  - Y_1 = 1 (expected **exponential** height of a 1-node tree is 2^0 = 1).
  - Define Y_0 = 0.

Randomly built binary search trees

- Define **indicator random variables** $Z_{n,1}, Z_{n,2}, \ldots, Z_{n,n}$, where $Z_{n,i} = I\{Rn = i\}$
- Rn is equally likely to be any element of {1,2,…,n}
  - ⇒Pr {Rn = i } = 1/n
  - ⇒E[Zn,i] = 1/n (since E[I{A}] = Pr{A})

- Consider a given n-node binary search tree.
- Exactly one Zn,i is 1, and all others are 0.
- Hence, $Y_n = \sum_{i=1}^{n} Z_{n,i} \cdot (2 \cdot \max(Y_{i-1}, Y_{n-i}))$

Randomly built binary search trees

- Thus,

$$
\begin{aligned}
\mathrm{E}[Y_n] &= \mathrm{E}\left[\sum_{i=1}^{n} Z_{n,i}\,(2\cdot\max(Y_{i-1},Y_{n-i}))\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}\cdot(2\cdot\max(Y_{i-1},Y_{n-i}))] && \text{(linearity of expectation)} \\
&= \sum_{i=1}^{n} \mathrm{E}[Z_{n,i}]\cdot\mathrm{E}[2\cdot\max(Y_{i-1},Y_{n-i})] && (Z_{n,i}\text{ is independent of }Y_{i-1}\text{ and }Y_{n-i}) \\
&= \sum_{i=1}^{n} \frac{1}{n}\cdot\mathrm{E}[2\cdot\max(Y_{i-1},Y_{n-i})] && (\mathrm{E}[Z_{n,i}]=1/n) \\
&= \frac{2}{n}\sum_{i=1}^{n} \mathrm{E}[\max(Y_{i-1},Y_{n-i})] && (\mathrm{E}[aX]=a\,\mathrm{E}[X]) \\
&\le \frac{2}{n}\sum_{i=1}^{n} (\mathrm{E}[Y_{i-1}]+\mathrm{E}[Y_{n-i}]) && (\mathrm{E}[\max(X,Y)]\le \mathrm{E}[X]+\mathrm{E}[Y])
\end{aligned}
$$

Randomly built binary search trees

- Observe that the last summation is

$$(\mathrm{E}[Y_0] + \mathrm{E}[Y_{n-1}]) + \cdots + (\mathrm{E}[Y_{n-1}] + \mathrm{E}[Y_0]) = 2\sum_{i=0}^{n-1} \mathrm{E}[Y_i]$$

- Hence,

$$\mathrm{E}[Y_n] \le \frac{4}{n}\sum_{i=0}^{n-1} \mathrm{E}[Y_i]$$

- We will show that for all integers n > 0,

$$\mathrm{E}[Y_n] \le \frac{1}{4}\binom{n+3}{3}$$

- Hence,

$$\mathrm{E}[Y_n] \le \frac{1}{4}\binom{n+3}{3} = \frac{1}{4} \cdot \frac{(n+3)(n+2)(n+1)}{6} = O(n^3)$$

- But,

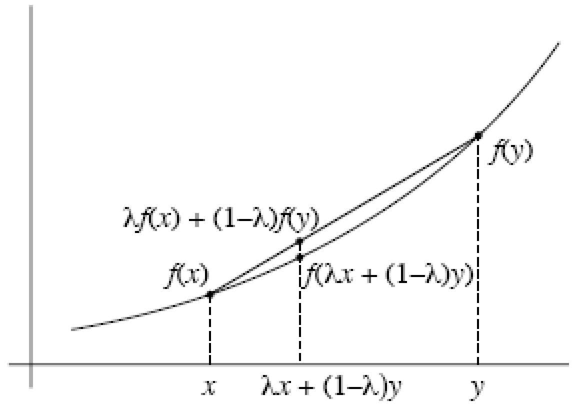
$$2^{\mathrm{E}[X_n]} \le \mathrm{E}[2^{X_n}] = \mathrm{E}[Y_n] \quad (\textbf{\textit{Jensen's inequality}}\text{: } \mathrm{E}[f(X)] \ge f(\mathrm{E}[X])$$
$$\text{provided the expectations exist and are finite, and } f(x) \text{ is } \textbf{\textit{convex}})$$

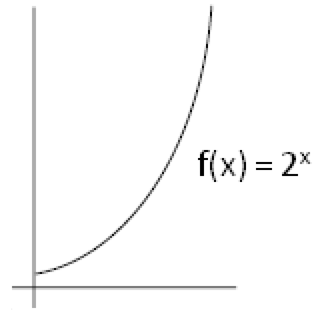
- Taking logs of both sides gives E[Xn] = O(lg n).

Convex Functions

f(x) is **convex** if for all x, y and all $0 \le \lambda \le 1$,

$f(\lambda x + (1 - \lambda)y) \le \lambda f(x) + (1 - \lambda) f(y)$.





Randomly built binary search trees

$$\mathrm{E}[Y_n] \le \frac{1}{4}\binom{n+3}{3}$$

Basis: $n = 1$.

$$1 = Y_1 = \mathrm{E}[Y_1] \le \frac{1}{4}\binom{1+3}{3} = \frac{1}{4} \cdot 4 = 1 .$$

Inductive step: Assume that $\mathrm{E}[Y_i] \le \frac{1}{4}\binom{i+3}{3}$ for all $i < n$. Then

$$\begin{aligned}
\mathrm{E}[Y_n] &\le \frac{4}{n}\sum_{i=0}^{n-1}\mathrm{E}[Y_i] && \text{(from before)} \\
&\le \frac{4}{n}\sum_{i=0}^{n-1}\frac{1}{4}\binom{i+3}{3} && \text{(inductive hypothesis)} \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\binom{i+3}{3} \\
&= \frac{1}{n}\binom{n+3}{4} && \text{(lemma)} \\
&= \frac{1}{n} \cdot \frac{(n+3)!}{4!\,(n-1)!} \\
&= \frac{1}{4} \cdot \frac{(n+3)!}{3!\,n!} \\
&= \frac{1}{4}\binom{n+3}{3} .
\end{aligned}$$

Randomly built binary search trees

Lemma

$$\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4} .$$

Proof Use Pascal's identity : $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

Also using the simple identity $\binom{4}{4} = 1 = \binom{3}{3}$, we have

$$\begin{aligned}
\binom{n+3}{4} &= \binom{n+2}{3} + \binom{n+2}{4} \\
&= \binom{n+2}{3} + \binom{n+1}{3} + \binom{n+1}{4} \\
&= \binom{n+2}{3} + \binom{n+1}{3} + \binom{n}{3} + \binom{n}{4} \\
&\;\vdots \\
&= \binom{n+2}{3} + \binom{n+1}{3} + \binom{n}{3} + \cdots + \binom{4}{3} + \binom{4}{4} \\
&= \sum_{i=0}^{n-1} \binom{i+3}{3} .
\end{aligned}$$