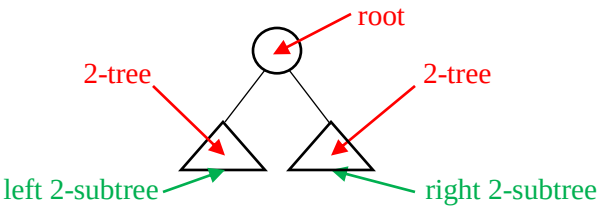
Transcribed and edited by Mr. Payman Khani

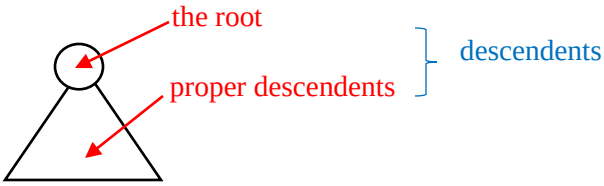
2-trees

A 2-tree △ is either:

1. An empty 2-tree □
2. A node ○ (the root of the 2-tree) and two 2-subtrees attached below it



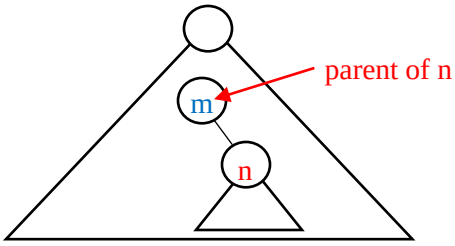
2-trees of kind 2 (above) are non-empty. They are visualized as:



Nodes ○ are also called internal nodes.

Empty trees □ are also called external nodes.

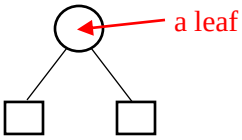
Let n be a node ((n)) in a tree T:



m is the parent of n and n is a child of m.

The root has no parents.

A leaf is an internal node (○) whose both 2-subtrees are empty.



Property of 2-trees: Each internal node has 2 children (○'s, □'s, or ○ and □).

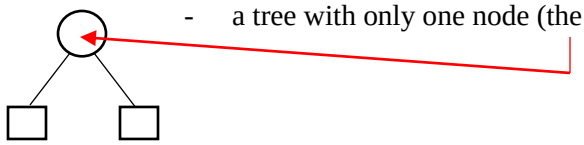
Decision tree is a 2-tree.
In decision trees, internal nodes (◯) represent decisions, while external nodes (□) represent outcomes.

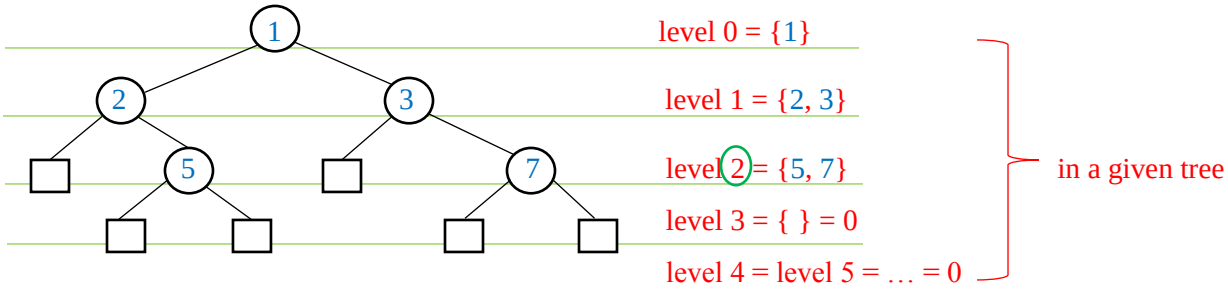
A binary tree is a 2-tree pruned of its external nodes (□).

Examples of 2-trees:

□ - the empty tree (it has no root).

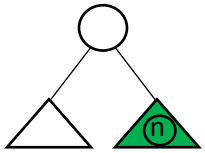
- a tree with only one node (the root).





Definition of level

- Each level is a set of nodes.
- Levels are enumerated with natural numbers:
  level 0, level 1, level 2, …
- If the tree is empty then all its levels are empty.
- If the tree is non-empty then its level 0 consists of one node only: its root.
- In the tree T



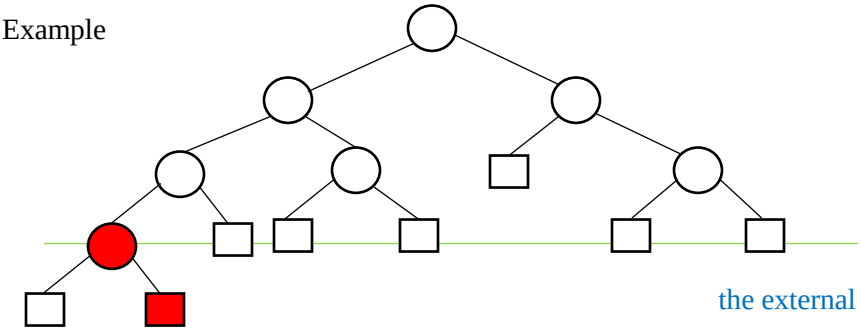
the level of any node n of T is equal to the level of n in ▲ plus 1.

- All other levels of the tree are empty.

The level of a node ◯ in a decision tree T is the number of decisions made in T before making decision ◯.

The number of decisions made in a decision tree T in order to reach outcome □ is equal to the level of the parent of that outcome □ plus 1.

Example



level 0

level 1

level 2

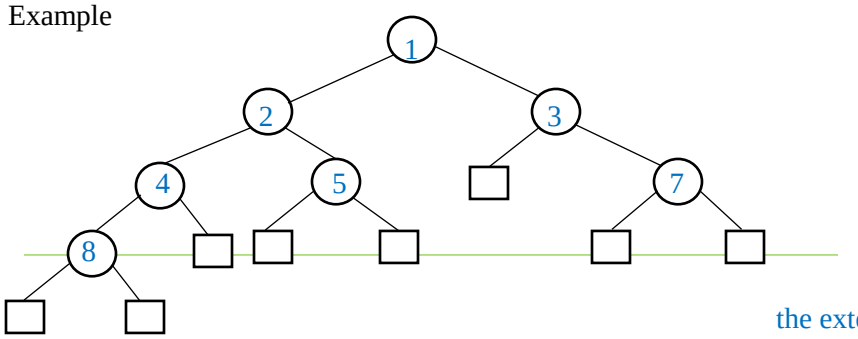
level 3 (last level)

the external level 4

The number of decisions made before reaching decision ● is level (●) = 3.
The number of decisions made to reach outcome ■ is level (●) + 1 = 3 + 1 = 4 (the external level of ■).

Definition of the depth of tree T
It is the level number of the last non-empty level of T.

Example



level 0 = {1}

level 1 = {2, 3}

level 2 = {4, 5, 7}

level 3 = {8}

the external level 4 = 0

The above tree has depth 3. (= ⌊ lg 8 ⌋)

WARNING

Note Some authors (including myself) use the term "height" in the meaning of "depth".
Some others use "height" as "depth + 1".
Make sure you read relevant definition first before jumping to the text.

Internal path length in 2-tree T

$$ipl(T) = \sum_{i \in I(T)} level(i)$$

where I (T) is the set of all internal nodes of T.

External path length in 2-tree T

$$epl(T) = \sum_{i \in E(T)} level(i)$$

where E (T) is the set of all external nodes of T.

Note. With appropriate enumeration of nodes of the tree (like the one that was used in examples), the level level(i) to which a node i belongs is given by this formula:

level (i) = $\lfloor \lg i \rfloor$

This formula was derived in Data Structures class (heaps, priority queues). We will derive it later.

Under such arrangement

ipl (T) = $\sum_{i \in I(T)} \lfloor \lg i \rfloor$

and

epl (T) = $\sum_{i \in E(T)} \lfloor \lg i \rfloor$

This convention greatly simplifies proofs of several theorems in the textbook.

Theorem 1 For every 2-tree T_n with n internal nodes,

$$epl(T_n) = ipl(T_n) + 2n$$

Proof in file Internal_External_Path_Length.pdf

The above fact can be used to prove that if a search in an ordered array by decision tree is average-case optimal then it is worst-case optimal; the average number of comps is

$$C_{avg} = \frac{ipl\ (D) + n}{n} = ipl\ \frac{D}{n} + 1$$

where D is a decision tree of the search on a n-element ordered array, so if it is minimal then epl (D) is minimal, too. This can only happen if D has external nodes on last 2 levels only, and that implies that D is the shortest decision tree, thus making the search in question optimal.

Balanced 2-trees

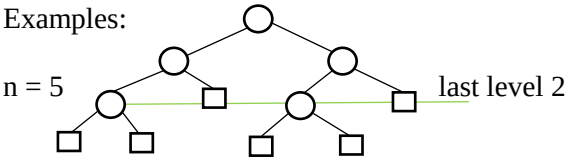
Definition

A 2-tree T with n internal nodes is called a balanced 2-tree if, and only if, it has the shortest epl (T) among all 2-trees with n internal nodes.

It follows that T is balanced iff it has the shortest ipl (T) among all such 2-trees (ipl (T) = epl (T) - 2n)

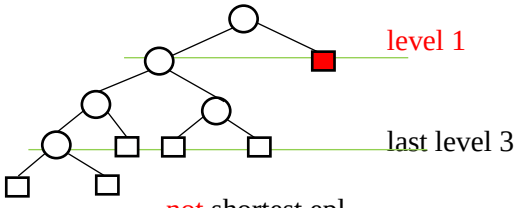
Theorem A 2-tree T has the shortest epl (T) among all 2-trees with n internal nodes if, and only if, there are no external nodes □ above the last level of T.

Examples:



Shortest epl

epl = 2 + 2 +3 + 3 + 3 + 3 = 16



not shortest epl

epl = 1 + 3 + 3 + 3 + 4 + 4 = 18

Proof Because epl (T) = ipl (T) + 2n, it suffices to prove that the above condition characterizes the shortest ipl (T).

Indeed, since given any tree T on n nodes,

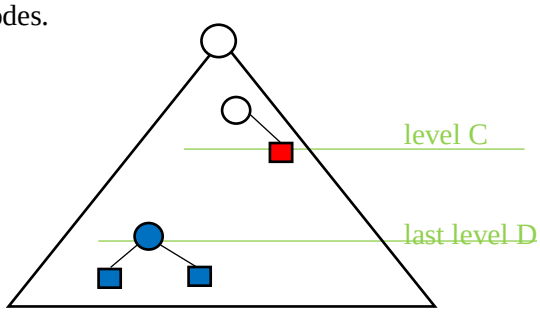
ipl (T) = $\sum_{i \in I(T)} \lfloor \lg i \rfloor$,

ipl (T) has the minimal value iff the values of $\lfloor \lg i \rfloor$ are minimal for all $i \in I(T)$.
In order to accomplish that, all internal nodes i of T must be pushed up as far as possible. That would mean that all levels of T, except perhaps in the leaf level of T, must have the maximal numbers of nodes: This would leave no room for any external nodes above the leaf level of T.
This completes the proof.

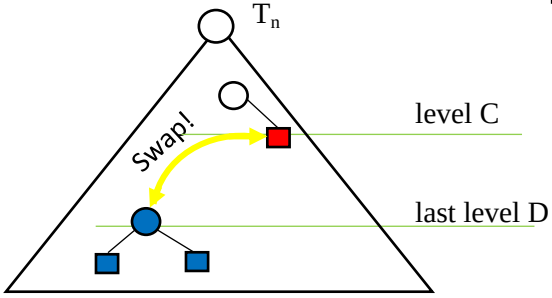
Here is the proof from the textbook:

Proof by contradiction.

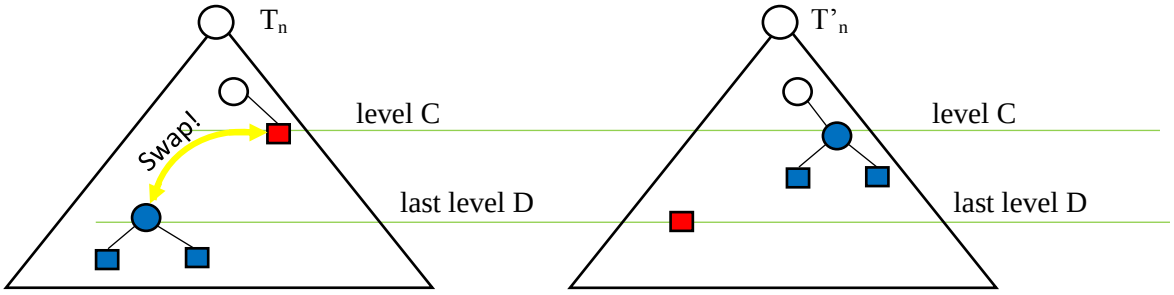
Let T_n be a 2-tree with n internal nodes, and an external node ■ above the last non-empty level of T_n, and the shortest epl (T_n) among all 2-trees with n nodes.



Let's consider a 2-tree T'_n that was a result of swapping a leaf ● and its empty subtrees ■ ■ ■ with the external node ■.



This swap will result in this 2-tree:



As a result of the swap, two paths were shortened by D-C and one path were prolonged by D-C. In total, epl (T'_n) = epl (T_n) - (D - C).

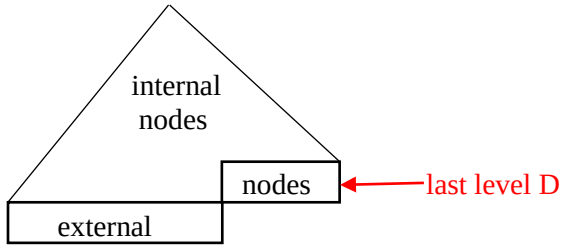
So, epl (T'_n) < epl (T_n), contradicting the assertion that epl (T_n) was the shortest among all 2-trees with n internal nodes.

This completes the proof.

One can conclude from the above theorem that binary search is optimal in class of search of an ordered array by a decision tree, because the decision tree D of binary search had external nodes on last two levels only, so it had minimal ipl (D), which means that

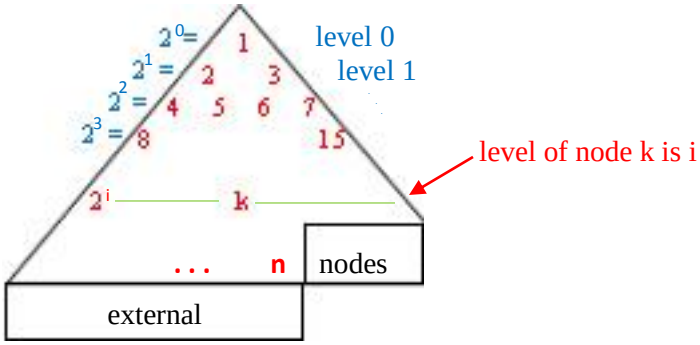
$C_{avg} = \frac{ipl\,(D)+n}{n} = ipl\frac{D}{n} + 1$ was minimal as well.

We will visualize balanced 2-trees as if their all internal nodes in the last level was flushed all the way to the left:



Properties of balanced 2-trees

Let's enumerate, level-by-level, from the left to the right all internal nodes of a balanced 2-tree:



What is the level of node k?

$$\text{level}(k) = \max \{ i \mid 2^i \leq k \}$$

or (because lg is an increasing function)

$$\text{level}(k) = \max \{ i \mid \lg 2^i \leq \lg k \}$$

that is,

$$\text{level}(k) = \max \{ i \mid i \leq \lg k \}$$

(Recall that $\lfloor x \rfloor = \max \{ i \mid i \leq x \}$ and substitute lg k for x.)

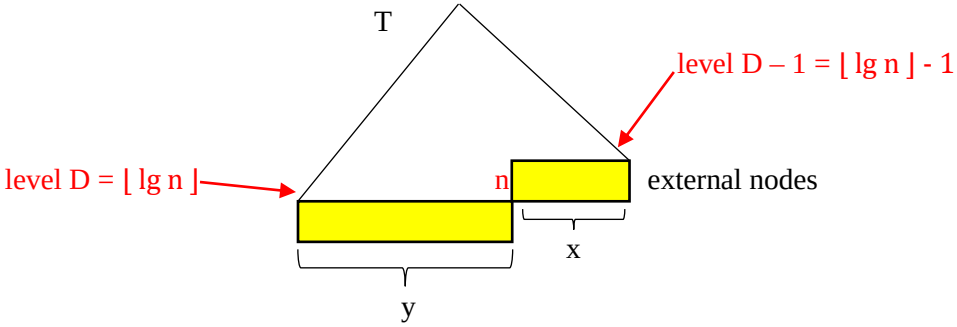
$$\max \{ i \mid i \leq \lg k \} = \lfloor \lg k \rfloor$$

Hence, $\text{level}(k) = \lfloor \lg k \rfloor$.

In particular, since the last node belongs to the last non-empty level, and the depth of any tree is the number of its last non-empty level, the depth of any balanced 2-three on n nodes is

$$D = \text{level}(n) = \lfloor \lg n \rfloor.$$

External path length in balanced 2-tree



epl (T) = (⌊ lg n ⌋ - 1 + 1) x + (⌊ lg n ⌋ + 1) y
= ⌊ lg n ⌋ x + ⌊ lg n ⌋ y + y = ⌊ lg n ⌋ (x + y) + y
= ⌊ lg n ⌋(n + 1) + y

All we have to do is to find y.

See file BS_decision_tree_slides.pdf for derivation of this equation:

$$x + \frac{y}{2} = 2^{\lfloor \lg n \rfloor}$$

Combining it with $x + y = n + 1$, we conclude

$$\frac{y}{2} = n + 1 - 2^{\lfloor \lg n \rfloor}$$

or

$$y = 2n - 2^{\lfloor \lg n \rfloor + 1} + 2$$

Hence,

$$epl(T) = (n+1)\lfloor \lg n \rfloor + 2n - 2^{\lfloor \lg n \rfloor + 1} + 2$$

Since $ipl(T) = epl(T) - 2n$,

$$ipl(T) = (n+1)\lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor + 1} + 2n + 2 - 2n$$

$$= (n+1)\lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor + 1} + 2.$$

Since for balanced tree, ipl (T) $= \sum_{i=1}^{n} \lfloor \lg i \rfloor$, we obtained a useful result:

$$\sum_{i=1}^{n} \lfloor \lg i \rfloor = (n+1) \lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor + 1} + 2$$

Also, see file Knuth-Suchenek_formulas_sums_of_floors_ceilings_logs.pdf

Also, one can show (exercise: show it using the method in the proof presented in class of the worst-case number of comps by Mergesort, figure 4.14 in textbook) that the best-case number of comps done by Quicksort while sorting n-element array is the same as ipl (T) where T is a balanced 2 tree of n nodes, that is: $(n+1)\lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor + 1} + 2$.

Finally, simplifying expression for epl (T), we conclude

$$\begin{aligned} \text{epl}(T) &= (n+1)(\lfloor \lg n \rfloor + 1) - 2^{\lfloor \lg n \rfloor + 1} + n + 1 \\ &= (n+1)\lceil \lg (n+1) \rceil - 2^{\lceil \lg (n+1) \rceil} + n + 1 \\ &= (n+1)(\lceil \lg (n+1) \rceil + 1) - 2^{\lceil \lg (n+1) \rceil} \\ &= m\,(\lceil \lg m \rceil + 1) - 2^{\lceil \lg m \rceil} \end{aligned}$$

where $m = n + 1$ is the number of external nodes in T.

Note: epl (T) in a balanced 2-tree with n nodes is quite close to the worst-case number of comps by Mergesort on n-element array:

$$n \lceil \lg (n+1) \rceil - 2^{\lceil \lg (n+1) \rceil} + 1$$

$$= n \lceil \lg (n) \rceil - 2^{\lceil \lg (n) \rceil} + 1$$

Make sure you never confuse these two!

We will use the formula for epl in order to establish tight lower bound for average number of comps while sorting an n-element array by decision tree.

We will provide close approximation for epl, too.

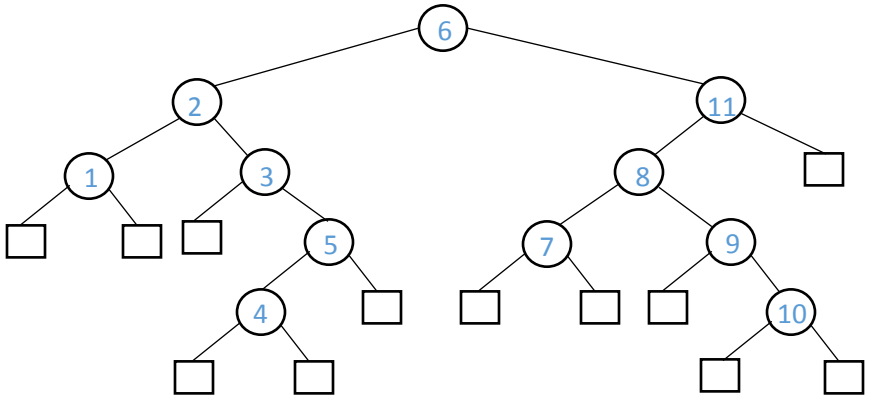
The formula for ipl is used, among other things, to quickly derive lower bound on search by decision tree and optimality of binary search.

This section is optional

Average ipl (T)

Now, we will compute the ipl (T_n) in an "average 2-tree T_n" with n nodes, which we define as an "average binary search (or simply, B.S.) 2-tree".

Here is example of a B.S. 2-tree for unsuccessful search that has been constructed by a sequence of consecutive insertions of 6, 2, 1, 3, 5, 4, 11, 8, 7, 9, 10 into initially empty B.S. 2-tree.



What we call an "average B.S. tree T_n" with n nodes is a result of consecutive insertion of "average" permutation of first n natural numbers , …, n into an initially empty B.S. tree.

(This process is sometimes referred to as the B.S. tree sort if followed by the in-order traversal of the tree T_n constructed this way.)

We assume that all n! permutations of these numbers have the same probability $\frac{1}{n!}$

In other words, the probability distribution on the set of all B.S. trees with n nodes is not even: the probability of a B.S. tree T is $\frac{1}{n!}$ times the number of permutations of numbers 1, …, n that – if inserted consecutively to an initially empty tree – result in the tree T.

We will call such a permutation the creative permutation of T.

It follows (show it!) that the total number of comps during construction (by means of consecutive insertions) of the tree T_n is equal to ipl (T_n) after all said insertions took place.

Now, think of the creative permutation π as a sequence of pivots that Quicksort will select while sorting some (perhaps different) permutation of numbers 1, …, n.

It follows that the number of comps during the execution of that Quicksort is the same as the number of comps while constructing a B.S. tree T out of creative permutation π, or – in other words – to ipl (T).

(Exercise: Prove it!)

File average_case_Quicksort.nb contains derivation (by means of experimental mathematics) of this number

$$A(n) = 1.386\,(n+1)\lg n - 2.846n - 2.156 + \frac{5}{6n} - \frac{1}{6n^2}$$

which is equal to ipl (T_n) in an "average B.S. 2-tree T_n" with n internal nodes.

From this we conclude that the epl (T_n) in an average 2-tree of n internal nodes is

$$A(n) + 2n = 1.386\,(n+1)\lg n - 0.846n - 2.156 + \frac{5}{6n} - \frac{1}{6n^2}$$

In conclusion, we derive formulas for an average number of comparisons in an unsuccessful and successful searches in an "average B.S. 2-tree" with n nodes.

Successful search:

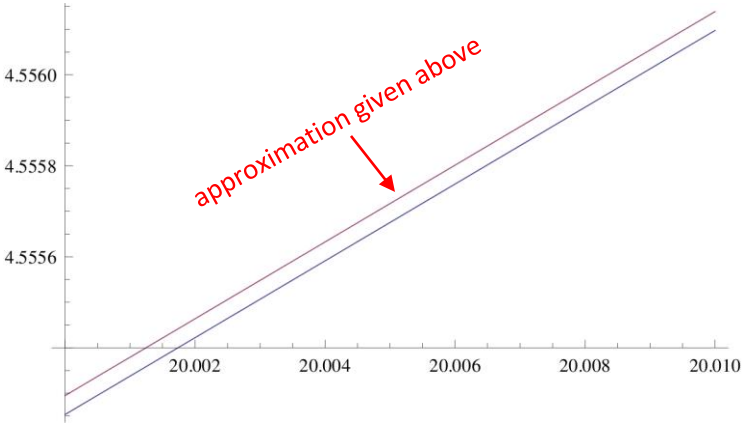
$$C_n = \frac{ipl\,(T_n)}{n} + 1$$

$$= \frac{1.386\,(n+1)\lg n - 2.846n - 2.156 + \frac{5}{6n} - \frac{1}{6n^2}}{n} + 1$$

$$= 1.386\lg n - 1.846 + 1.386\frac{\lg n}{n} - \frac{2.154}{n} + \frac{5}{6n^2} - \frac{1}{6n^3}$$

(since these are n successful searches in T_n, and the number of comps along a path p in T_n is the length of $p + 1$).

Avg. number of comps C_n in successful search



The difference between the two lines here is about 0.0001 and it converges to 0 faster than $\frac{1}{n^3}$

exact number

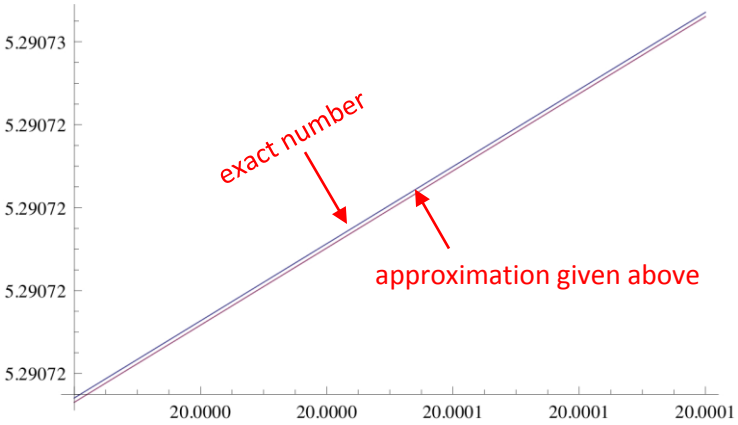
Unsuccessful search:

$$C_n' = \frac{epl\,(T_n)}{n+1} \approx \frac{1.386\,(n+1)\lg n - 0.846n + 2.154 + \frac{0.833}{n} - \frac{0.167}{n^2}}{n+1}$$

$$\approx 1.386 \lg n - 0.846 + \frac{1}{n} + \frac{2}{n+1} - \frac{1}{6n^2}$$

(since there are $n + 1$ unsuccessful searches in T_n).

The detailed calculations are in the file Average_search_BS_tree.nb



The difference between the two lines here is < 0.000001 and it converges to 0 faster than $\frac{1}{n^3}$