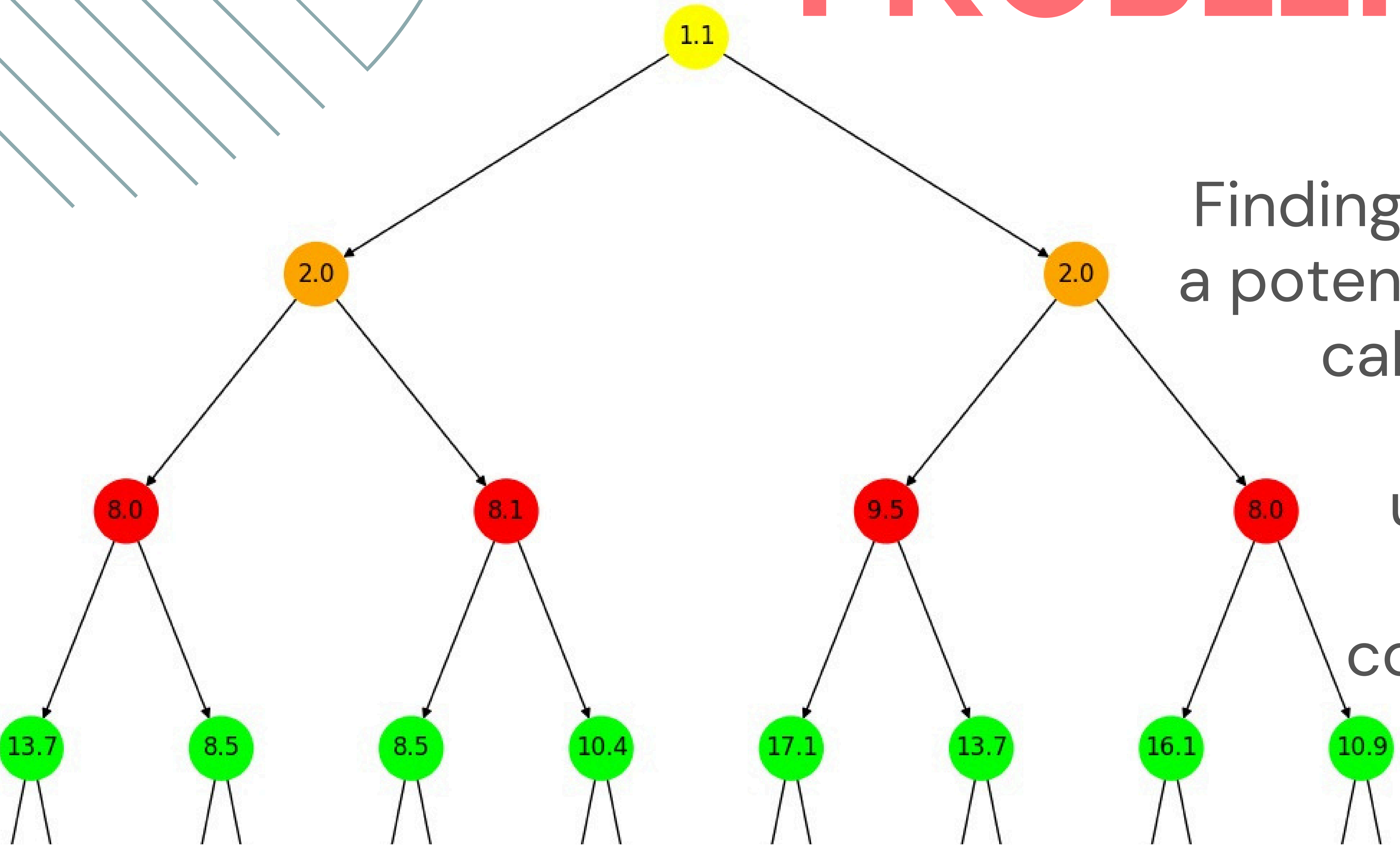
A NEARLY OPTIMAL RANDOMIZED ALGORITHM FOR EXPLORABLE HEAP SELECTION

BY SANDER BORST, DANIEL DADUSH, SOPHIE HUIBERTS, AND DANISH KASHAEV

Project by:
Meesum Abbas - ma08056
Musab Naeem Kasbati - mk07811

PROBLEM



Finding the n-th smallest element of a potentially infinite binary heap - we call this operation Select(n)

+

using O(log n) memory

+

consecutive heap queries must be local

Select(i) = [1.1, 2.0, 2.0, 8.0, 8.0, 8.1, ...]

MOTIVATION

Integer Linear Programming
Makes use of Branch and Bound, which implicitly considers its search space as a binary tree with monotonic paths from the root.

Local queries have overlapping constraints/computation, which can allow for faster computation

Many optimization problems can be modelled as ILP:

- 0-1 Knapsack
- Graph Coloring
- Travelling Salesman Problem

MOTIVATION

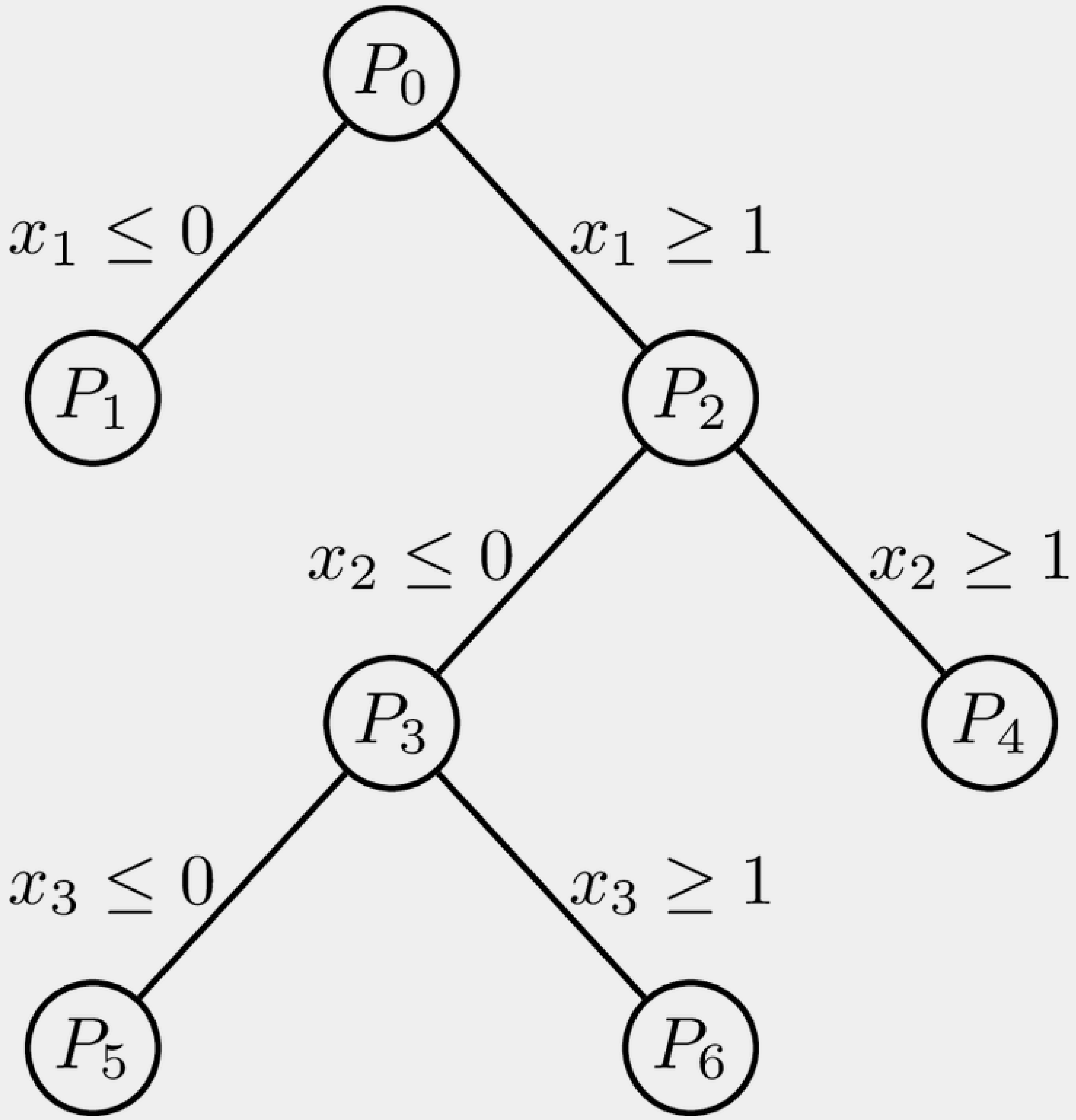


Image source: https://tikz.net/branch-and-bound/

Branch and Bound Search Space
For a maximisation objective, the best value possible gets smaller as more constraints are added (heap property satisfied)

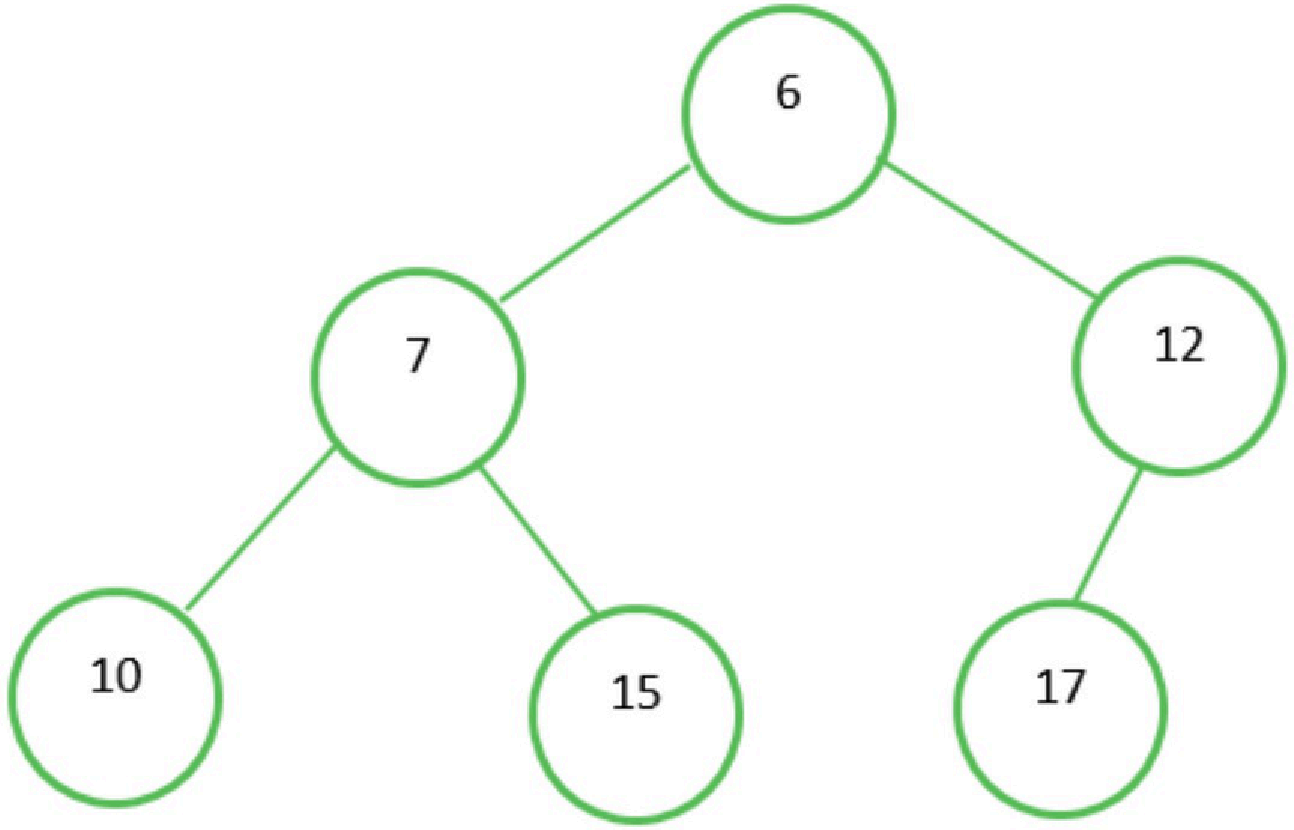
Note that it is easy to get best value solutions as they ignore the integer constraint/consider the problem as a Linear Programming problem

NAIVE APPROACH

Ignore the locality and memory constraints:

1. Keep track of the smallest unexplored nodes found so far in a priority queue - initially contains root.
2. Explore the children of the smallest unexplored node
3. Repeat until we have explored n nodes → return the value of the n-th node explored

Best First Approach



Select(i) = [6, 7, 10, 12, 15, 17]

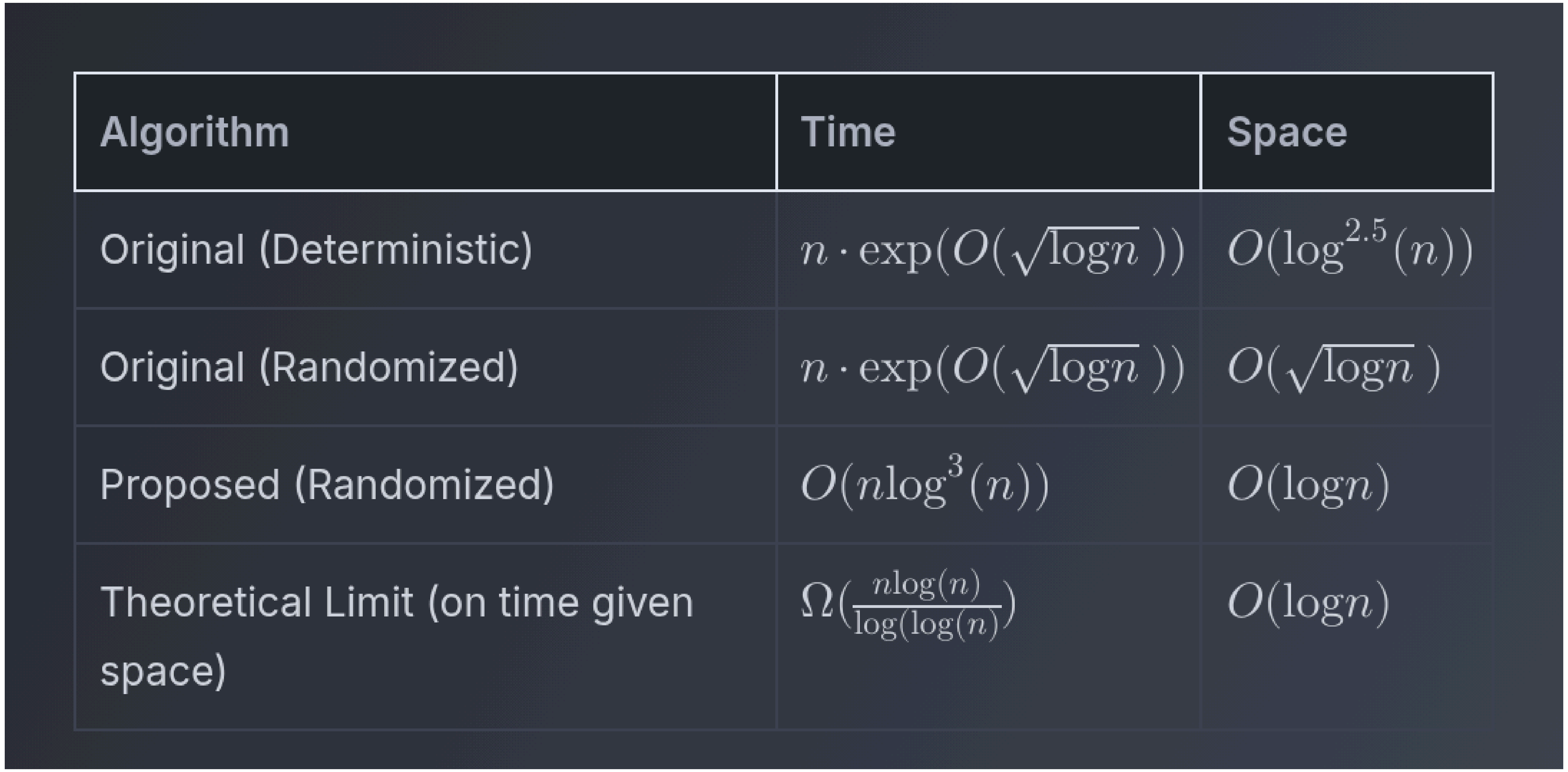
Time: O(n log n)

Space: O(n)

Image source: https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/

PRIOR WORK

Original work by: Richard M Karp, Michael E Saks, and Avi Wigderson (Turing Award Recipient 2023)
Paper: *On a search problem related to branch-and-bound procedures.*

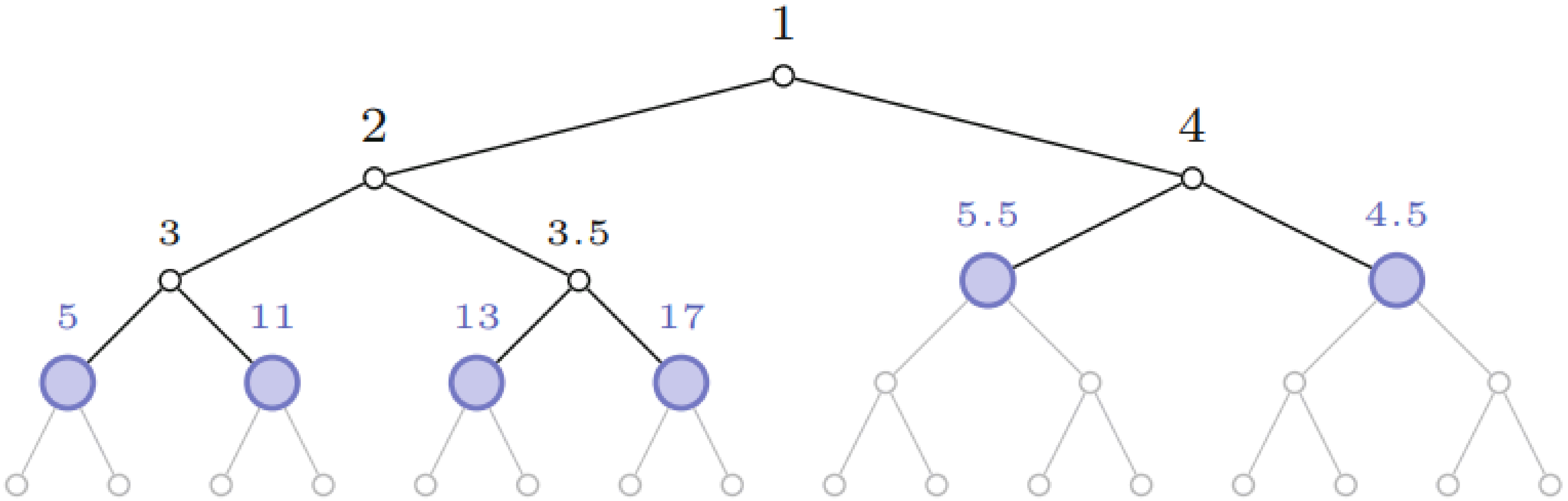


Algorithm	Time	Space
Original (Deterministic)	$n \cdot \exp(O(\sqrt{\log n}))$	$O(\log^{2.5}(n))$
Original (Randomized)	$n \cdot \exp(O(\sqrt{\log n}))$	$O(\sqrt{\log n})$
Proposed (Randomized)	$O(n\log^{3}(n))$	$O(\log n)$
Theoretical Limit (on time given space)	$\Omega(\frac{n\log(n)}{\log(\log(n))})$	$O(\log n)$

ALGORITHM

CORE IDEAS

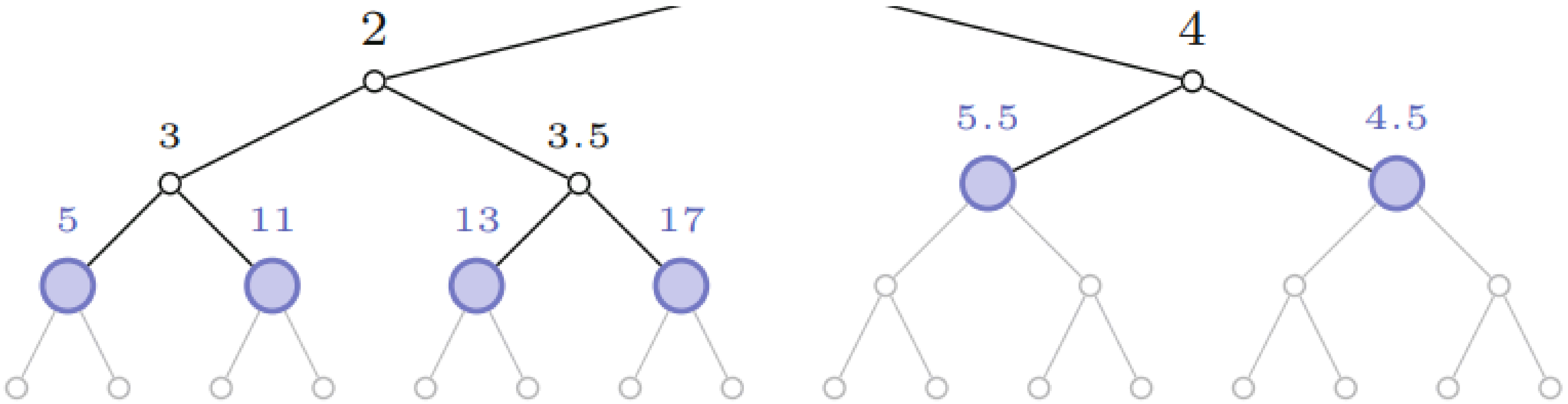
- Notion of good value - value less than n-th largest value: easy to check if a given value is good via dfs with a counter (note that this dfs must be implemented to use O(1) space).
- The algorithm expands a good subtree, i.e. if it knows the n smallest elements, it expands to find the 2n smallest elements
- The subtree can be expanded from any of its immediate children



ALGORITHM

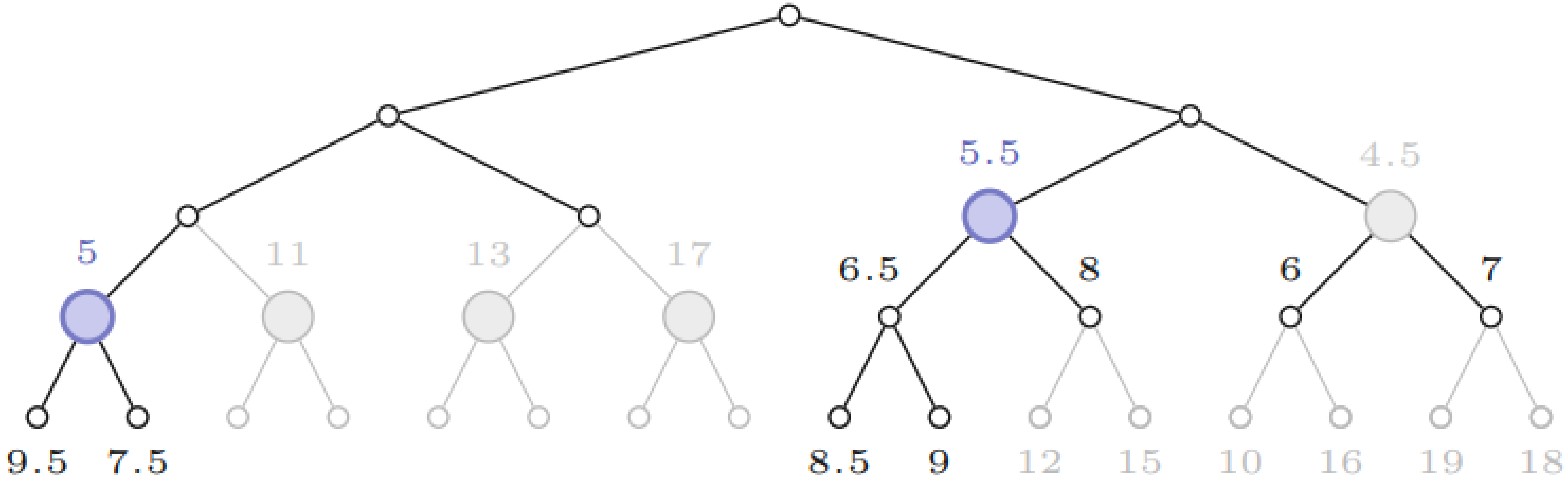
CORE IDEAS

- If I need to find k more nodes, I run Select(k) from a randomly selected node
- From this subtree, determine a range of answers by: largest good value, smallest bad value
- Select the next subtree to explore (which has some value within strictly within this range - as that can make my range tighter)
- explore until the range converges



ALGORITHM

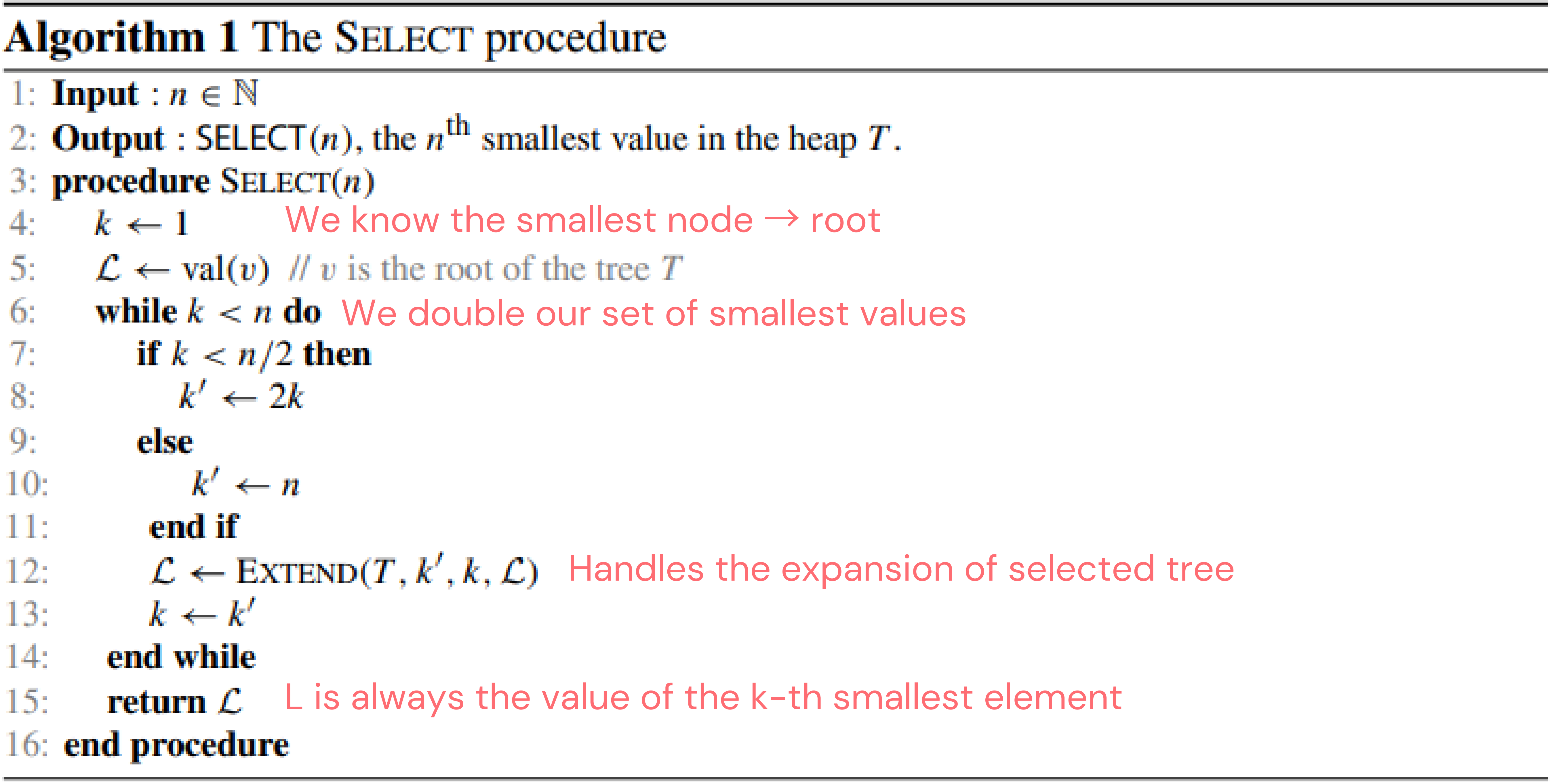
CORE IDEAS



Lower Bound: 7, Upper Bound: 10

ALGORITHM

PSEUDOCODE

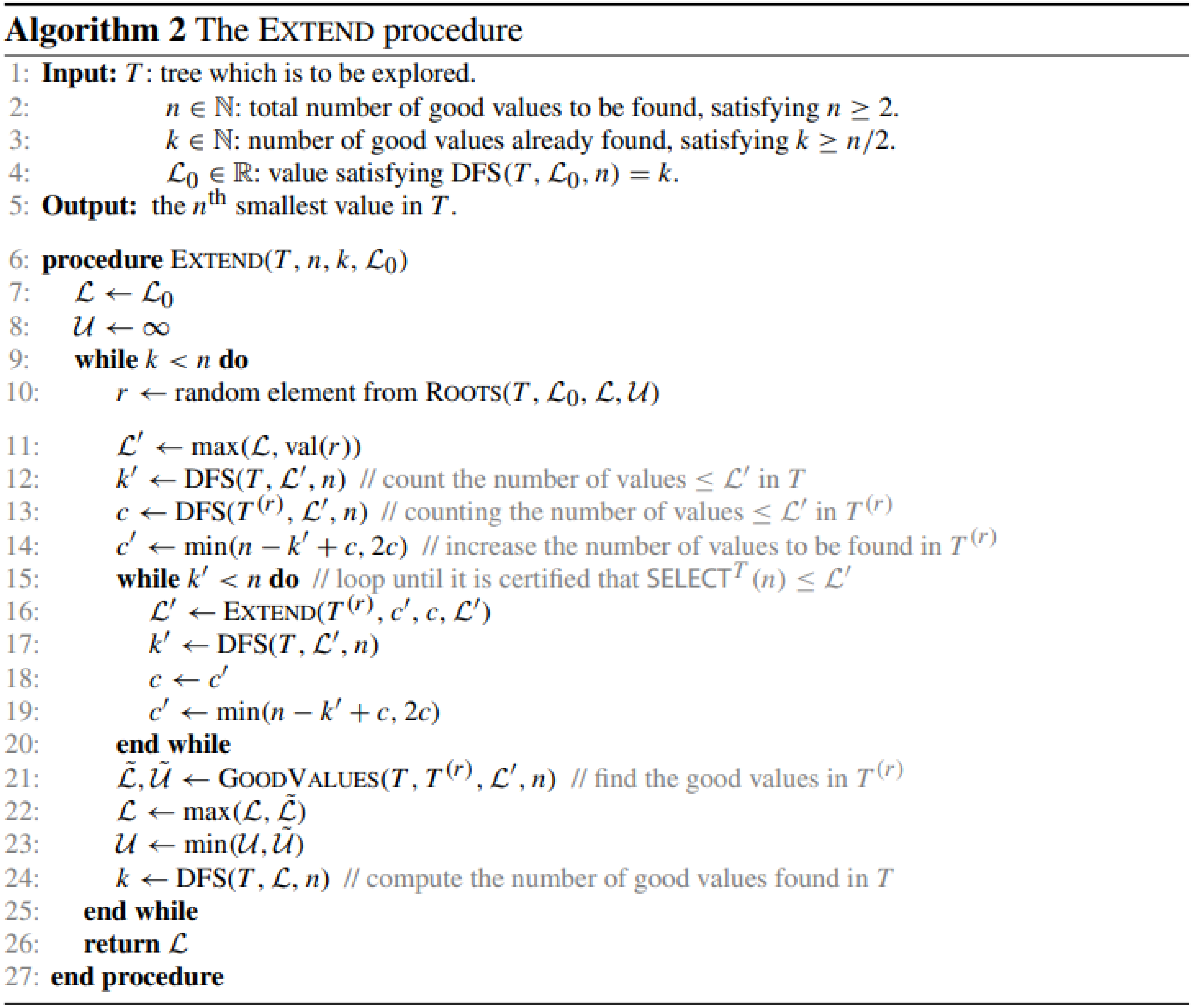


Algorithm 1 The SELECT procedure

```
Input : $n \in \mathbb{N}$
Output : SELECT($n$), the $n^{\text{th}}$ smallest value in the heap $T$.
procedure SELECT($n$)
    $k \leftarrow 1$        We know the smallest node → root
    $\mathcal{L} \leftarrow \text{val}(v)$  // $v$ is the root of the tree $T$
    while $k < n$ do   We double our set of smallest values
        if $k < n/2$ then
            $k' \leftarrow 2k$
        else
            $k' \leftarrow n$
        end if
        $\mathcal{L} \leftarrow$ EXTEND($T, k', k, \mathcal{L}$)   Handles the expansion of selected tree
        $k \leftarrow k'$
    end while
    return $\mathcal{L}$   L is always the value of the k-th smallest element
end procedure
```

ALGORITHM

PSEUDOCODE



Algorithm 2 The EXTEND procedure

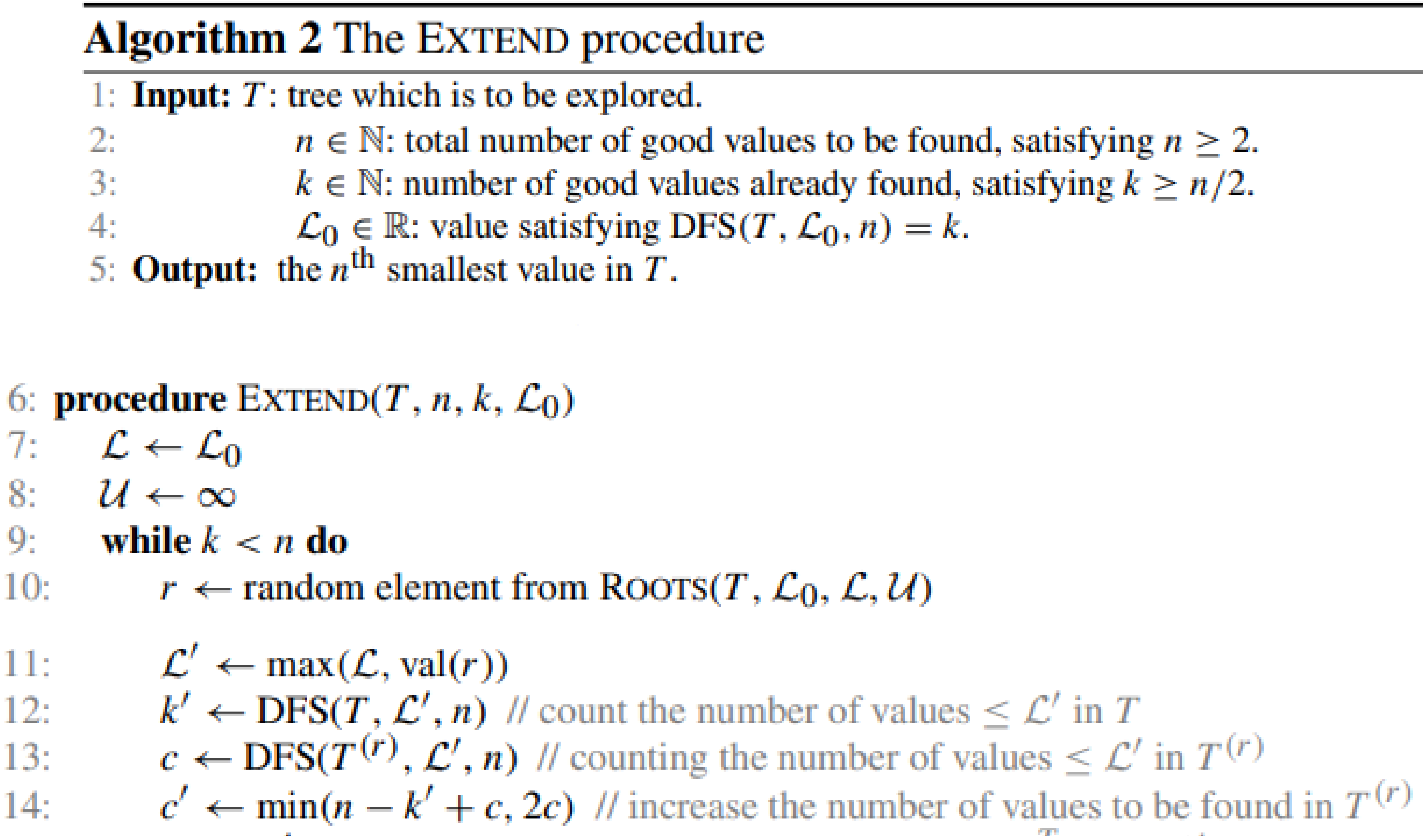
```
Input: $T$: tree which is to be explored.
            $n \in \mathbb{N}$: total number of good values to be found, satisfying $n \geq 2$.
            $k \in \mathbb{N}$: number of good values already found, satisfying $k \geq n/2$.
            $\mathcal{L}_0 \in \mathbb{R}$: value satisfying $\mathrm{DFS}(T, \mathcal{L}_0, n) = k$.
Output: the $n^{\text{th}}$ smallest value in $T$.

procedure EXTEND($T, n, k, \mathcal{L}_0$)
    $\mathcal{L} \leftarrow \mathcal{L}_0$
    $\mathcal{U} \leftarrow \infty$
    while $k < n$ do
        $r \leftarrow$ random element from ROOTS($T, \mathcal{L}_0, \mathcal{L}, \mathcal{U}$)

        $\mathcal{L}' \leftarrow \max(\mathcal{L}, \mathrm{val}(r))$
        $k' \leftarrow \mathrm{DFS}(T, \mathcal{L}', n)$ // count the number of values $\leq \mathcal{L}'$ in $T$
        $c \leftarrow \mathrm{DFS}(T^{(r)}, \mathcal{L}', n)$ // counting the number of values $\leq \mathcal{L}'$ in $T^{(r)}$
        $c' \leftarrow \min(n - k' + c, 2c)$ // increase the number of values to be found in $T^{(r)}$
        while $k' < n$ do // loop until it is certified that $\mathrm{SELECT}^T(n) \leq \mathcal{L}'$
            $\mathcal{L}' \leftarrow$ EXTEND($T^{(r)}, c', c, \mathcal{L}'$)
            $k' \leftarrow \mathrm{DFS}(T, \mathcal{L}', n)$
            $c \leftarrow c'$
            $c' \leftarrow \min(n - k' + c, 2c)$
        end while
        $\tilde{\mathcal{L}}, \tilde{\mathcal{U}} \leftarrow$ GOODVALUES($T, T^{(r)}, \mathcal{L}', n$) // find the good values in $T^{(r)}$
        $\mathcal{L} \leftarrow \max(\mathcal{L}, \tilde{\mathcal{L}})$
        $\mathcal{U} \leftarrow \min(\mathcal{U}, \tilde{\mathcal{U}})$
        $k \leftarrow \mathrm{DFS}(T, \mathcal{L}, n)$ // compute the number of good values found in $T$
    end while
    return $\mathcal{L}$
end procedure
```

ALGORITHM

PSEUDOCODE



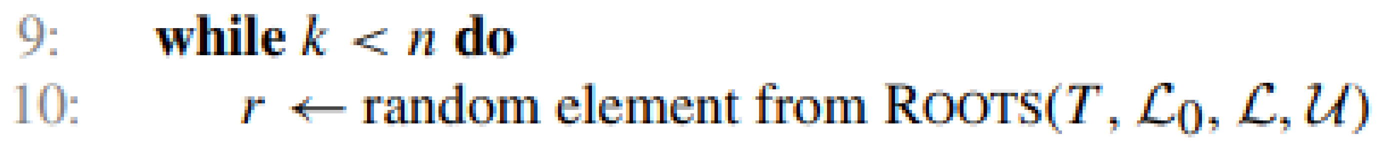
Algorithm 2 The EXTEND procedure

```
Input: $T$: tree which is to be explored.
          $n \in \mathbb{N}$: total number of good values to be found, satisfying $n \geq 2$.
          $k \in \mathbb{N}$: number of good values already found, satisfying $k \geq n/2$.
          $\mathcal{L}_0 \in \mathbb{R}$: value satisfying $\mathrm{DFS}(T, \mathcal{L}_0, n) = k$.
Output: the $n^{\text{th}}$ smallest value in $T$.

procedure EXTEND($T, n, k, \mathcal{L}_0$)
    $\mathcal{L} \leftarrow \mathcal{L}_0$
    $\mathcal{U} \leftarrow \infty$
    while $k < n$ do
        $r \leftarrow$ random element from ROOTS($T, \mathcal{L}_0, \mathcal{L}, \mathcal{U}$)
        $\mathcal{L}' \leftarrow \max(\mathcal{L}, \mathrm{val}(r))$
        $k' \leftarrow \mathrm{DFS}(T, \mathcal{L}', n)$  // count the number of values $\leq \mathcal{L}'$ in $T$
        $c \leftarrow \mathrm{DFS}(T^{(r)}, \mathcal{L}', n)$  // counting the number of values $\leq \mathcal{L}'$ in $T^{(r)}$
        $c' \leftarrow \min(n - k' + c, 2c)$  // increase the number of values to be found in $T^{(r)}$
```

ALGORITHM

PSEUDOCODE

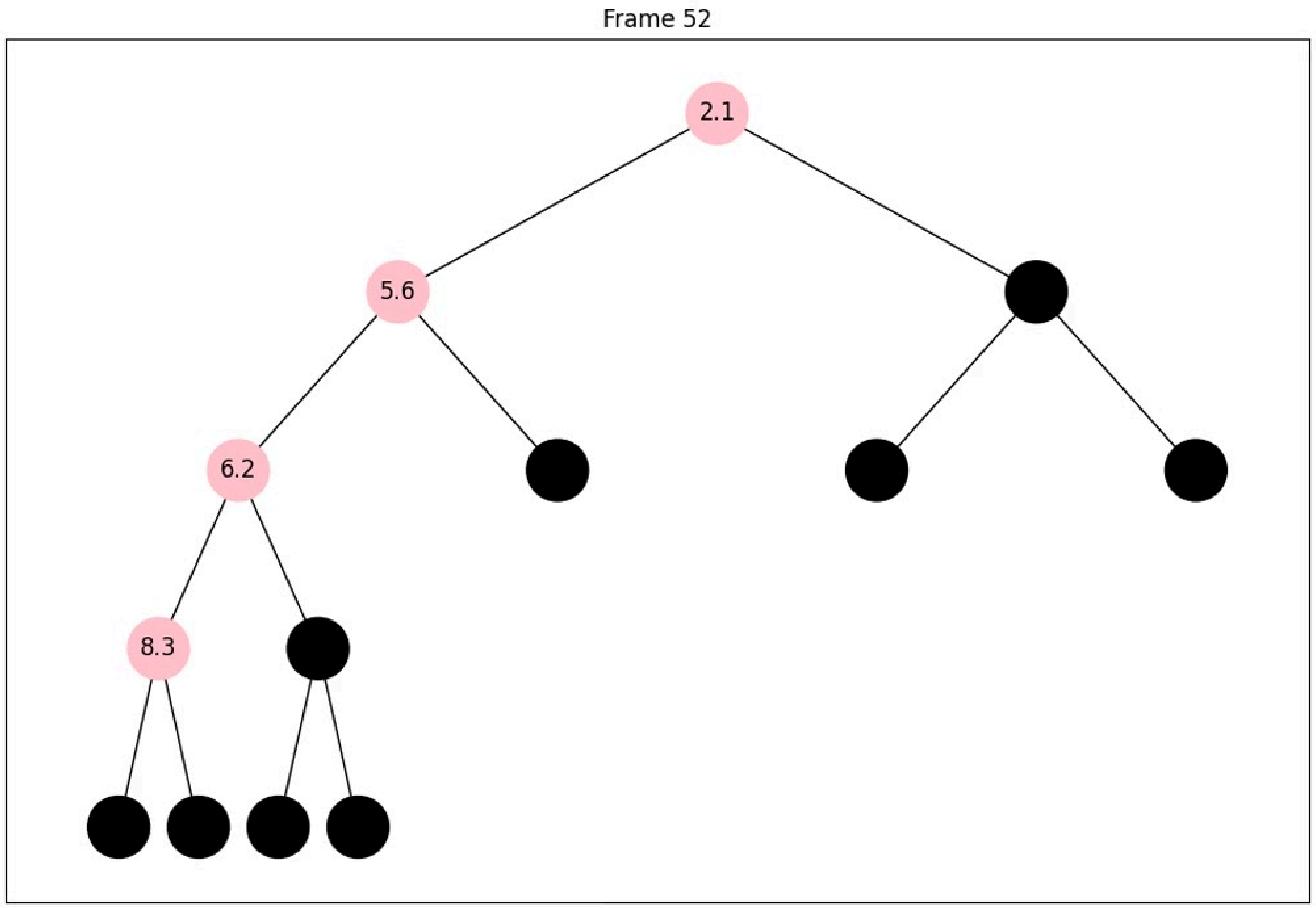


```
while $k < n$ do
    $r \leftarrow$ random element from ROOTS($T$, $\mathcal{L}_0$, $\mathcal{L}$, $\mathcal{U}$)
```

n = 8

Range is initially infinite

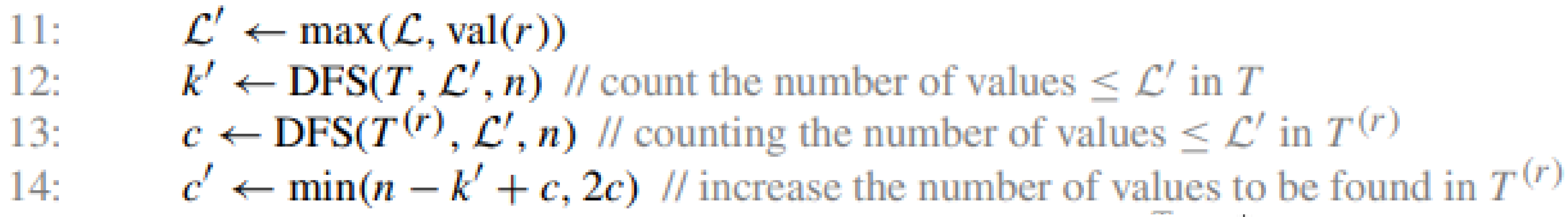
Randomly select a root within range



0:26 – 0:29

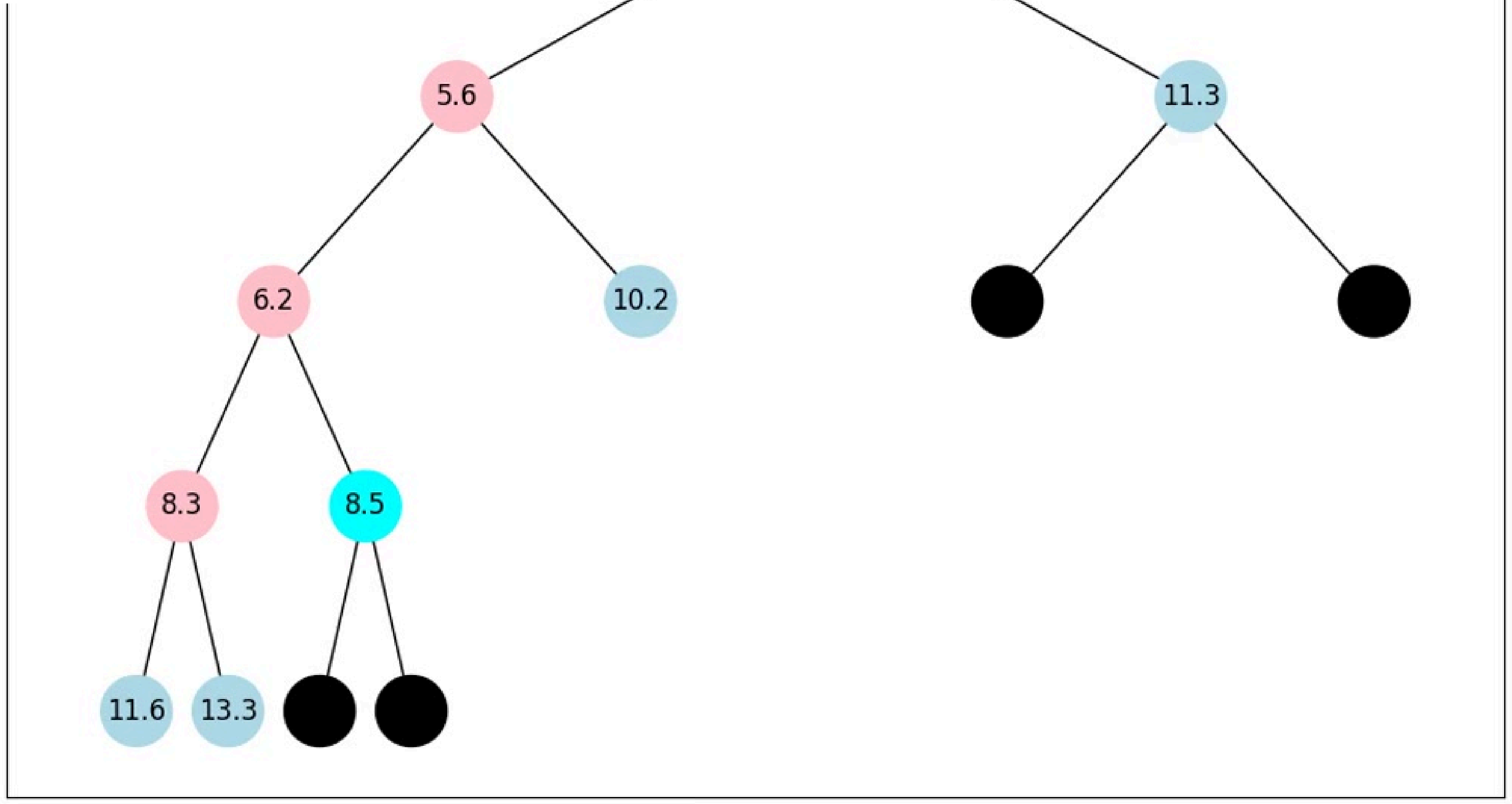
ALGORITHM

PSEUDOCODE



```
        $\mathcal{L}' \leftarrow \max(\mathcal{L}, \text{val}(r))$
        $k' \leftarrow \text{DFS}(T, \mathcal{L}', n)$ // count the number of values $\leq \mathcal{L}'$ in $T$
        $c \leftarrow \text{DFS}(T^{(r)}, \mathcal{L}', n)$ // counting the number of values $\leq \mathcal{L}'$ in $T^{(r)}$
        $c' \leftarrow \min(n - k' + c, 2c)$ // increase the number of values to be found in $T^{(r)}$
```

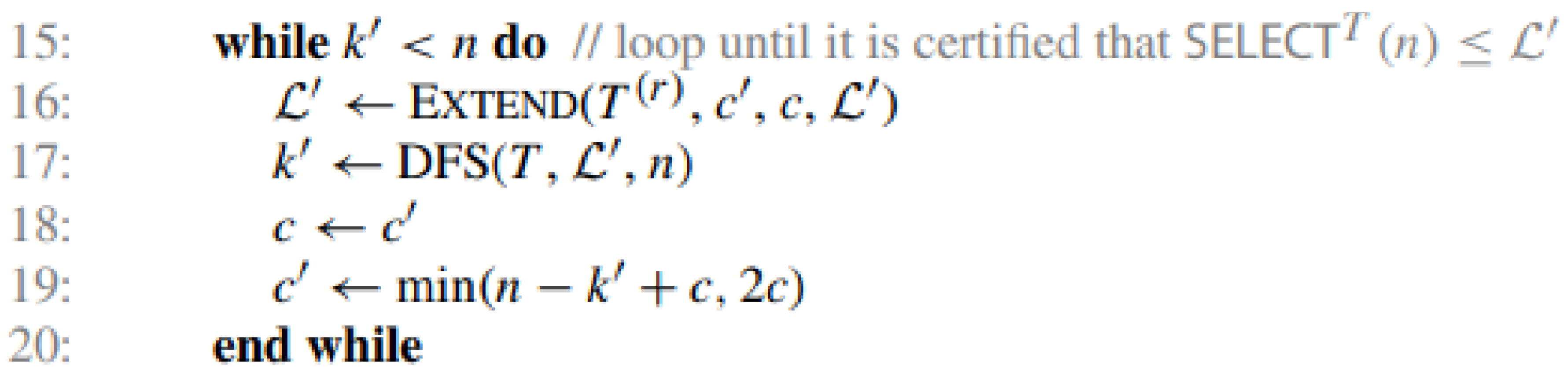
Root selected
Check how many values <
root in whole tree
And in subtree (none since
root of subtree)



0:29 - 0:32

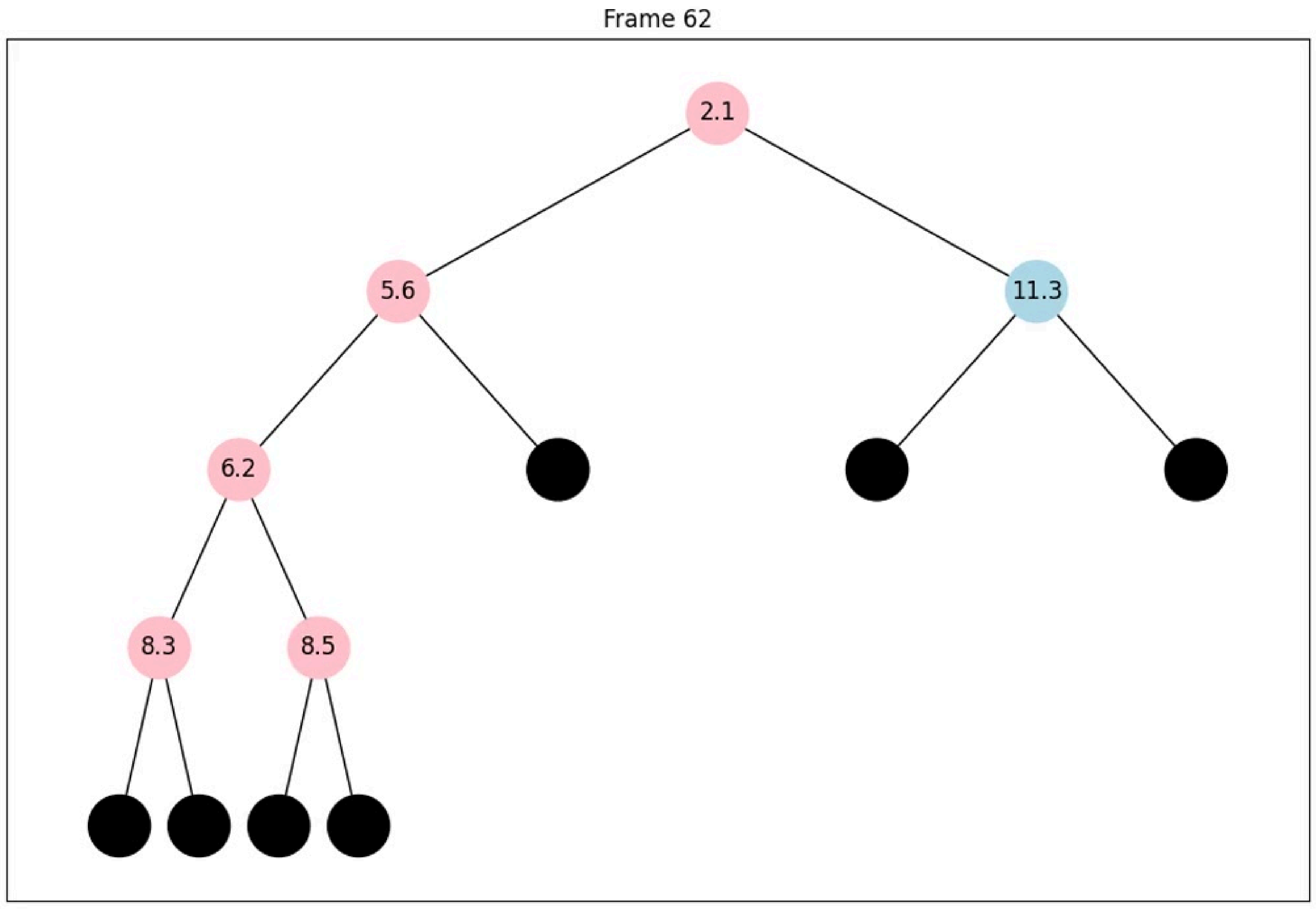
ALGORITHM

PSEUDOCODE



```
    while $k' < n$ do // loop until it is certified that SELECT$^T(n) \leq \mathcal{L}'$
        $\mathcal{L}' \leftarrow$ EXTEND($T^{(r)}, c', c, \mathcal{L}'$)
        $k' \leftarrow$ DFS($T, \mathcal{L}', n$)
        $c \leftarrow c'$
        $c' \leftarrow \min(n - k' + c, 2c)$
    end while
```

Expand subtree
c′ = 2
Finds 2 smallest node of subtree = 13.0
Check how many nodes of the whole tree fall below this value

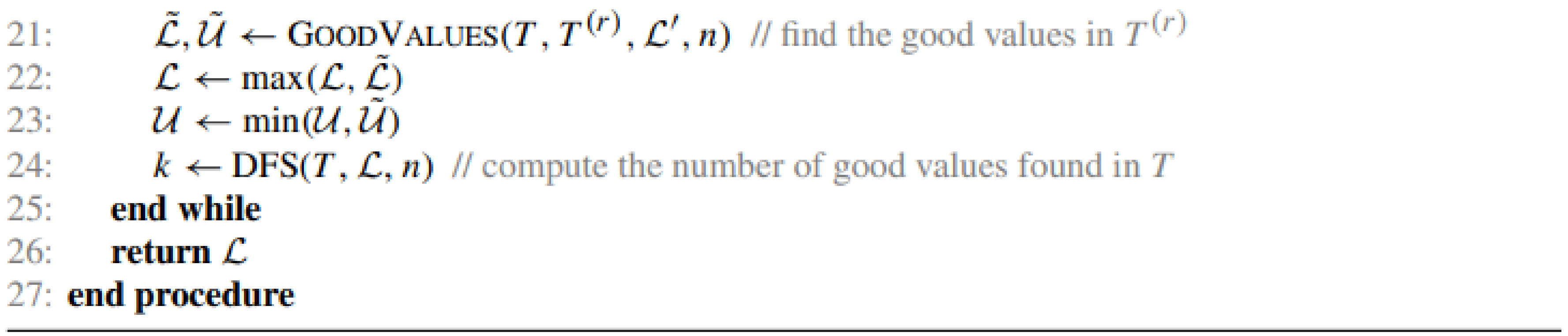


Frame 62

0:32 - 0:46

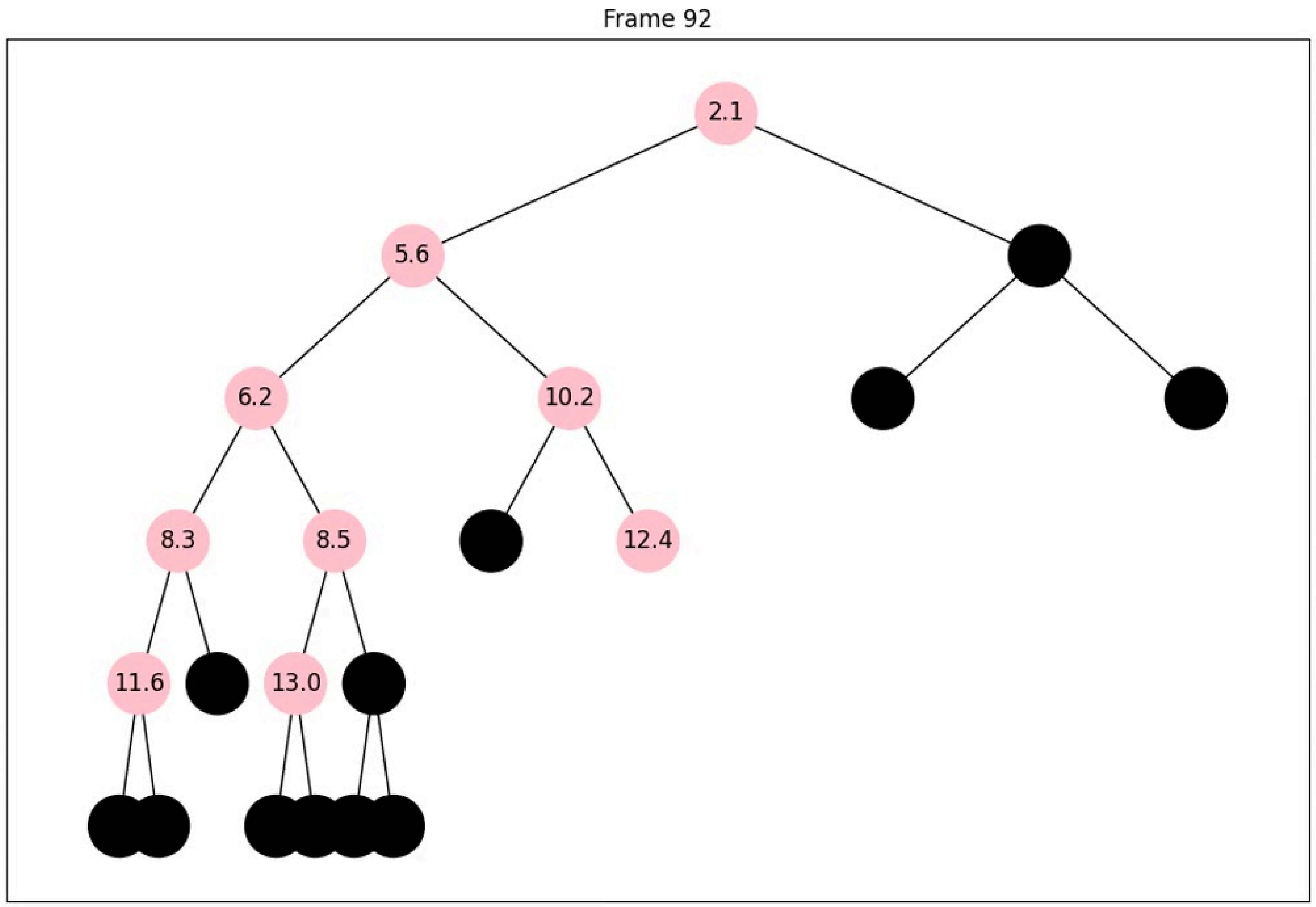
ALGORITHM

PSEUDOCODE



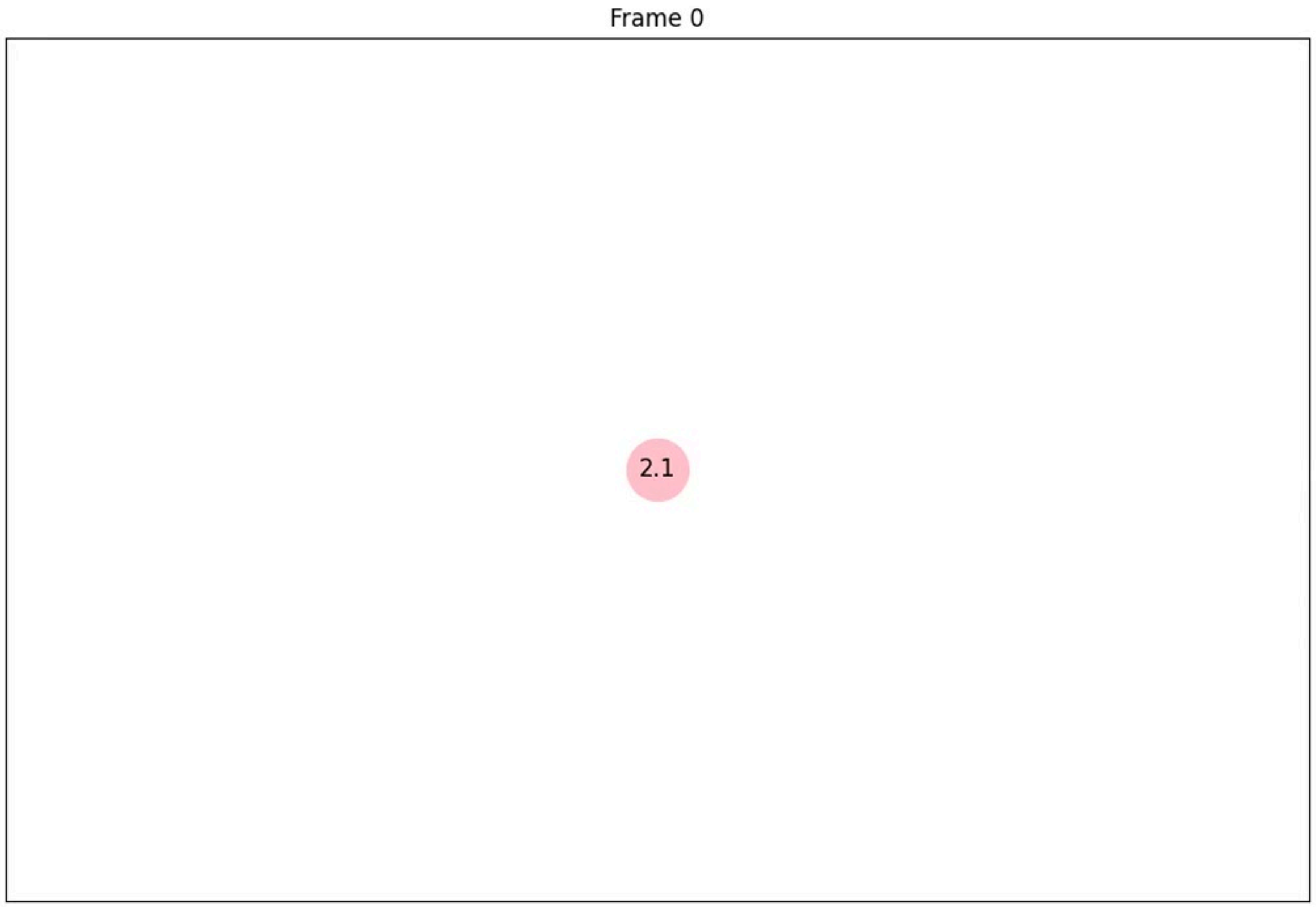
```
        $\tilde{\mathcal{L}}, \tilde{\mathcal{U}} \leftarrow$ GoodValues($T$, $T^{(r)}$, $\mathcal{L}'$, $n$)  // find the good values in $T^{(r)}$
        $\mathcal{L} \leftarrow \max(\mathcal{L}, \tilde{\mathcal{L}})$
        $\mathcal{U} \leftarrow \min(\mathcal{U}, \tilde{\mathcal{U}})$
        $k \leftarrow$ DFS($T$, $\mathcal{L}$, $n$)  // compute the number of good values found in $T$
    end while
    return $\mathcal{L}$
end procedure
```

We know that 8.5 (5 nodes) was too less
and 13 (9 nodes) was too much
We limit our range and continue



ALGORITHM

FULL RUN (for n = 8) / DEMO



0:00 – 1:08

Output = 11.6

IMPLEMENTATION & ENHANCEMENTS

- Heap nodes generated dynamically to create a virtually infinite heap. 3 types of heaps:
  - **FirstN:** Nodes are numbered 1, 2, 3, ...
  - **RandGen:** Nodes have random values (obeying heap property)
  - **KnapSack:** Nodes represent relaxed LP problems
- Added support for heaps with duplicate values *
- We had to implement subroutines that only had a high-level description and no pseudocode *
- One subroutine's description seemed to have a slight error, which we corrected (GoodValues) *
- We used the random heap exploration algorithm for branch and bound optimization on 0-1 Knapsack *
- Visualisation created *
- Empirical Testing *

* = Enhancement

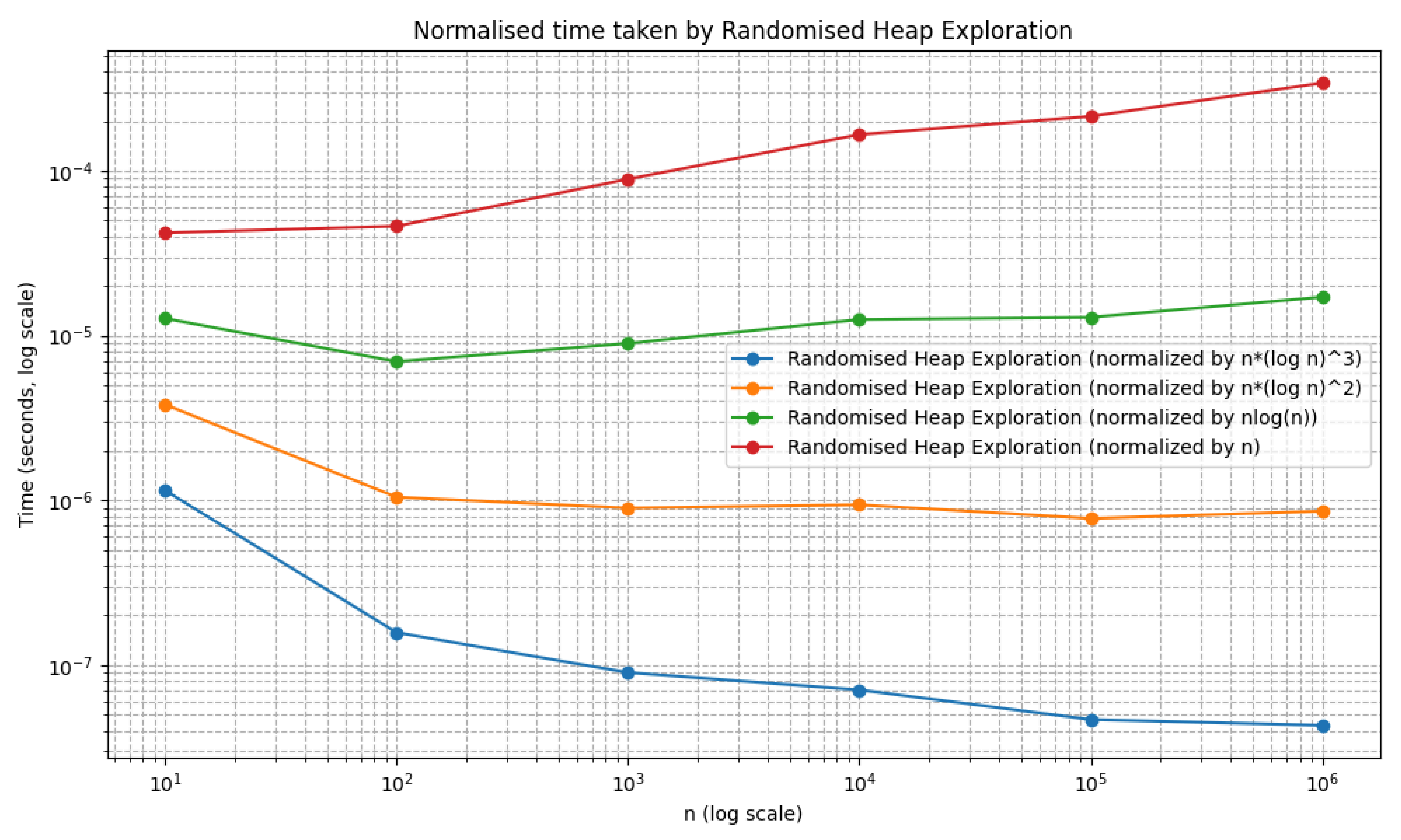
EVALUATION

Accuracy:

- Evaluated against best first (50 random heaps with n in [100, 50000)
- Evaluated on firstN for large values → expected output was n which matched
- Evaluated to solve Knapsack on 2 problem instances → Matched optimal value and solution obtained by ILP solver in PuLP library

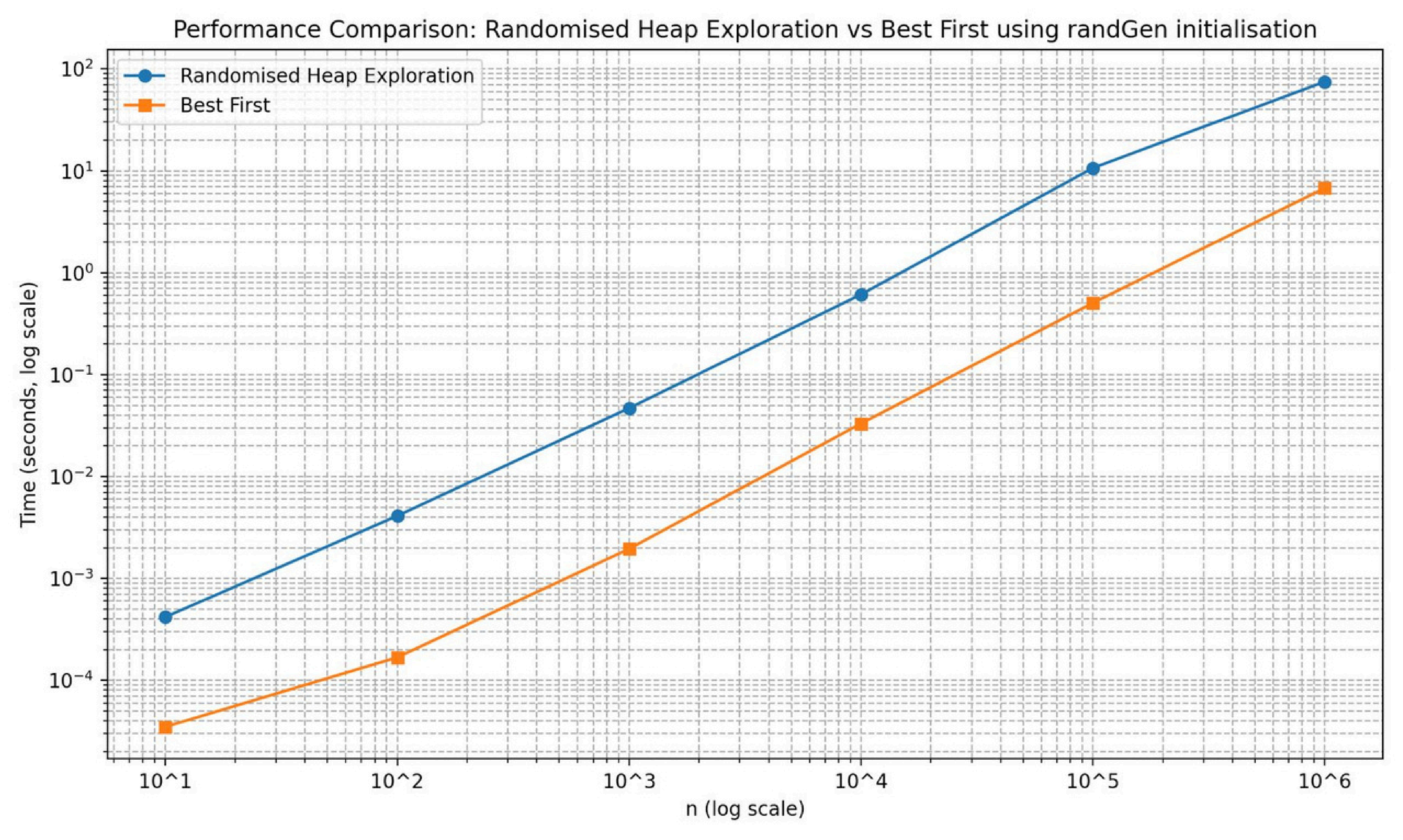
EVALUATION

Runtime – Empirical complexity analysis – O(n log²n)



EVALUATION

Runtime – Comparison against Best First



FUTURE DIRECTIONS

- Memory usage analysis
- Memory efficient DFS
- Support for heaps with heavily duplicated values
- Improved visualisation showing the current range and other parameters

KEY REFERENCES

1. Sander Borst, Daniel Dadush, Sophie Huiberts, and Dan▯ish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. Mathematical Programming, 210(1):75–96, March 2025.
2. Richard M Karp, Michael E Saks, and Avi Wigderson. On a search problem related to branch-and-bound procedures. In FOCS, pages 19–28, 1986
3. Tom S. (n.d.). The art of linear programming [Video]. YouTube. https://www.youtube.com/watch?v=E72DWgKP_1Y

THANK YOU

Any Questions?