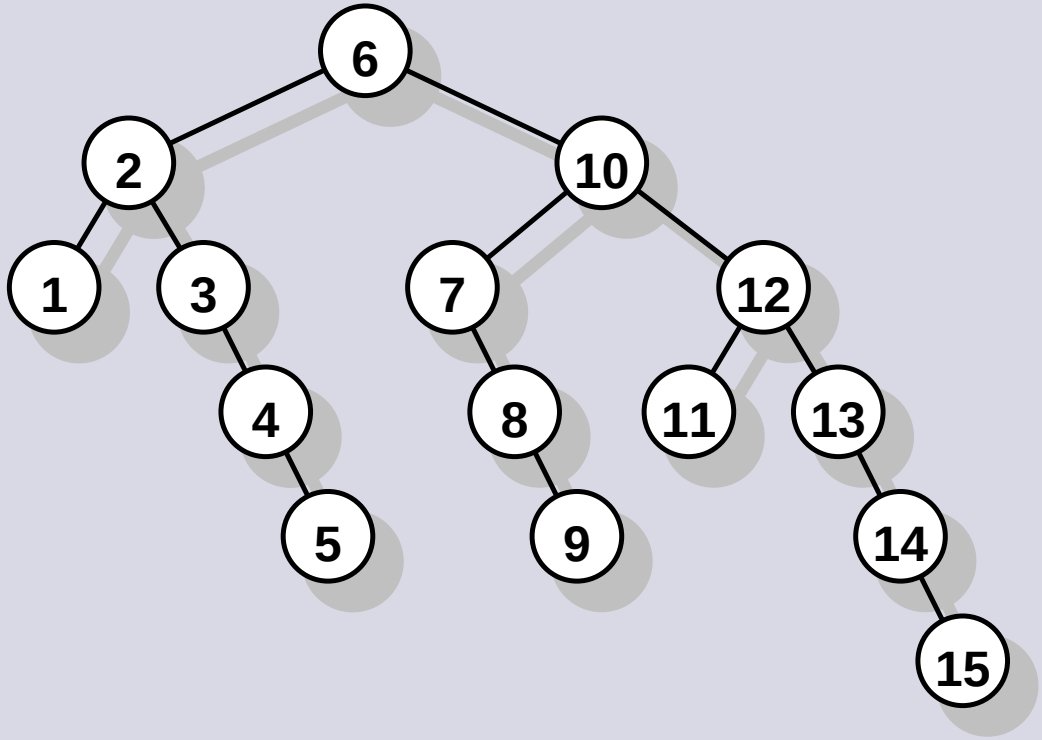
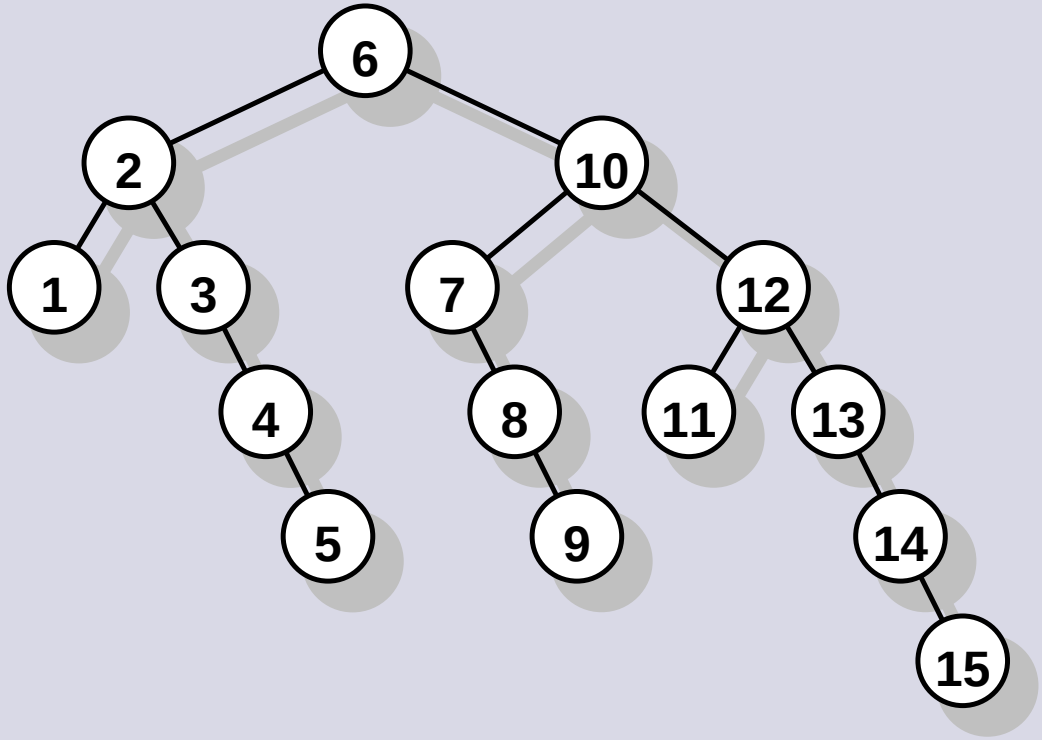
Skewed Binary Search Trees



Gerth Stølting Brodal
University of Aarhus

Joint work with Gabriel Moruz

Skewed Binary Search Trees



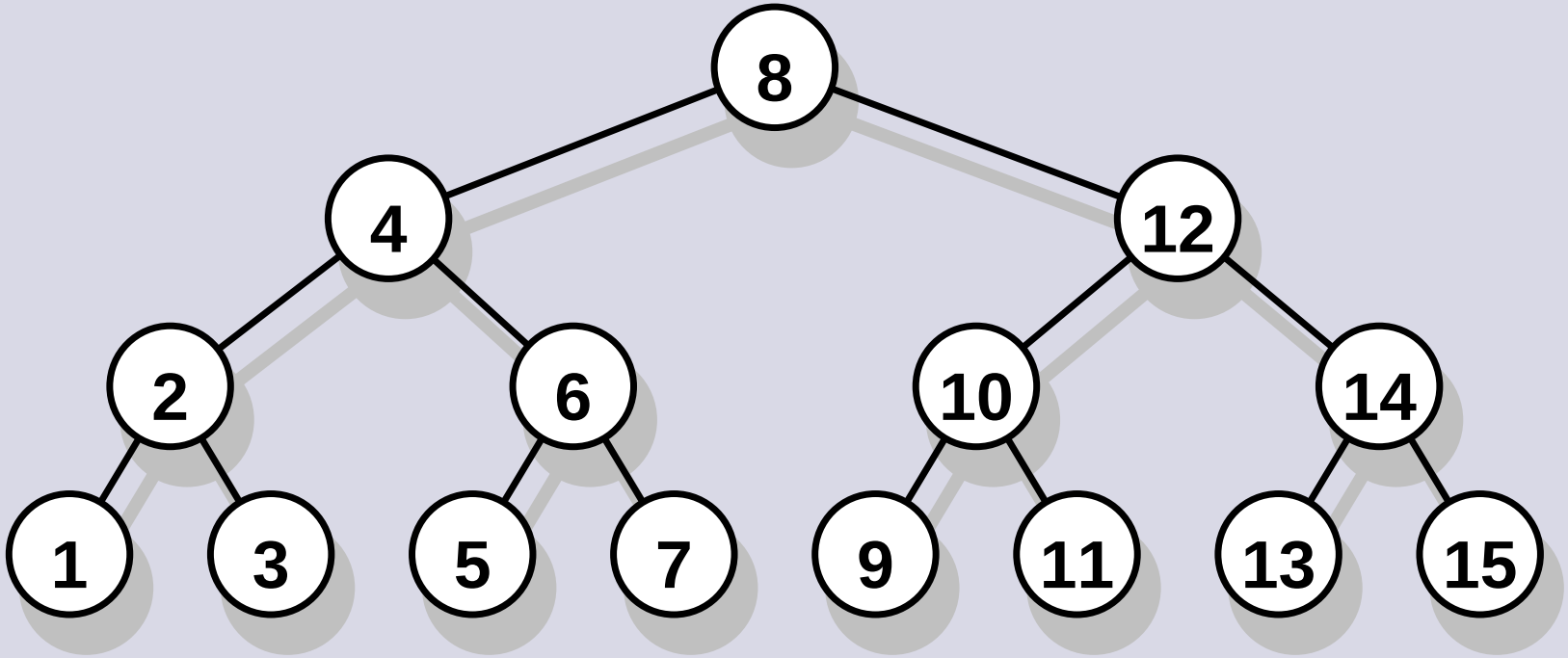
Gerth Stølting Brodal
University of Aarhus

Joint work with Gabriel Moruz

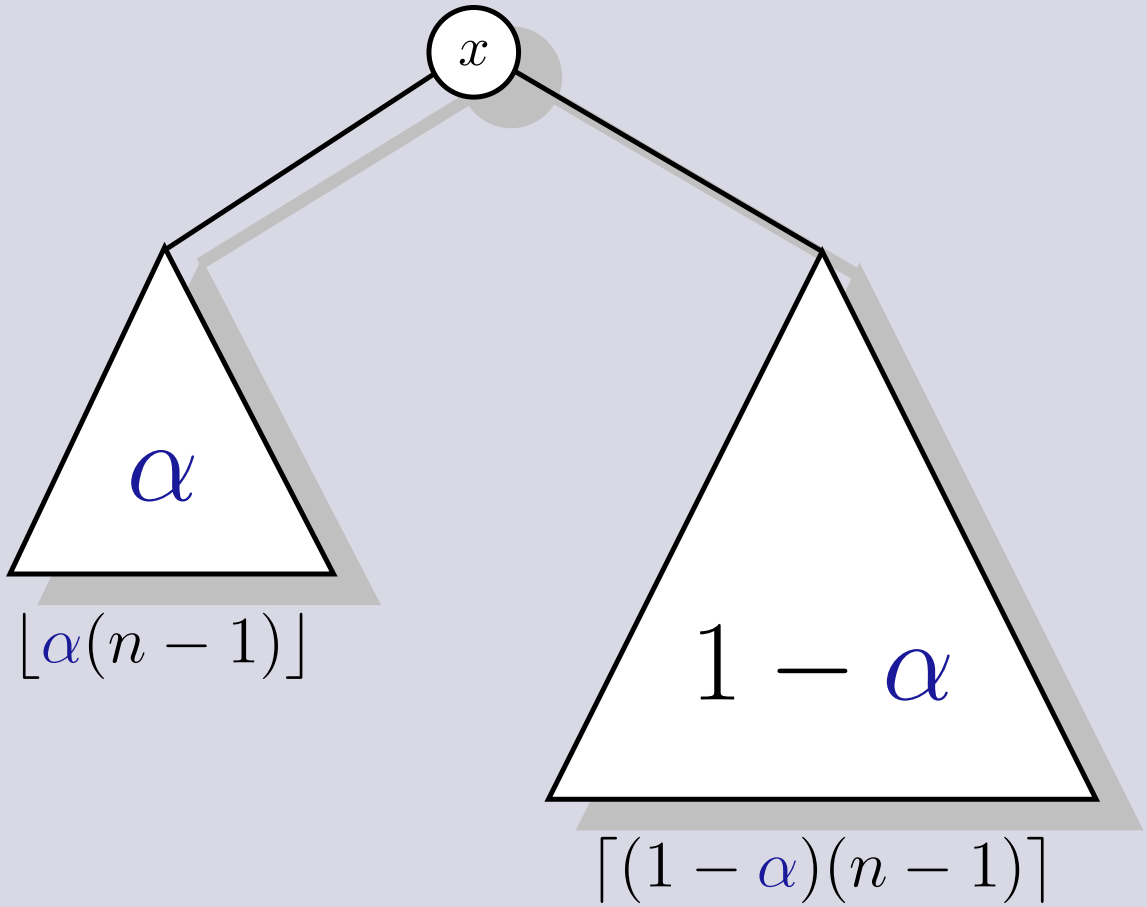
in progress

Dagstuhl seminar on Data Structures, February 26-March 3, 2006.

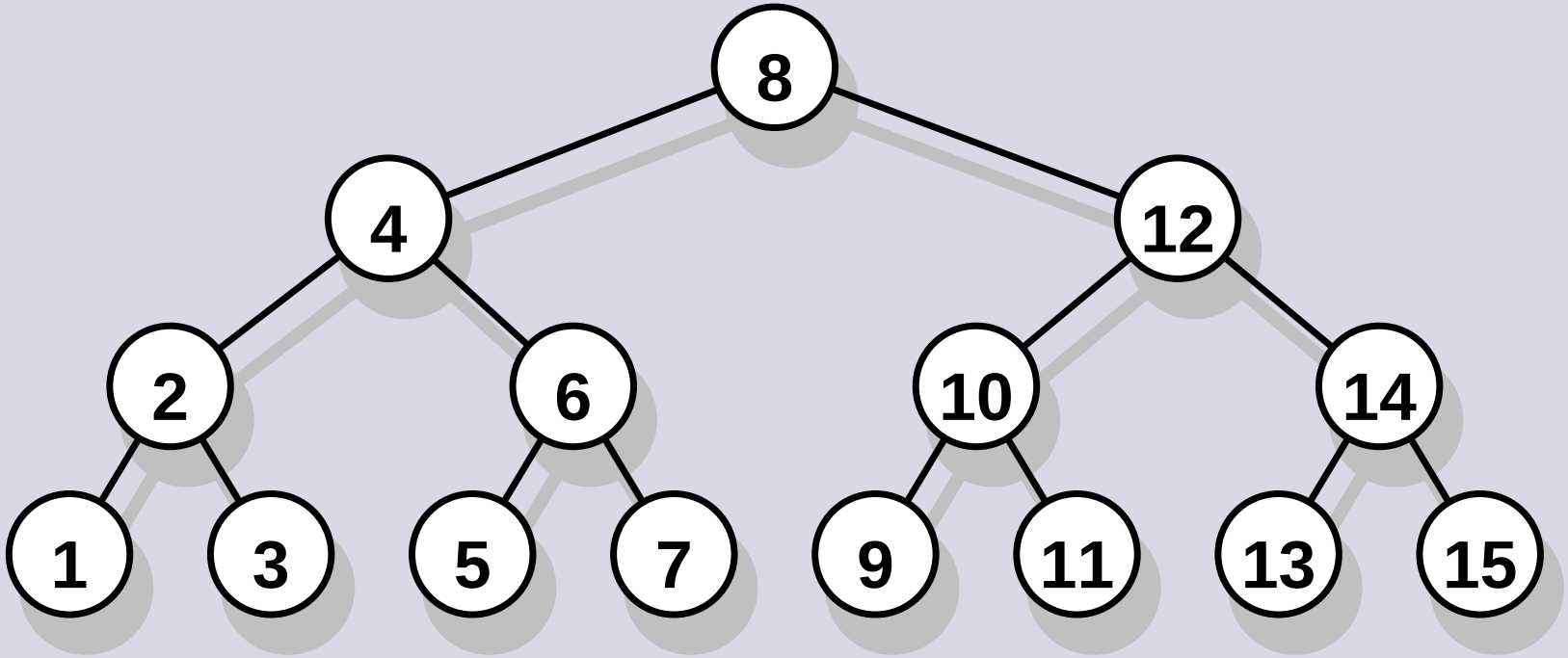
Perfectly Balanced Search Trees



Skewed Binary Search Trees

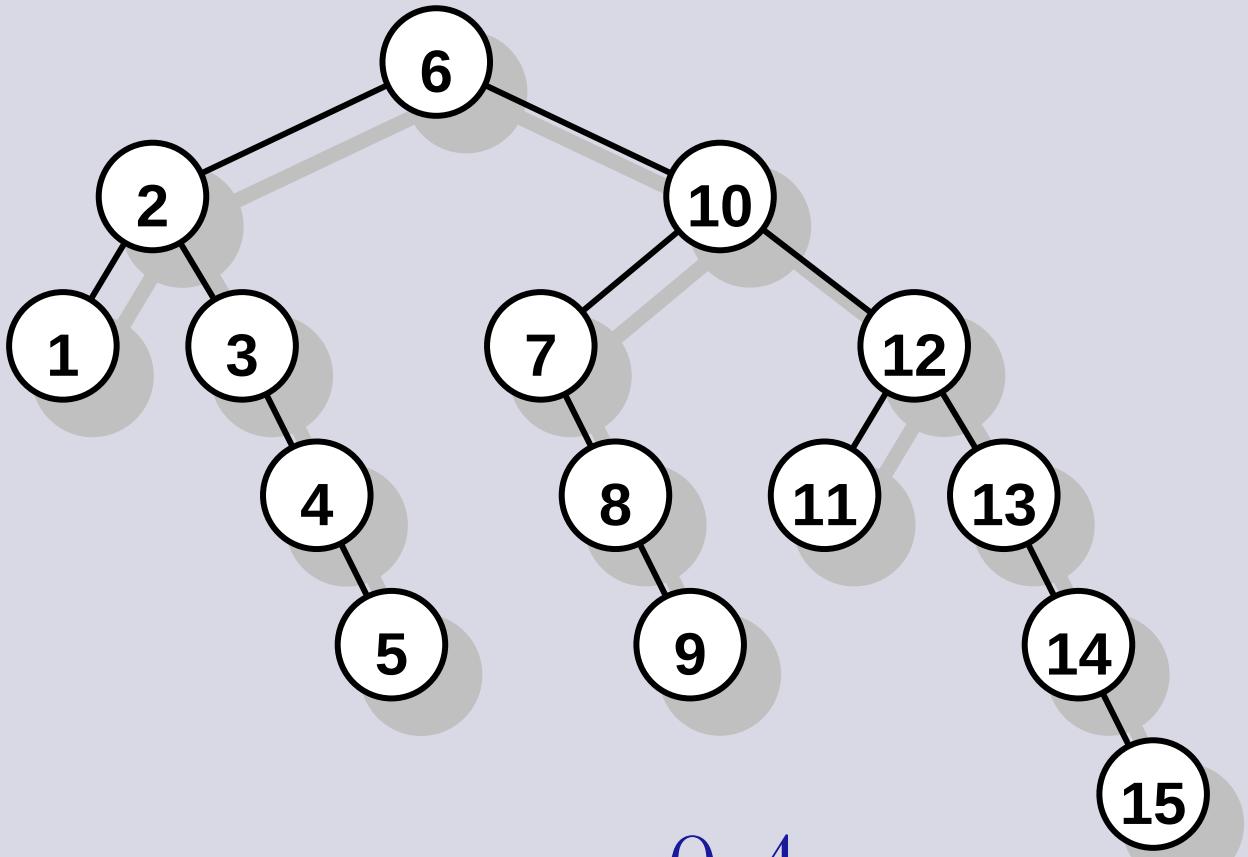


Skewed Binary Search Trees



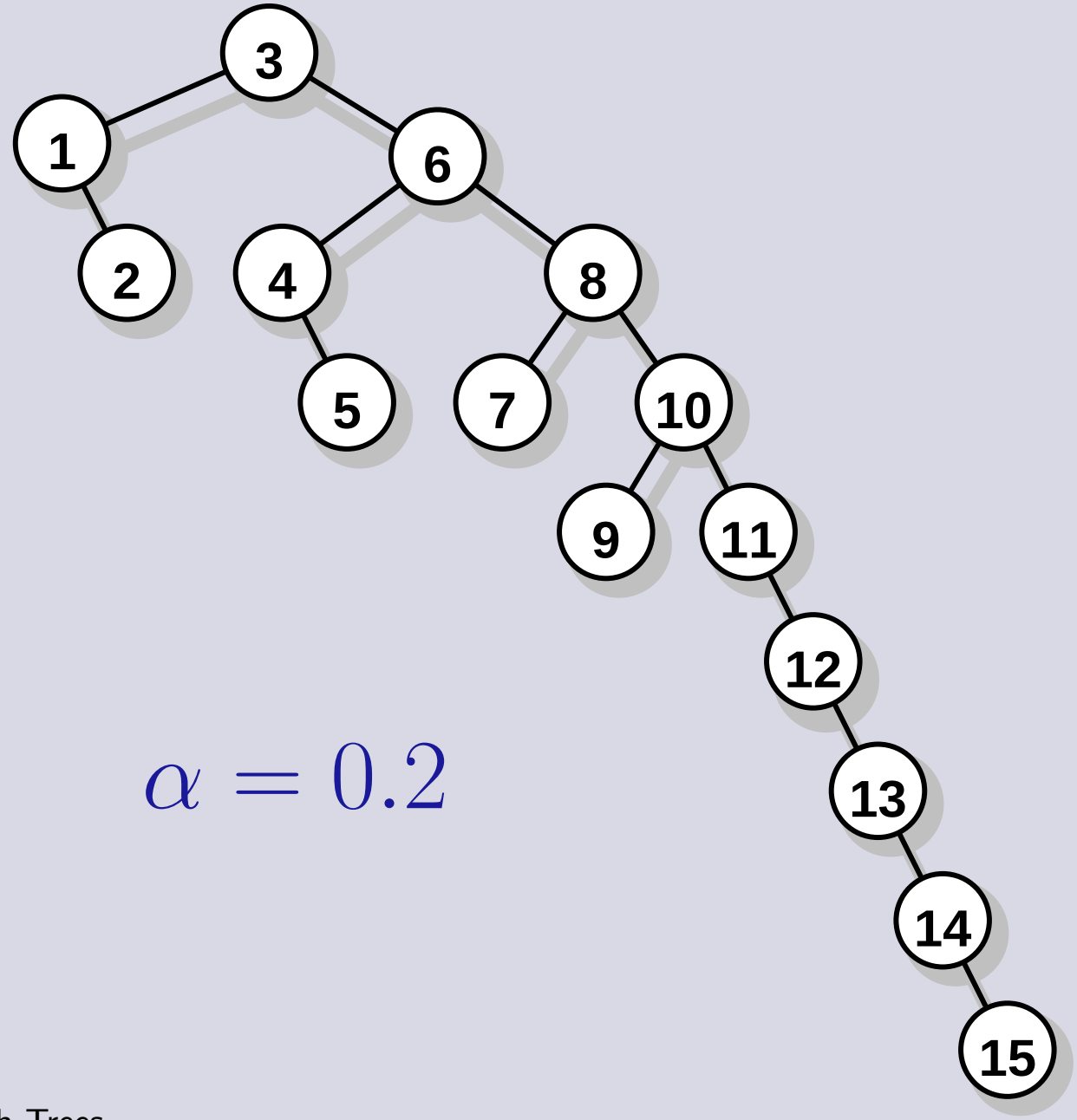
$$\alpha = 0.5$$

Skewed Binary Search Trees



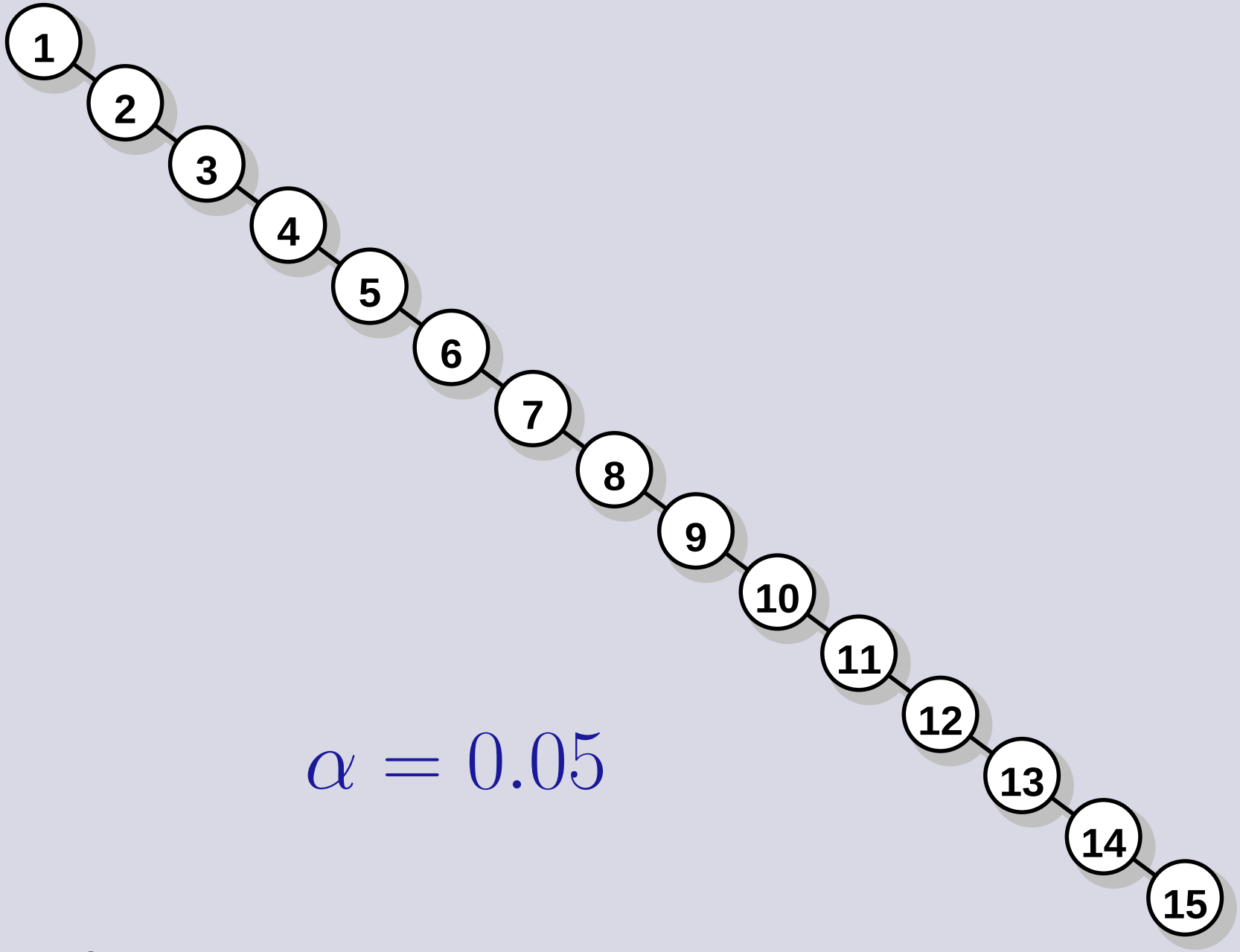
$$\alpha = 0.4$$

Skewed Binary Search Trees



$\alpha = 0.2$

Skewed Binary Search Trees



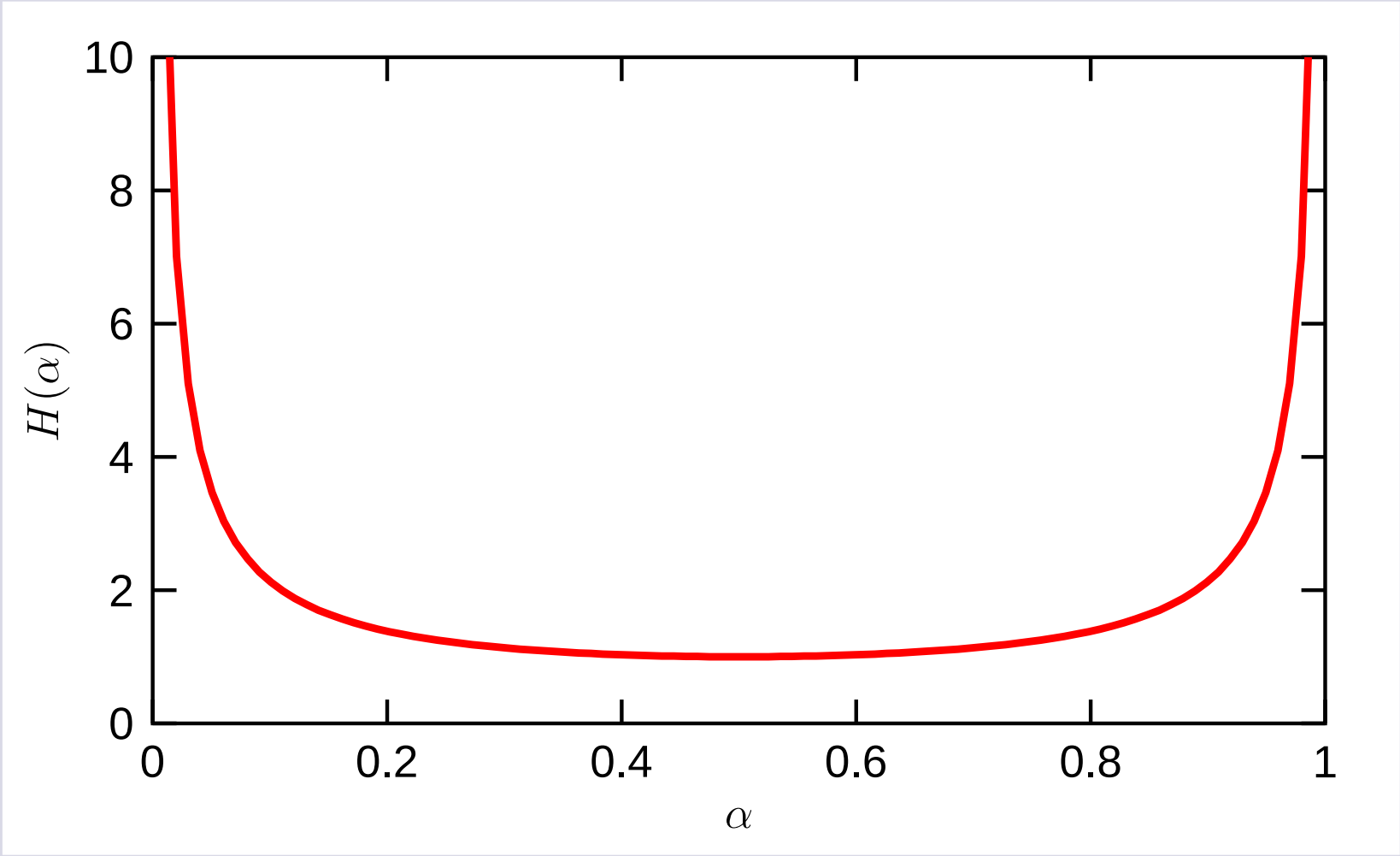
$$\alpha = 0.05$$

Skewed Binary Search Trees
— Average Node Depth

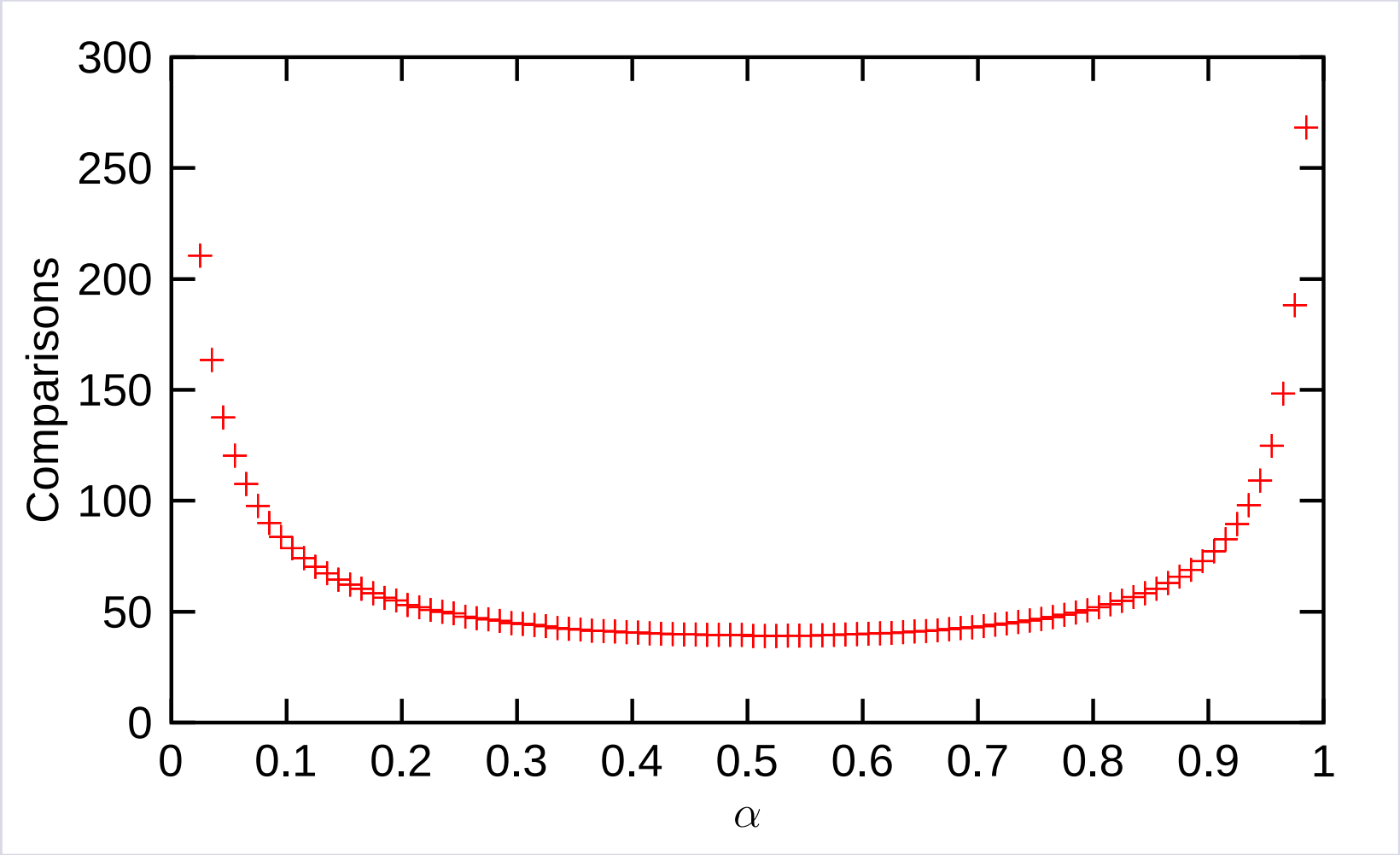
$$\leq \underbrace{\frac{1}{-\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)}}_{H(\alpha)} \cdot \frac{n+1}{n} \cdot \log_2(n+1) - 2$$

Nievergelt and E. M. Reingold, 1972

$H(\alpha)$

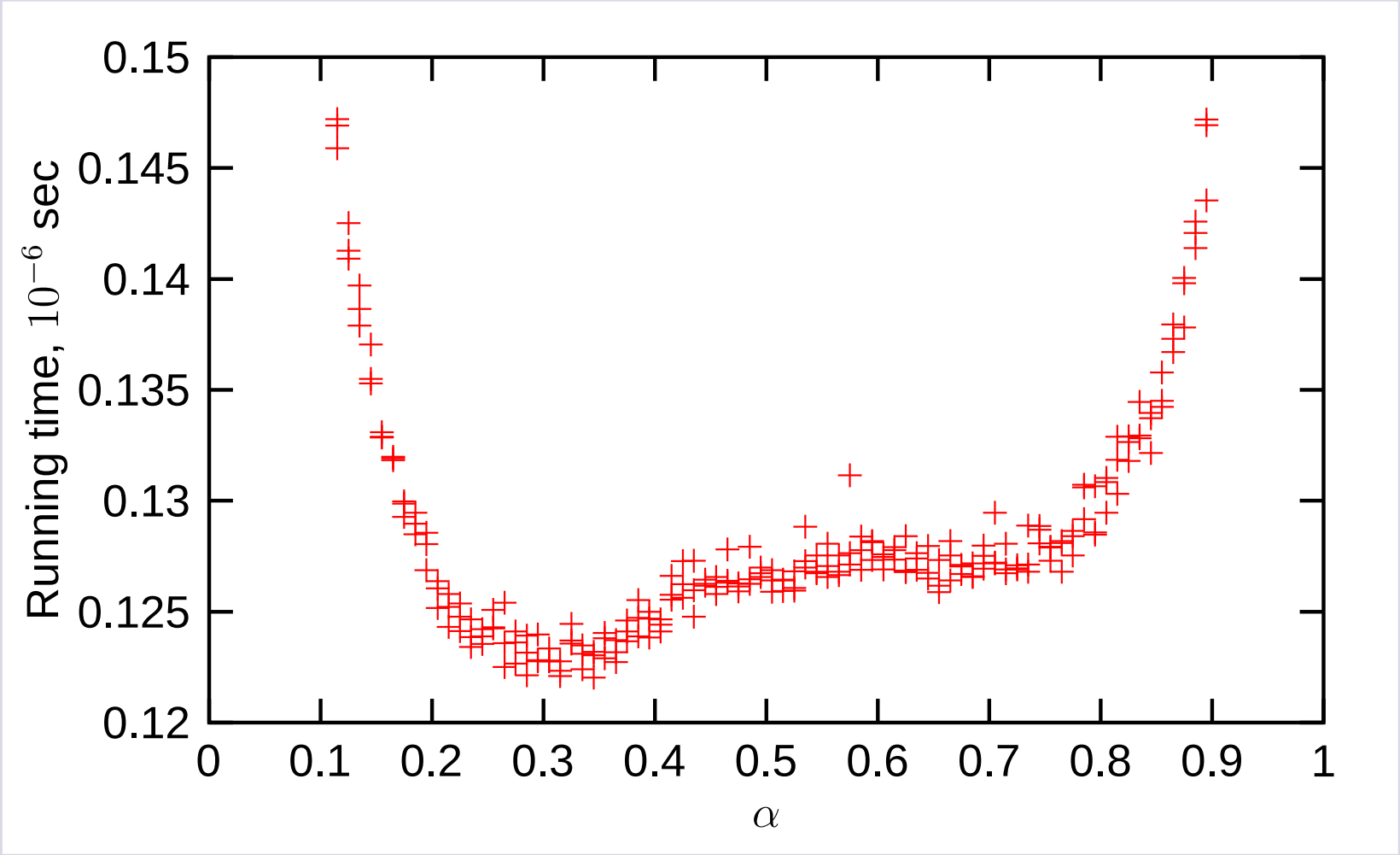


Comparisons



$n = 20.000$

Running Time



Best running time achieved for $\alpha \approx 0.3$!?

Conclusion

Skewed binary search trees

beat

Perfectly balanced binary search trees !

Conclusion

Skewed binary search trees

beat

Perfectly balanced binary search trees !

Why ?

Why ?

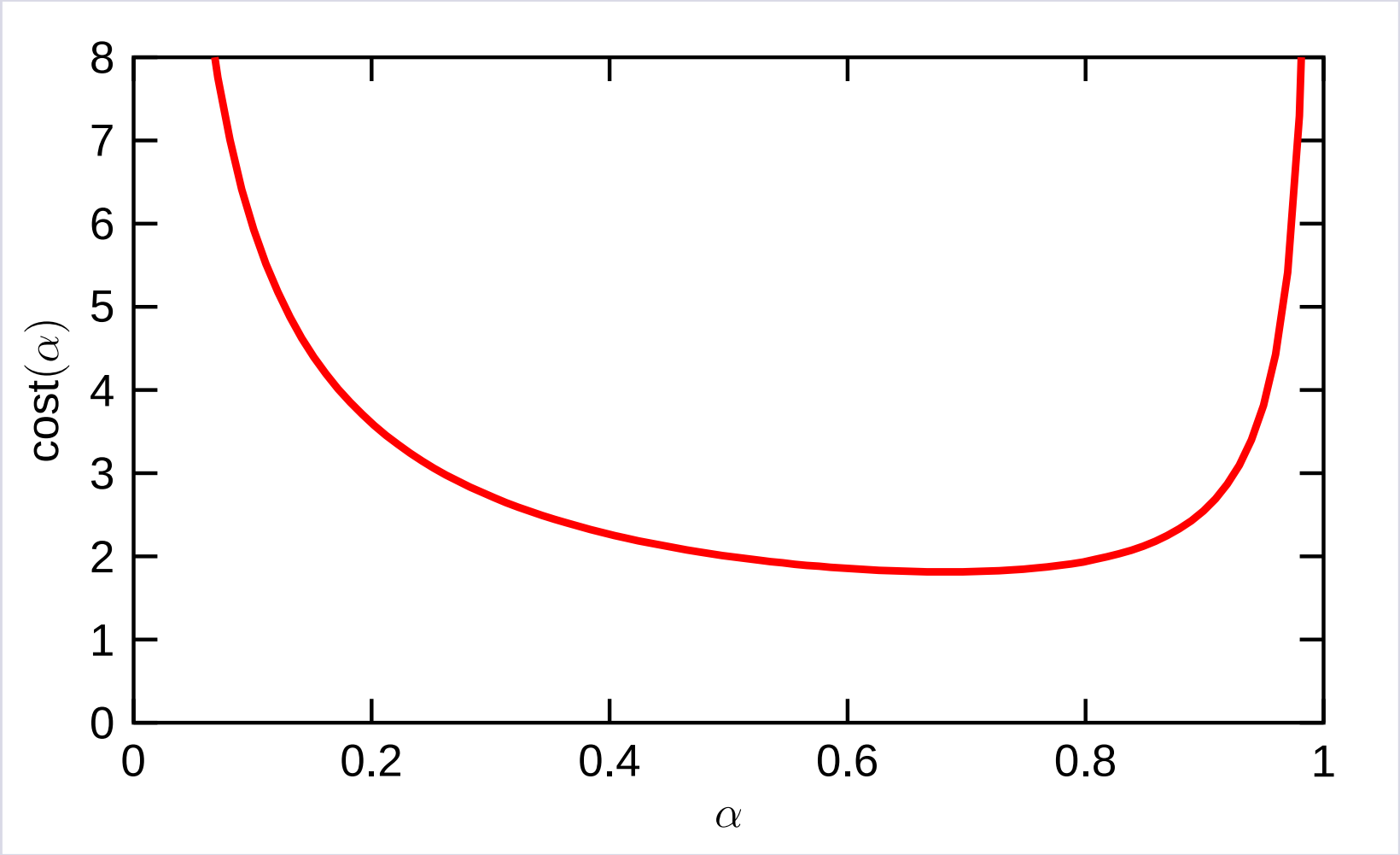
The costs going left and right are different !

Possible reasons

- Number of instructions
- Branch mispredictions
- Cache faults (what is a good memory layout?)
- ...

Expected Cost

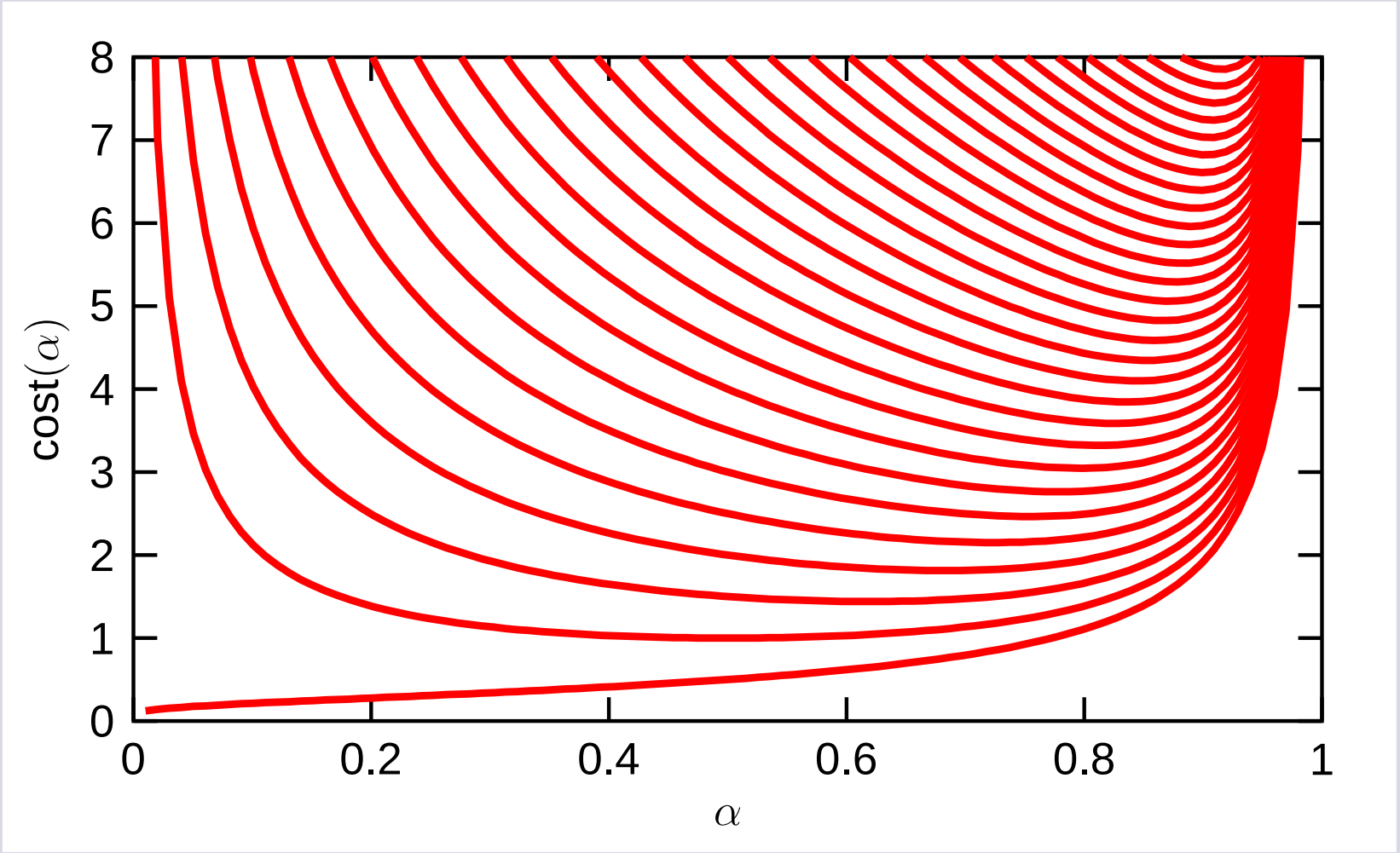
$$\text{cost}(\alpha) = (\alpha \cdot \{\text{left cost}\} + (1 - \alpha) \cdot \{\text{right cost}\}) \cdot H(\alpha)$$



left cost = 1 and right cost = 3

Expected Cost

$$\text{cost}(\alpha) = (\alpha \cdot \{\text{left cost}\} + (1 - \alpha) \cdot \{\text{right cost}\}) \cdot H(\alpha)$$

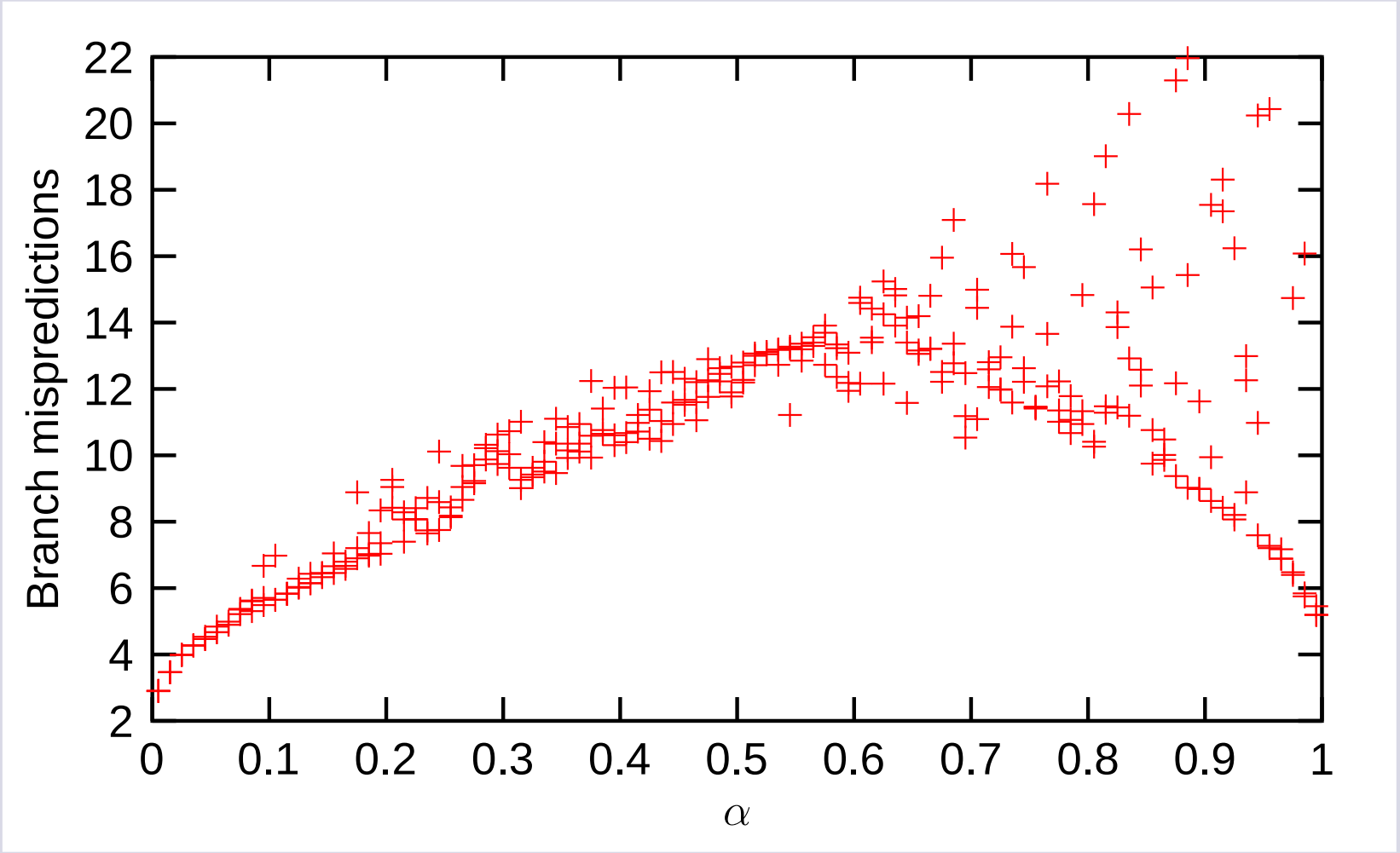


left cost = 1 and right cost = 0 .. 28

Search Code

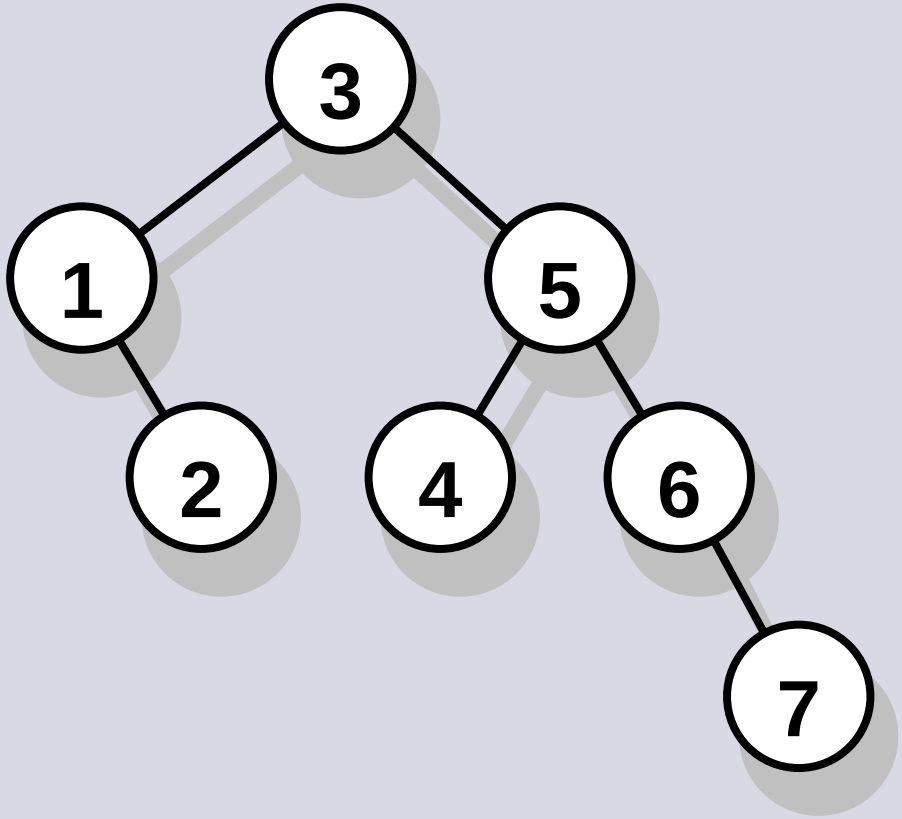
```
int search(int root, int key)
{
  if (root==NULLV)
    return NULLV;
  if (key==t[root].key)
    return root;
  if(key>t[root].key)
    return search(t[root].right, key);
  else
    return search(t[root].left, key);
}
```

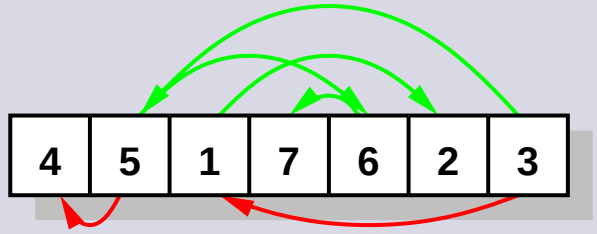
Branch Mispredictions



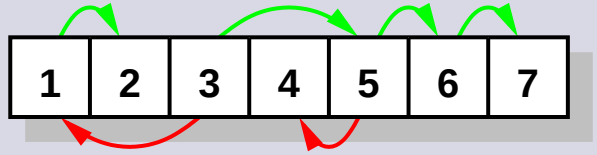
Static branch prediction: left cost = 1 and right cost = 0

Simple Layouts

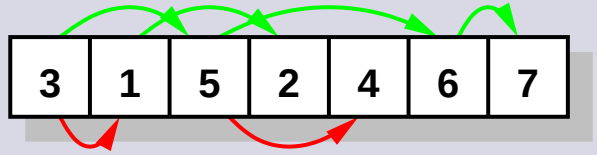




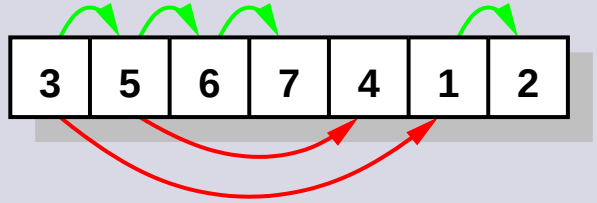
Random



Inorder

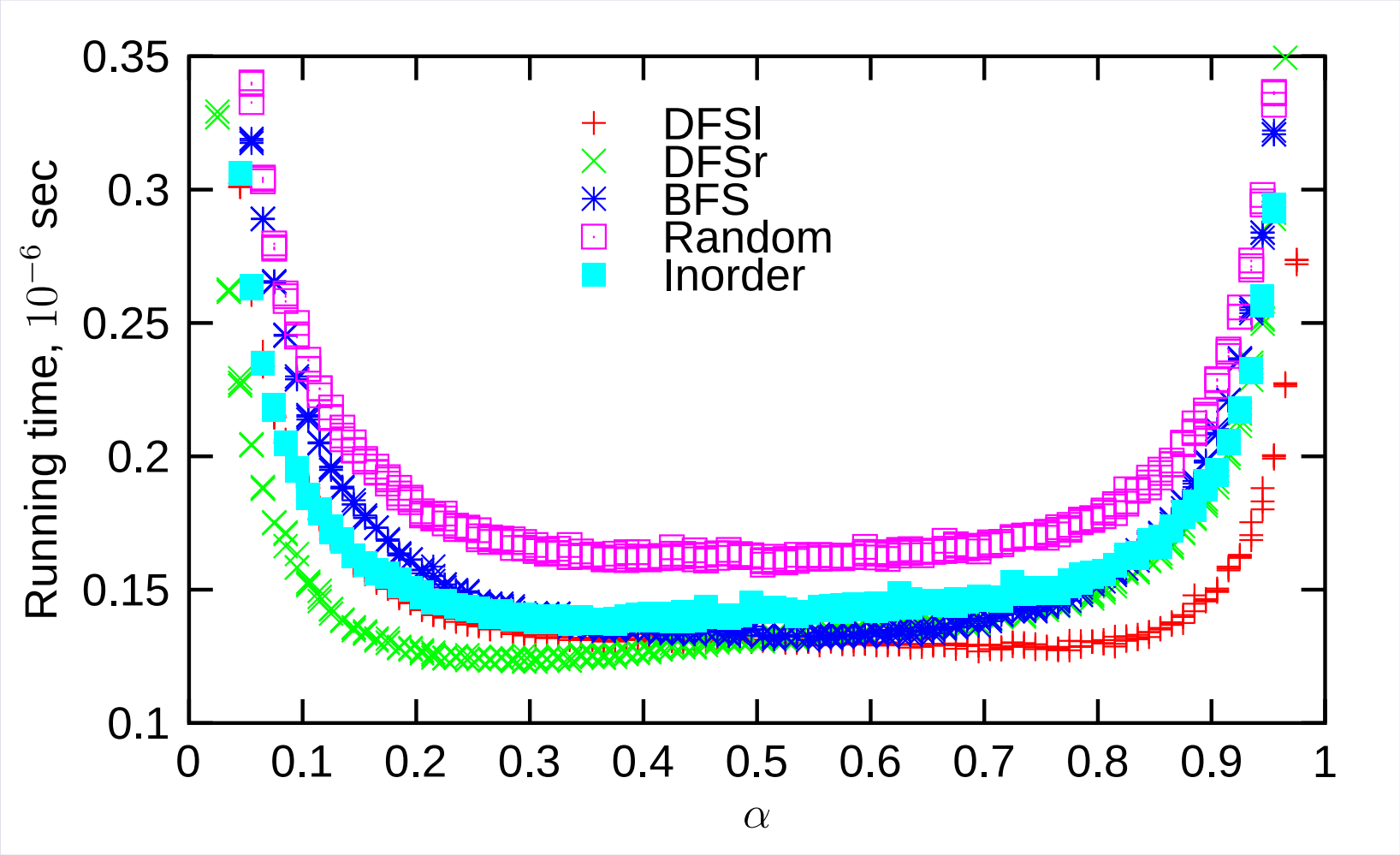


BFS



DFS

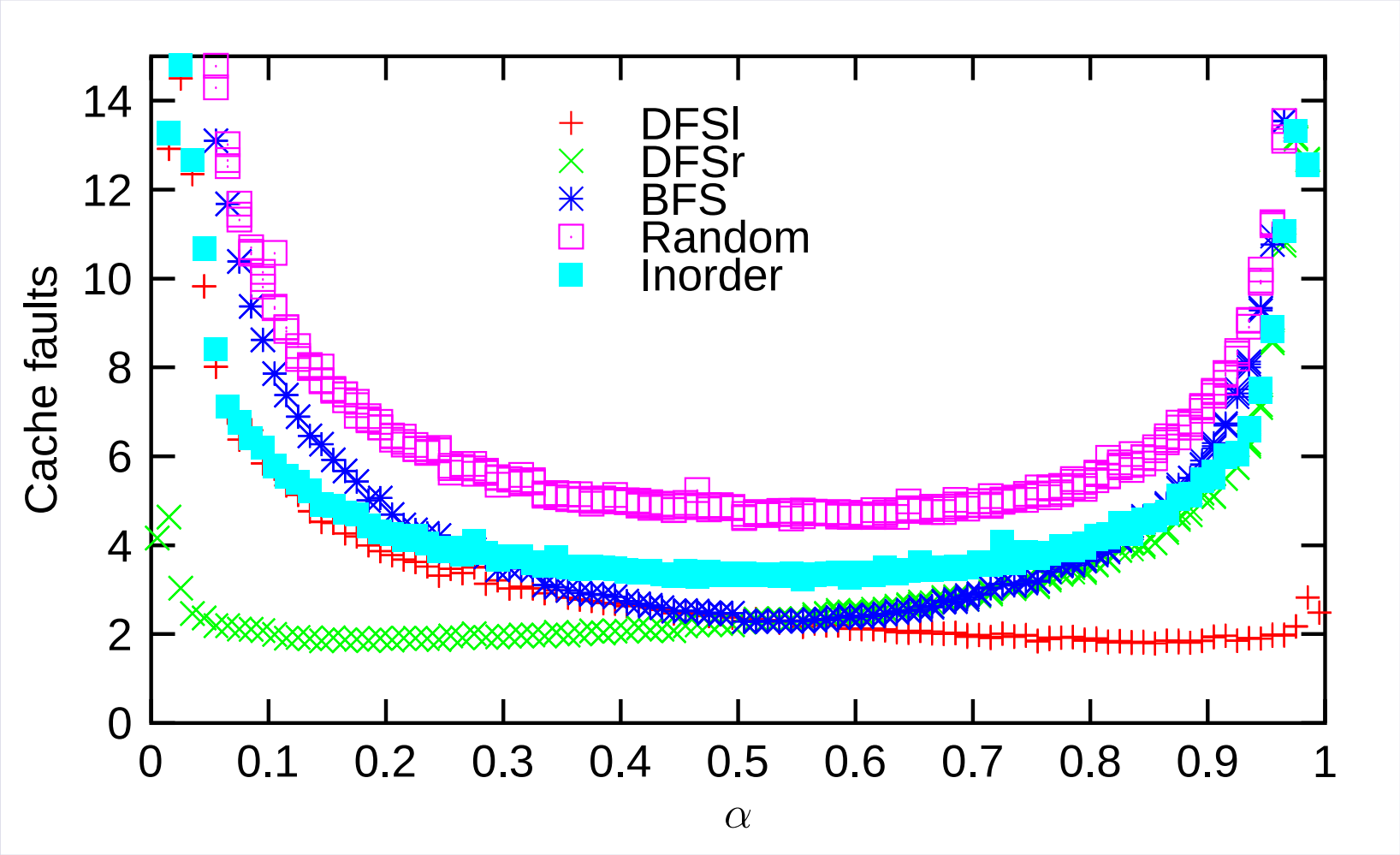
Running Time for Simple Layouts



DFS < Inorder < BFS < Random

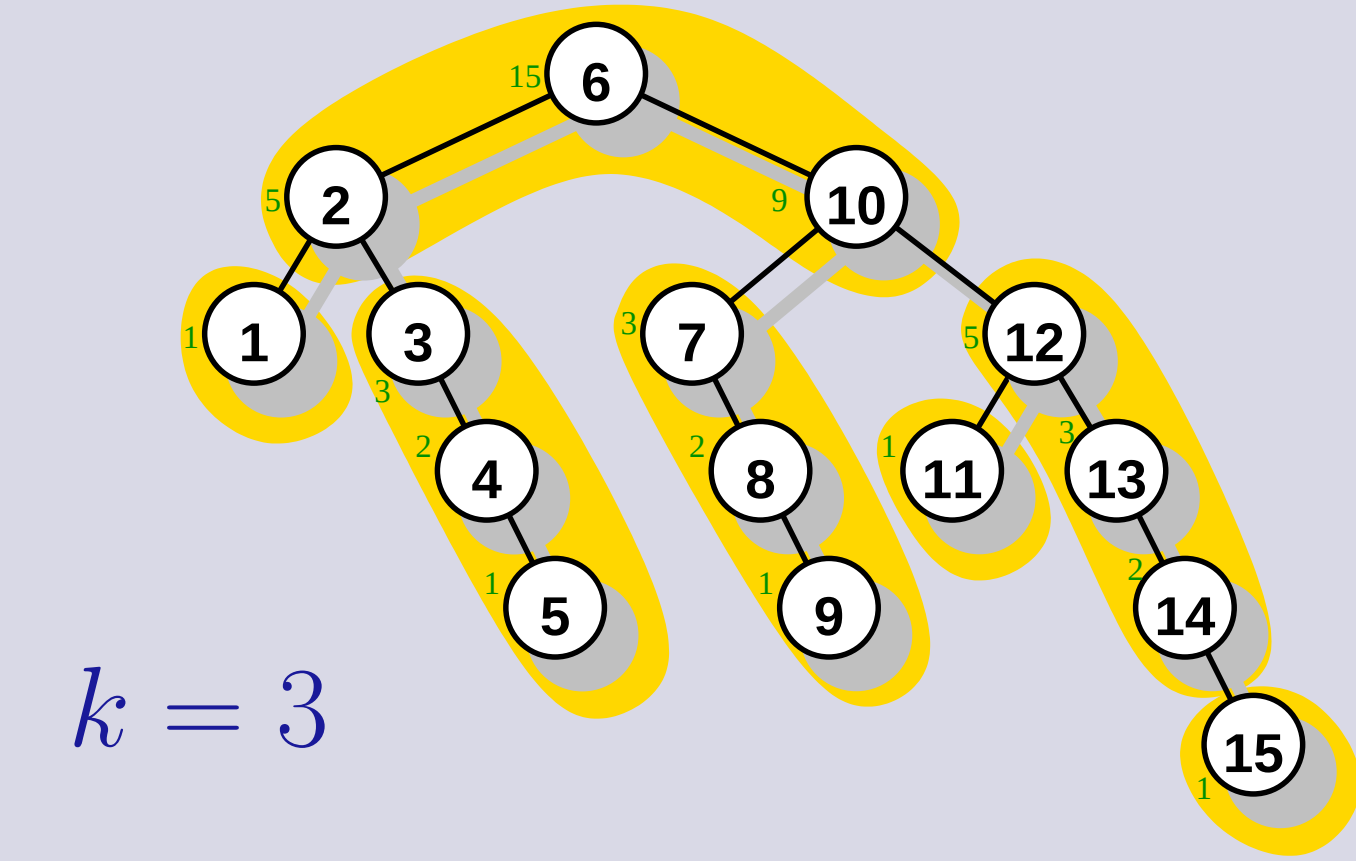
DFS achieves the best performance for $\alpha \approx 0.2$!

Cache Faults for Simple Layouts

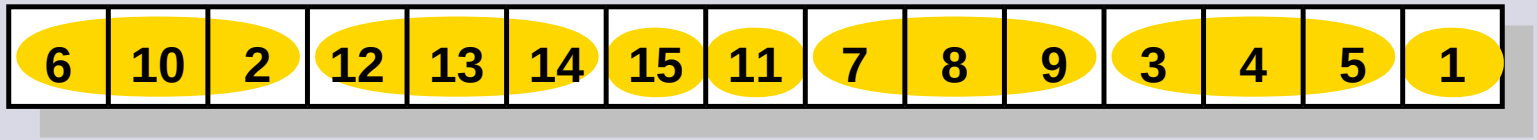


DFS $\approx$ expected left cost = 1 and right cost = $\dfrac{1}{\text{cache-line size}}$

Blocked Layouts — pqDFSk

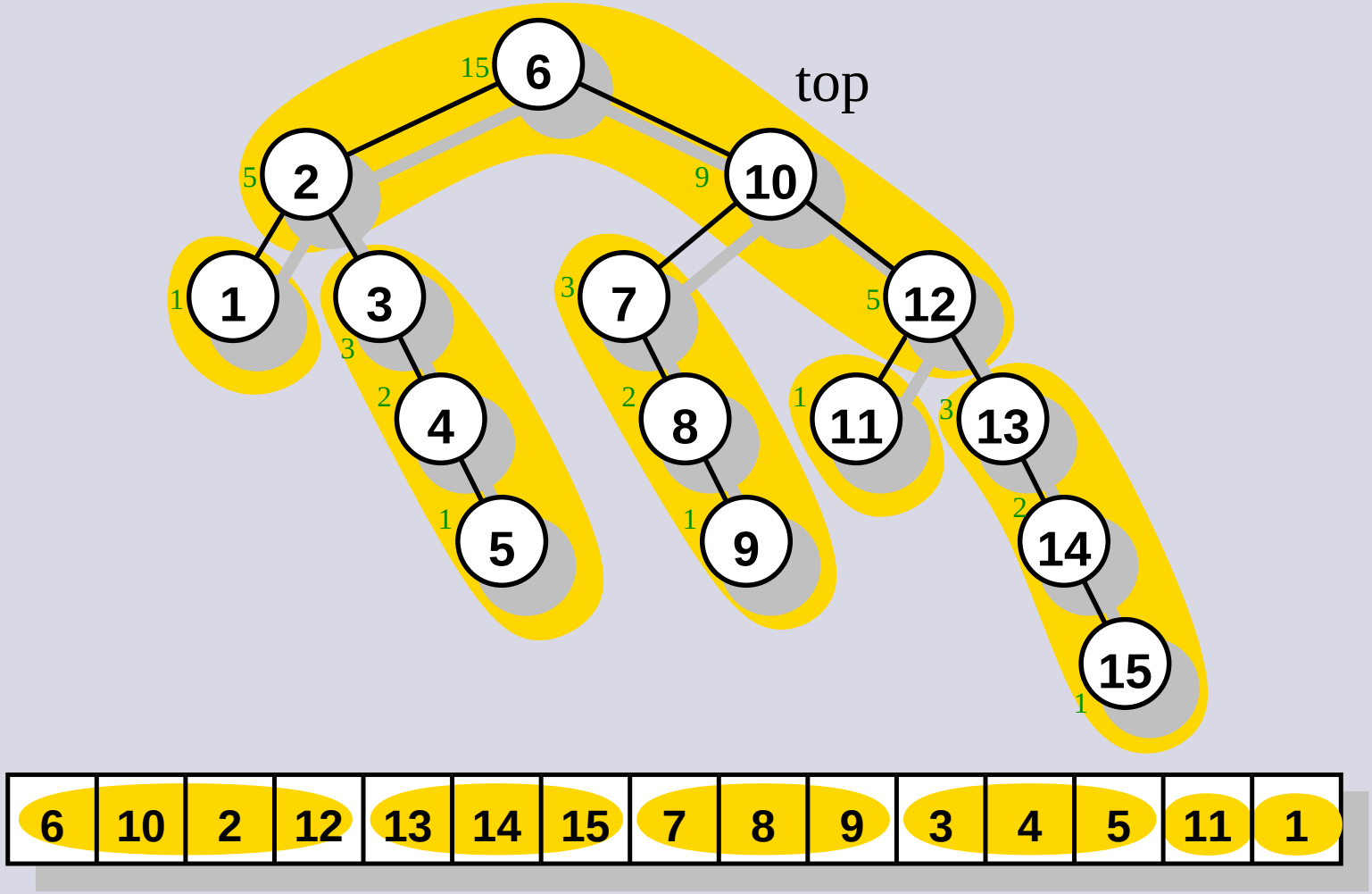


$k = 3$



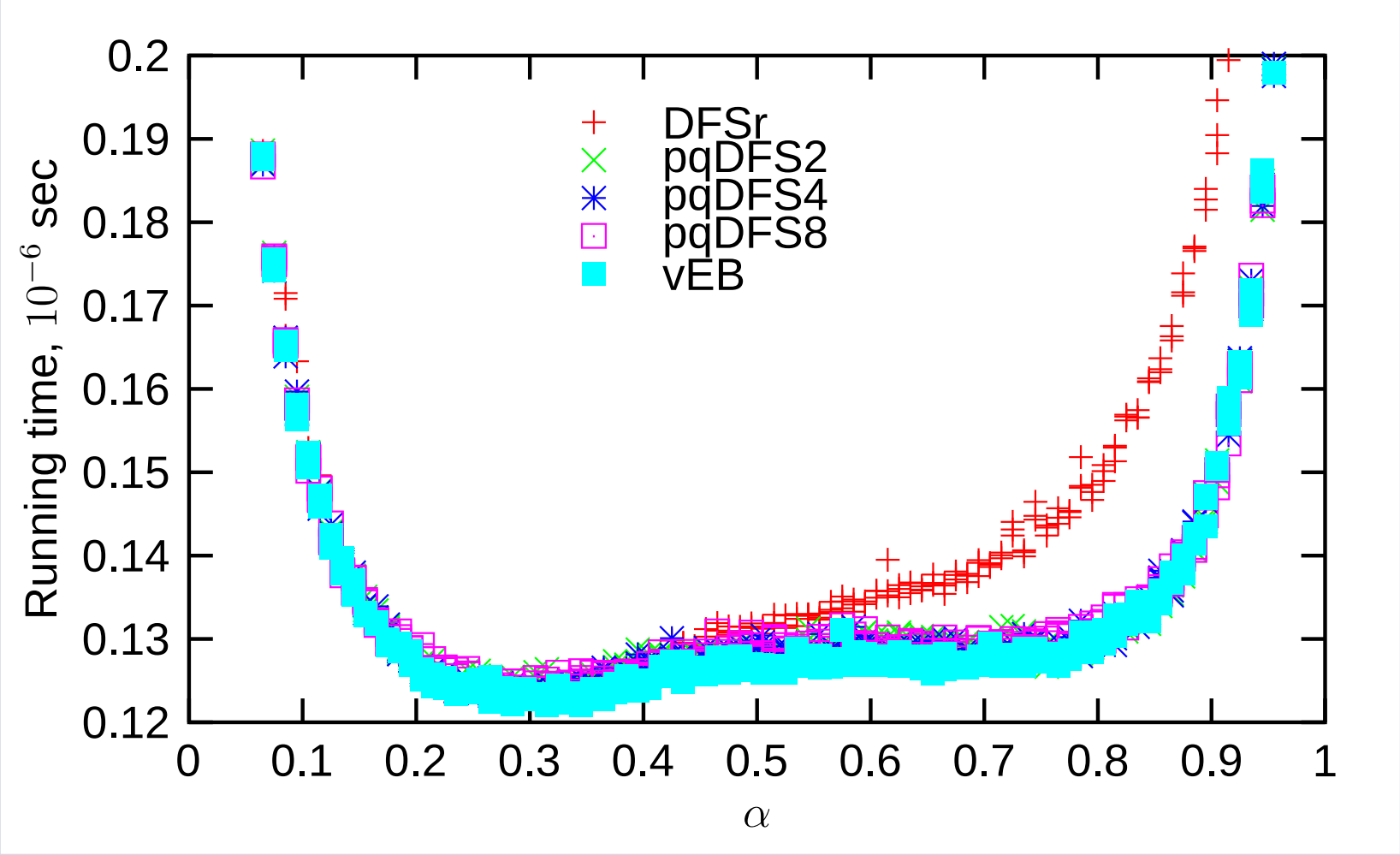
- layout the k heavest nodes in order of decreasing size
- recurse on subtrees in order of decreasing size

Blocked Layouts — veb

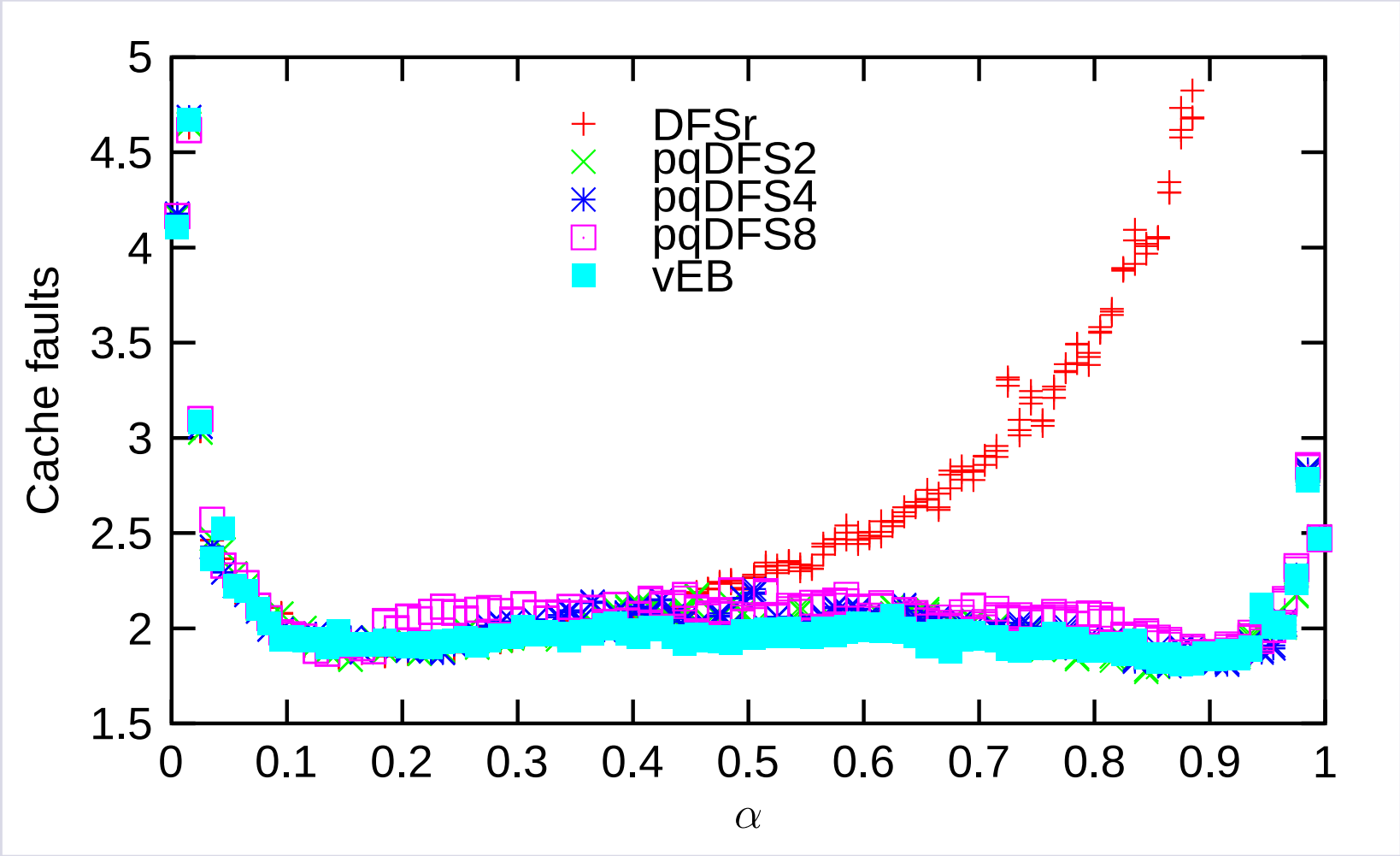


- top = $\lceil \sqrt{n} \rceil$ heavest nodes
- recurse on top and bottom trees in order of decreasing size

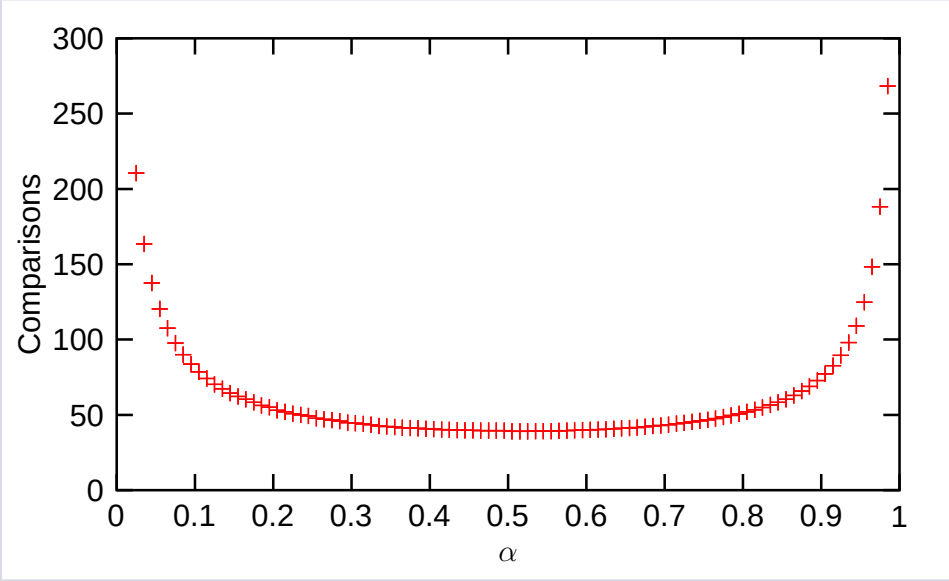
Running Time for Blocked Layouts

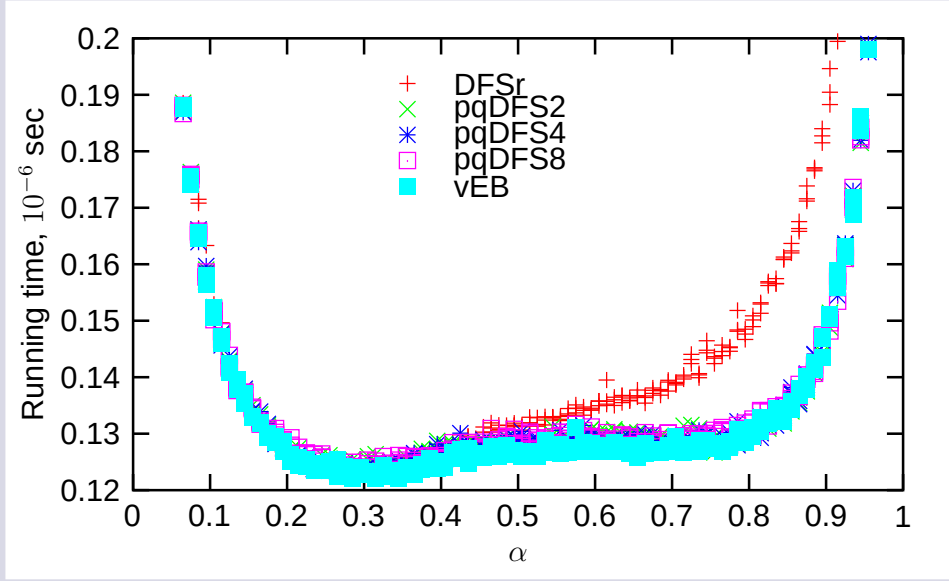


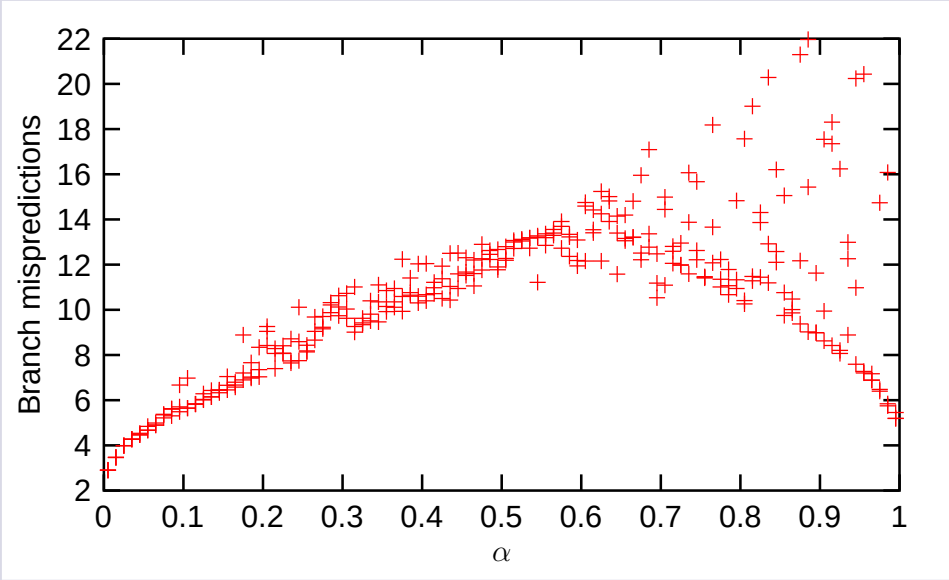
Cache Faults for Blocked Layouts

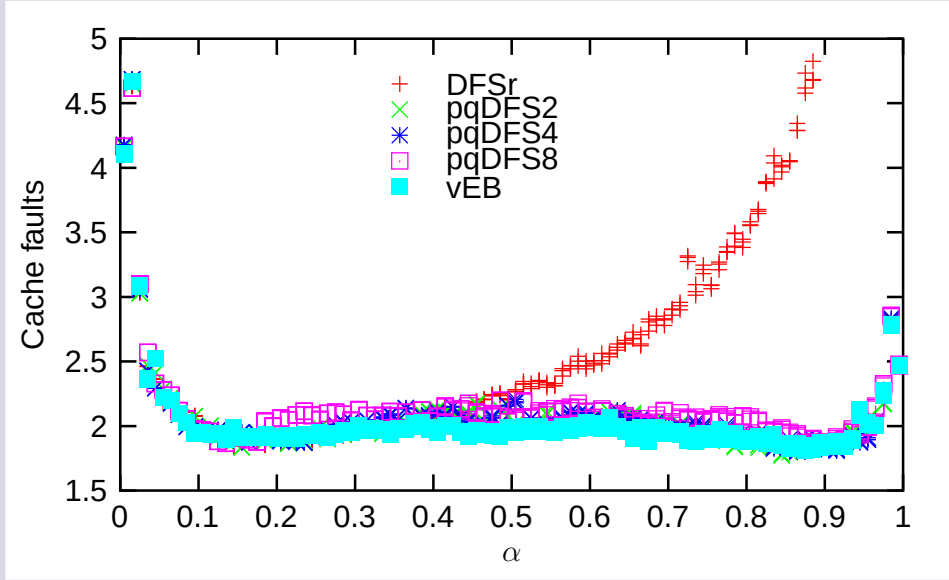


Experimental Summary









Conclusion

Skewed binary search trees

beat

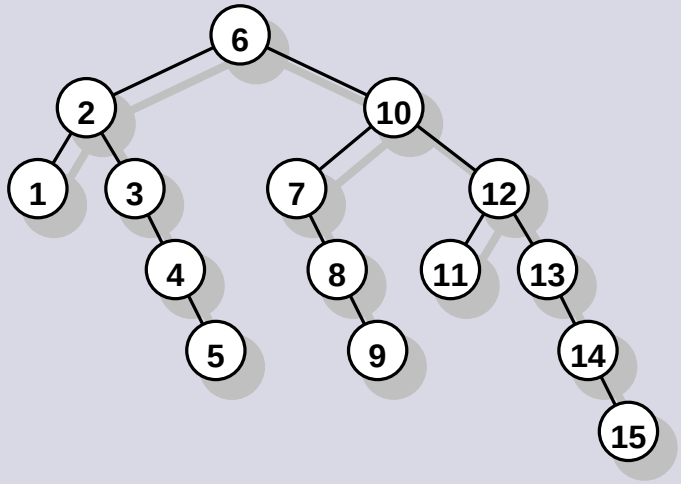
Perfectly balanced binary search trees

because

The costs going left and right are different !

Questions

- Can we give a precise theoretical explanation of the experimental results ?
- Different values of n ?
- Different computer architectures ?
- What is the best layout of a tree on a given machine ?
- What is the best tree ?
- ...



Experimental setup

- AMD Athlon XP 2400+
- 2.0 GHz
- 256 Kb L2 cache
- 64 Kb L1 data cache
- 64 Kb L1 instruction cache
- 1Gb RAM
- Linux 2.6.8.1
- GCC 3.3.2
- Tree nodes = 12 bytes