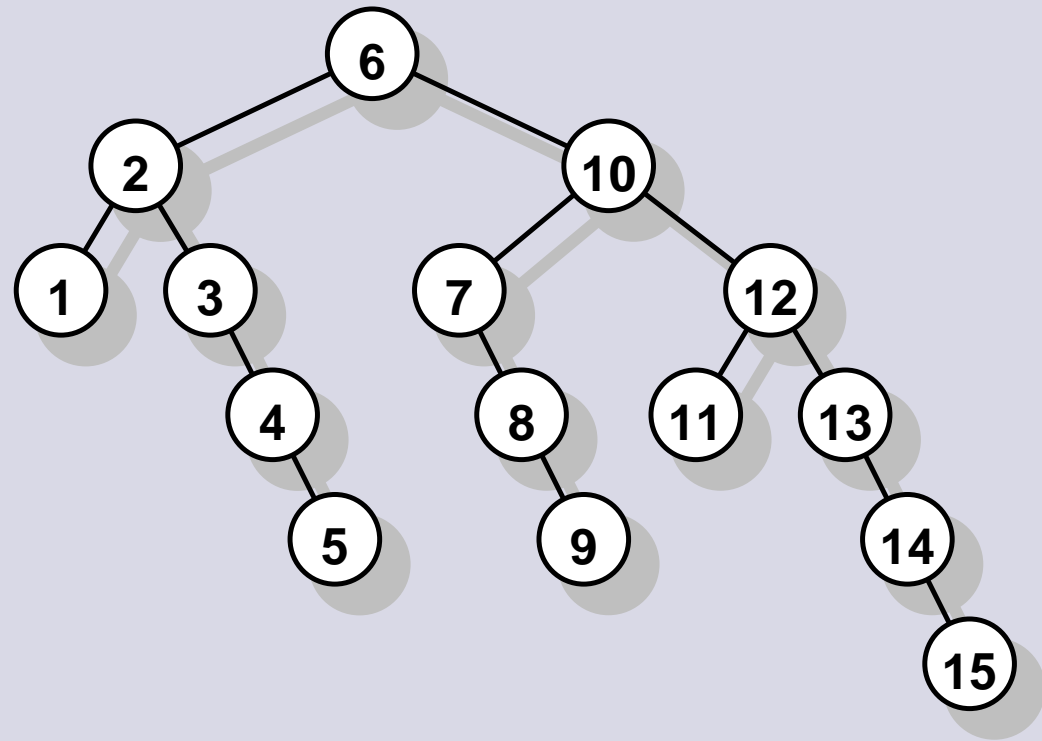


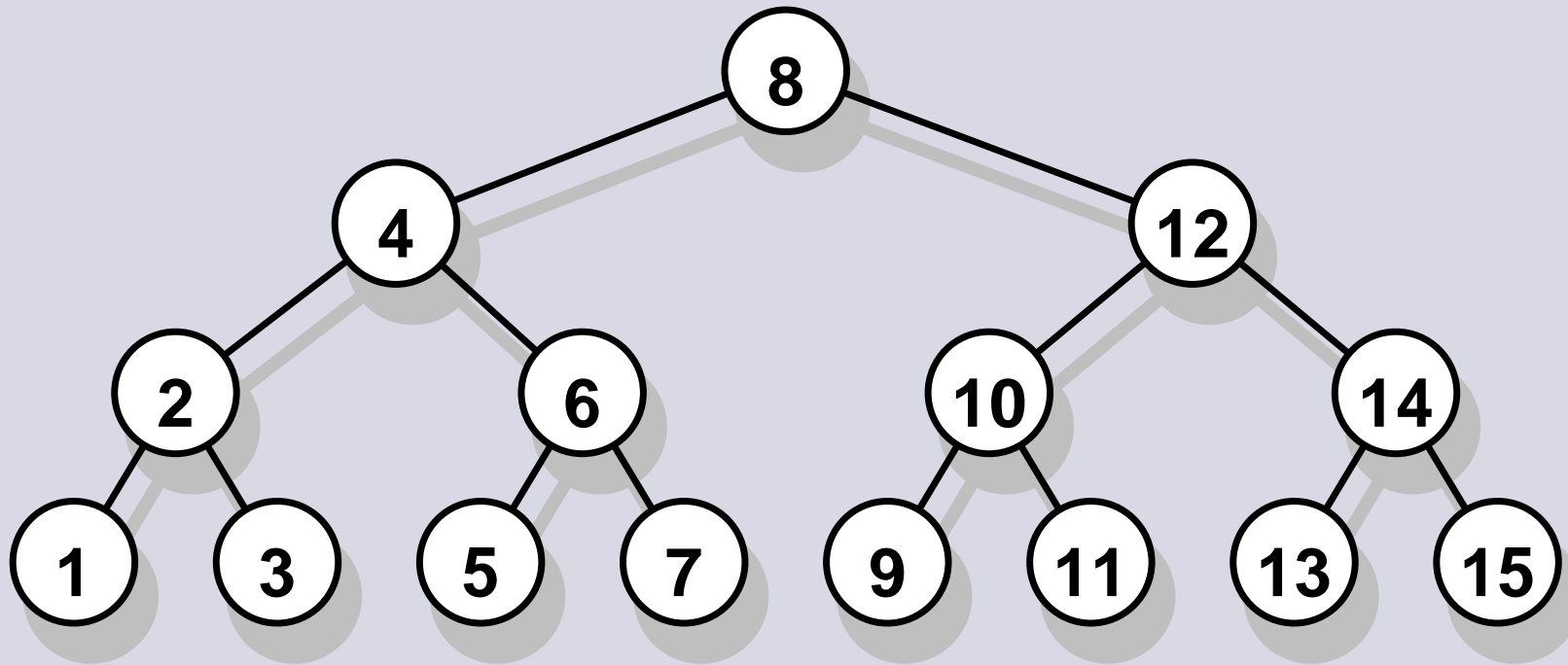
Skewed Binary Search Trees



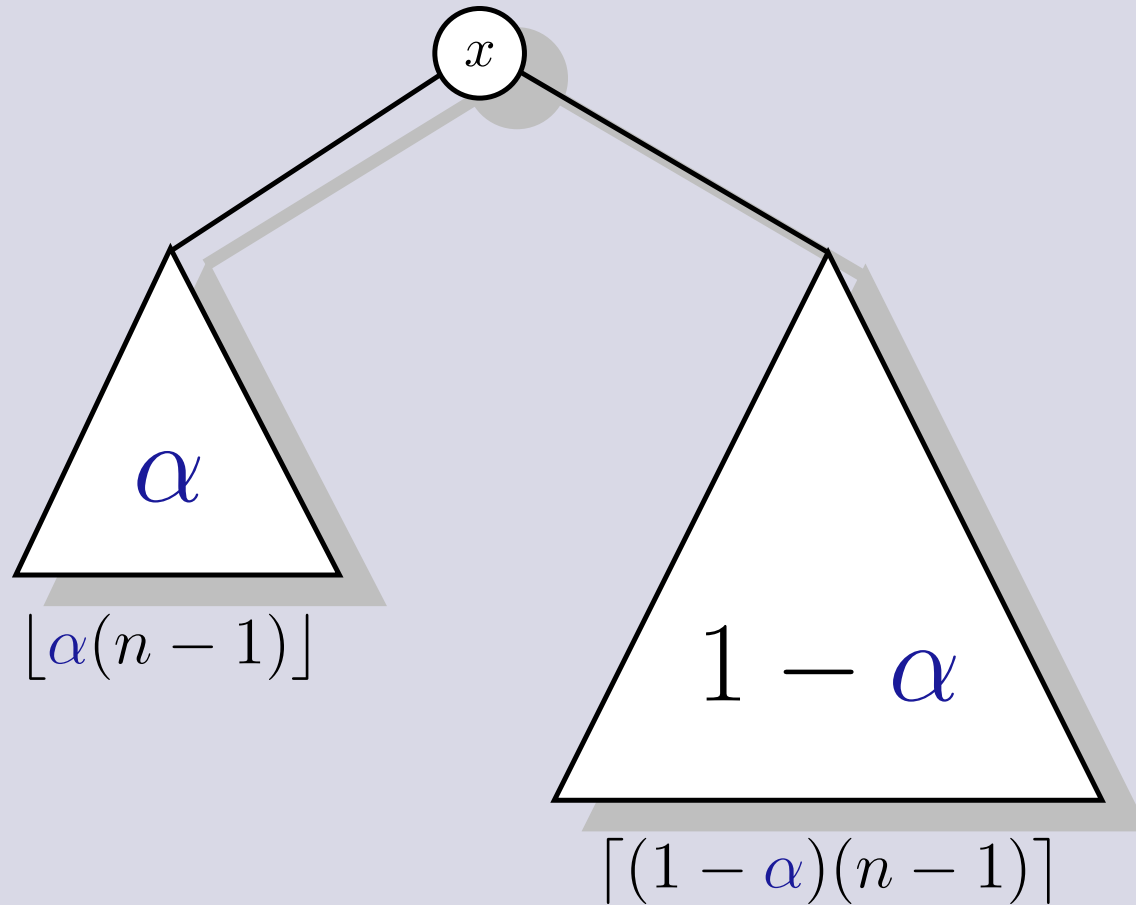
Gabriel Moruz
University of Aarhus

Joint work with Gerth Stølting Brodal

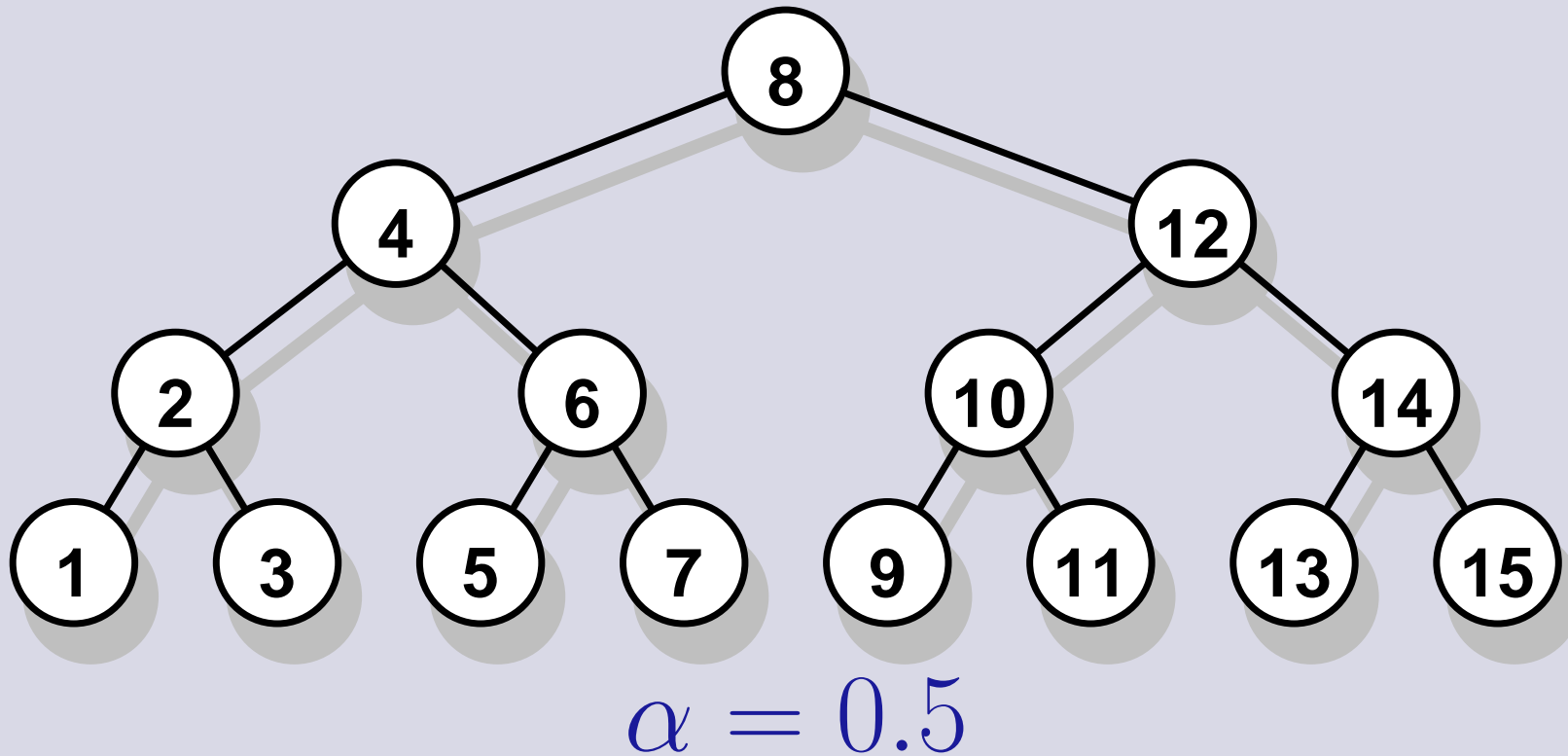
Perfectly Balanced Search Trees



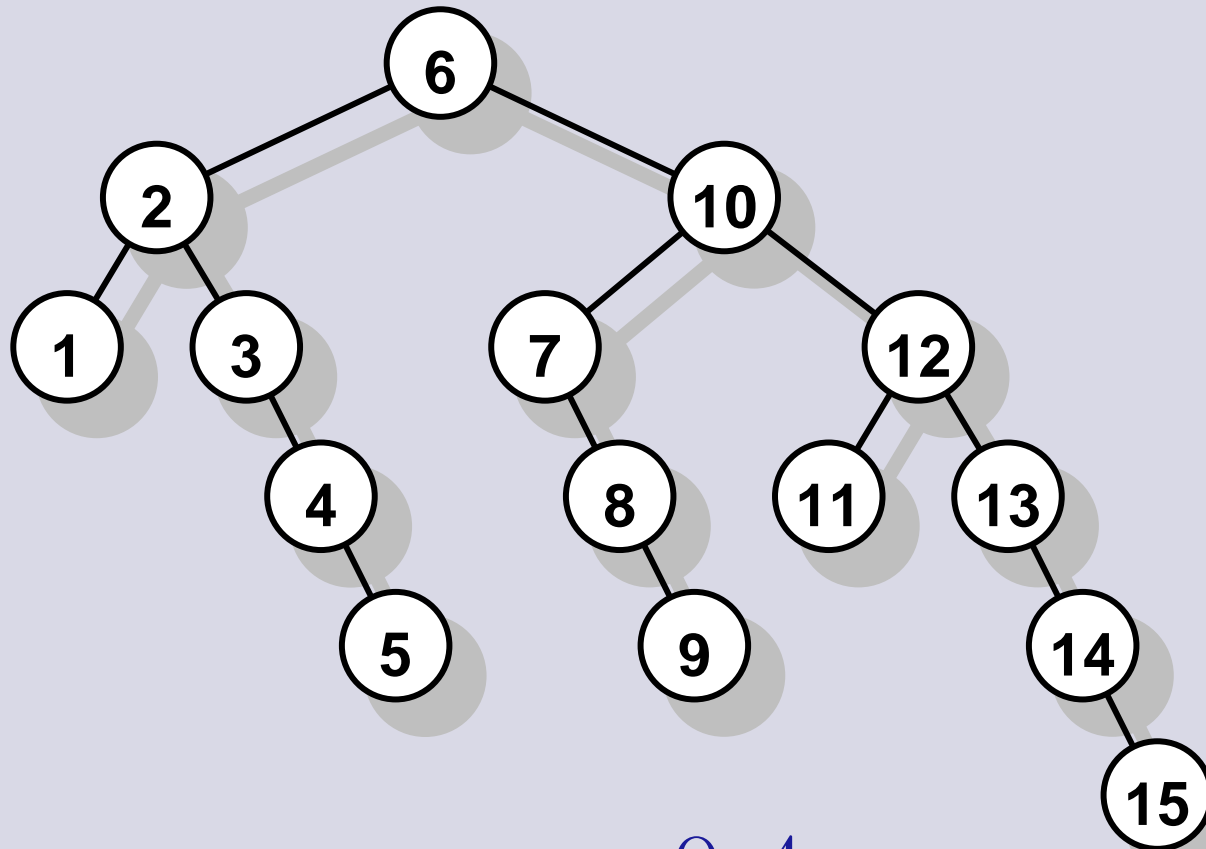
Skewed Binary Search Trees



Skewed Binary Search Trees

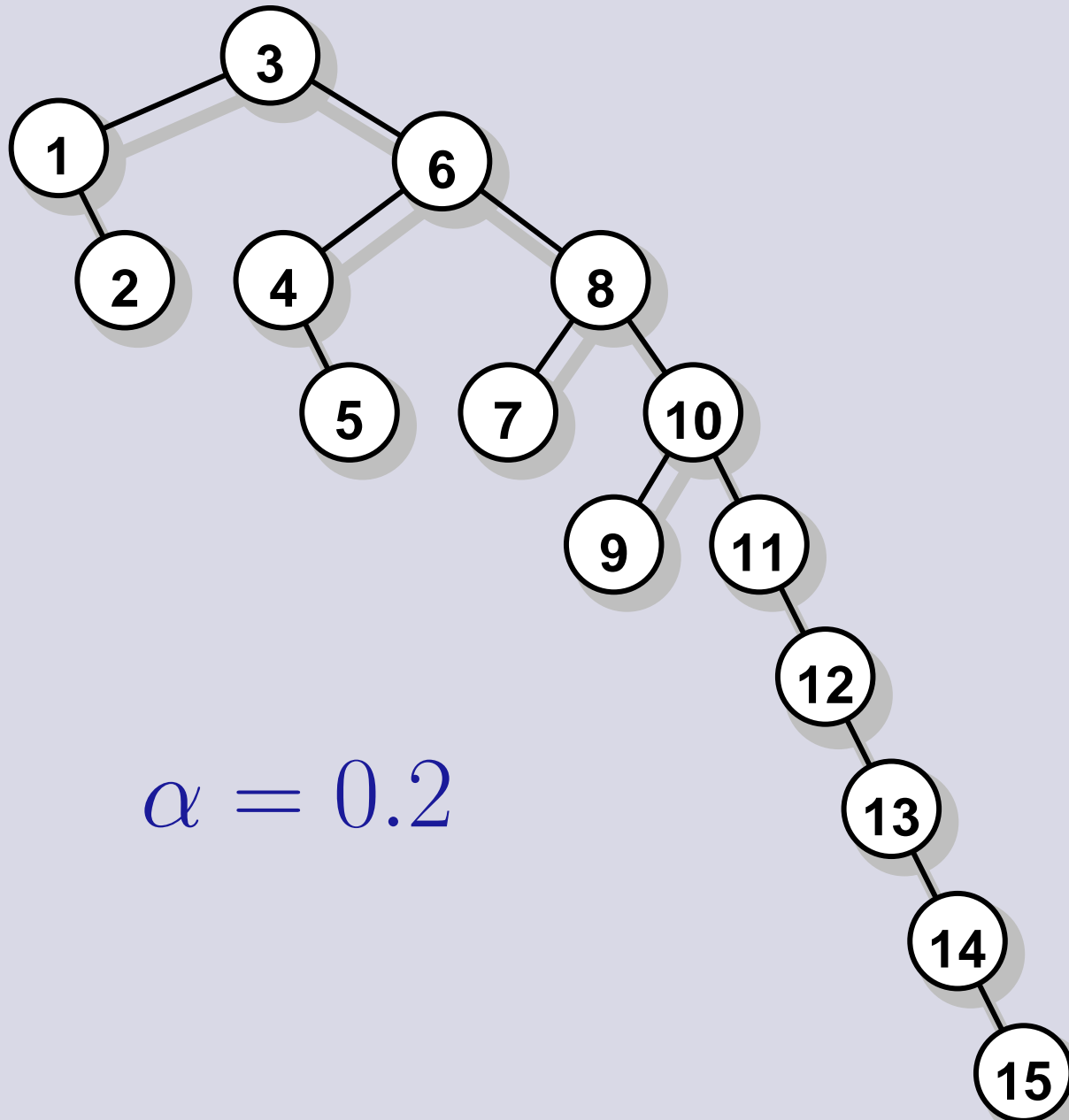


Skewed Binary Search Trees



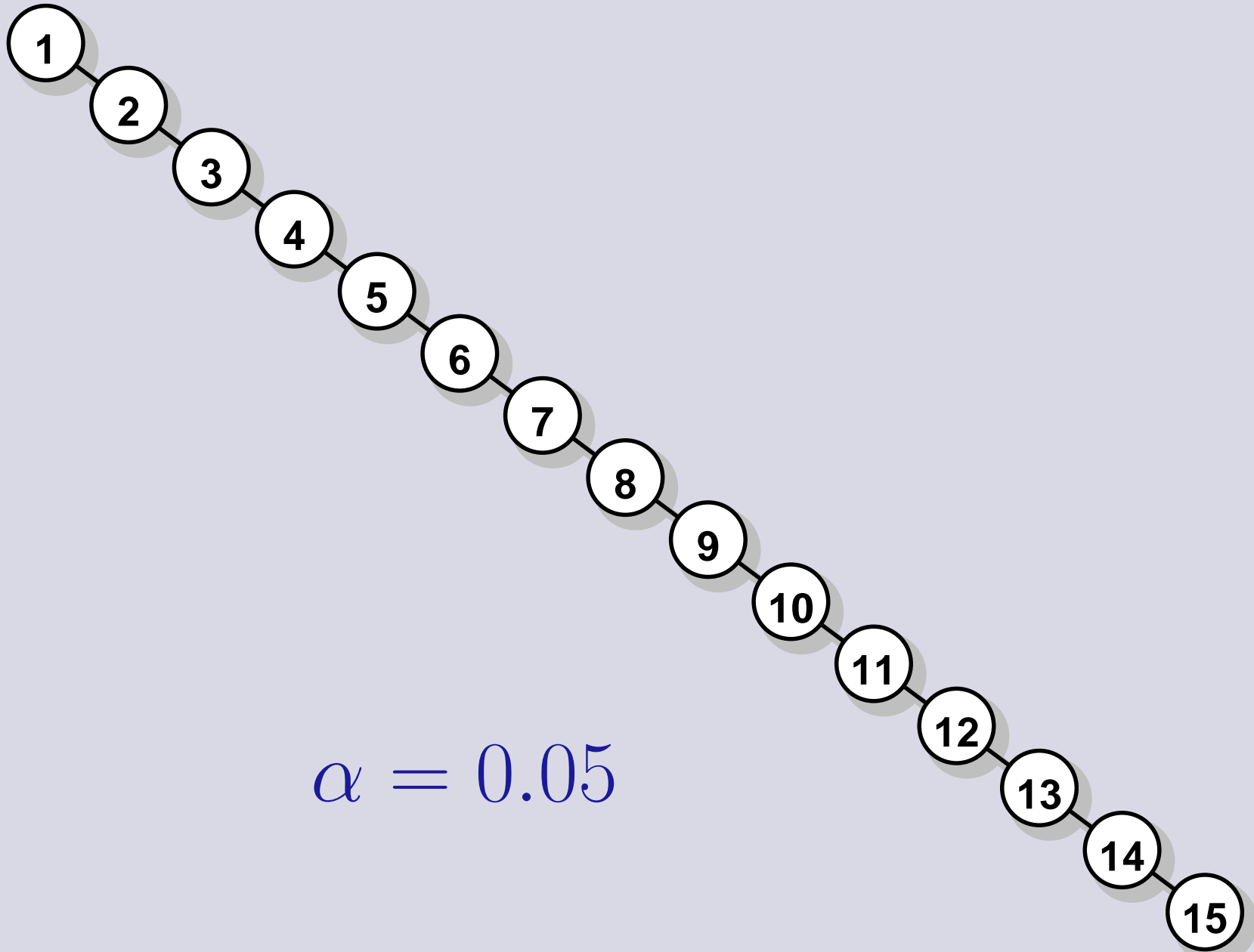
$$\alpha = 0.4$$

Skewed Binary Search Trees



$$\alpha = 0.2$$

Skewed Binary Search Trees



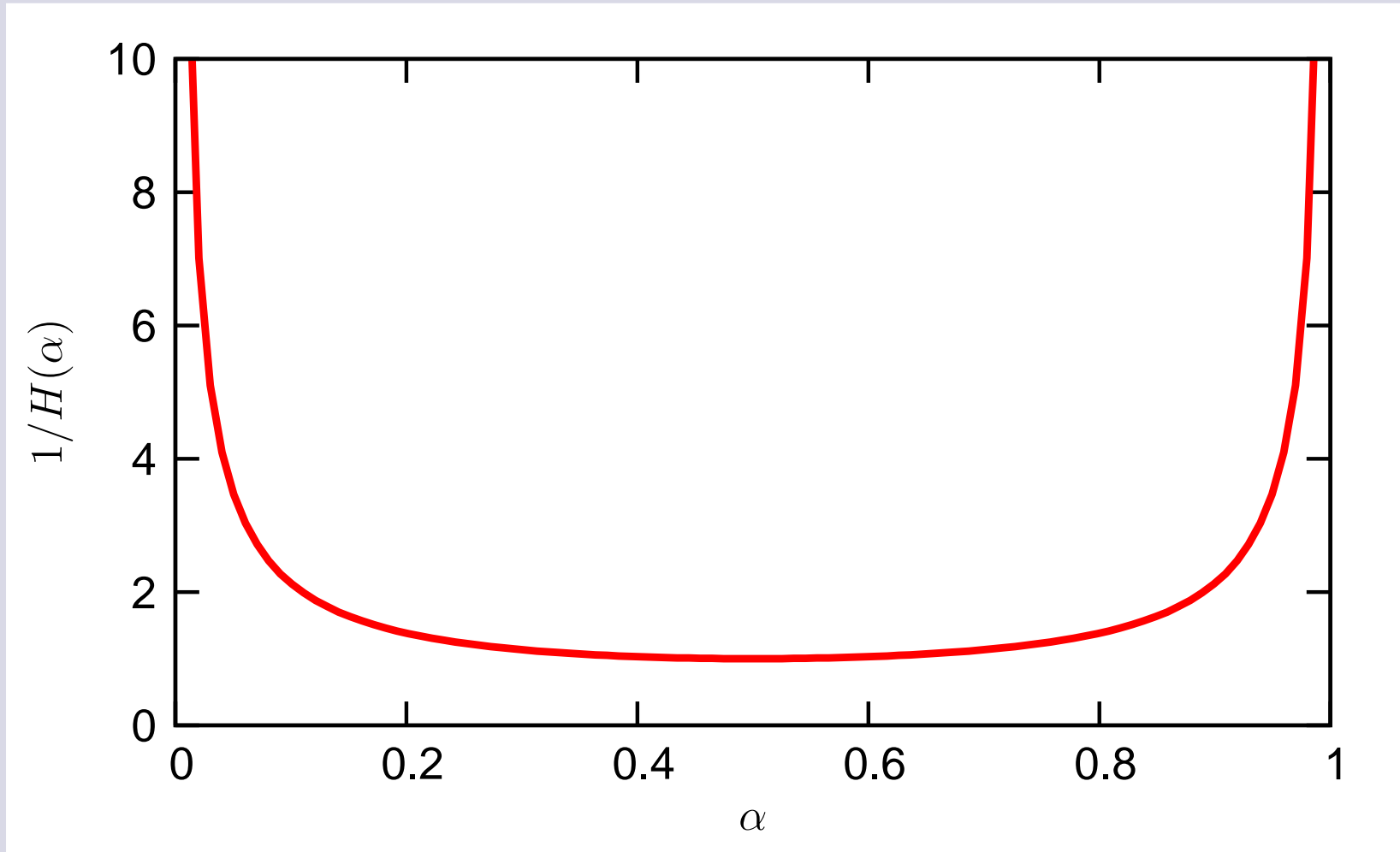
Skewed Binary Search Trees

— Average Node Depth

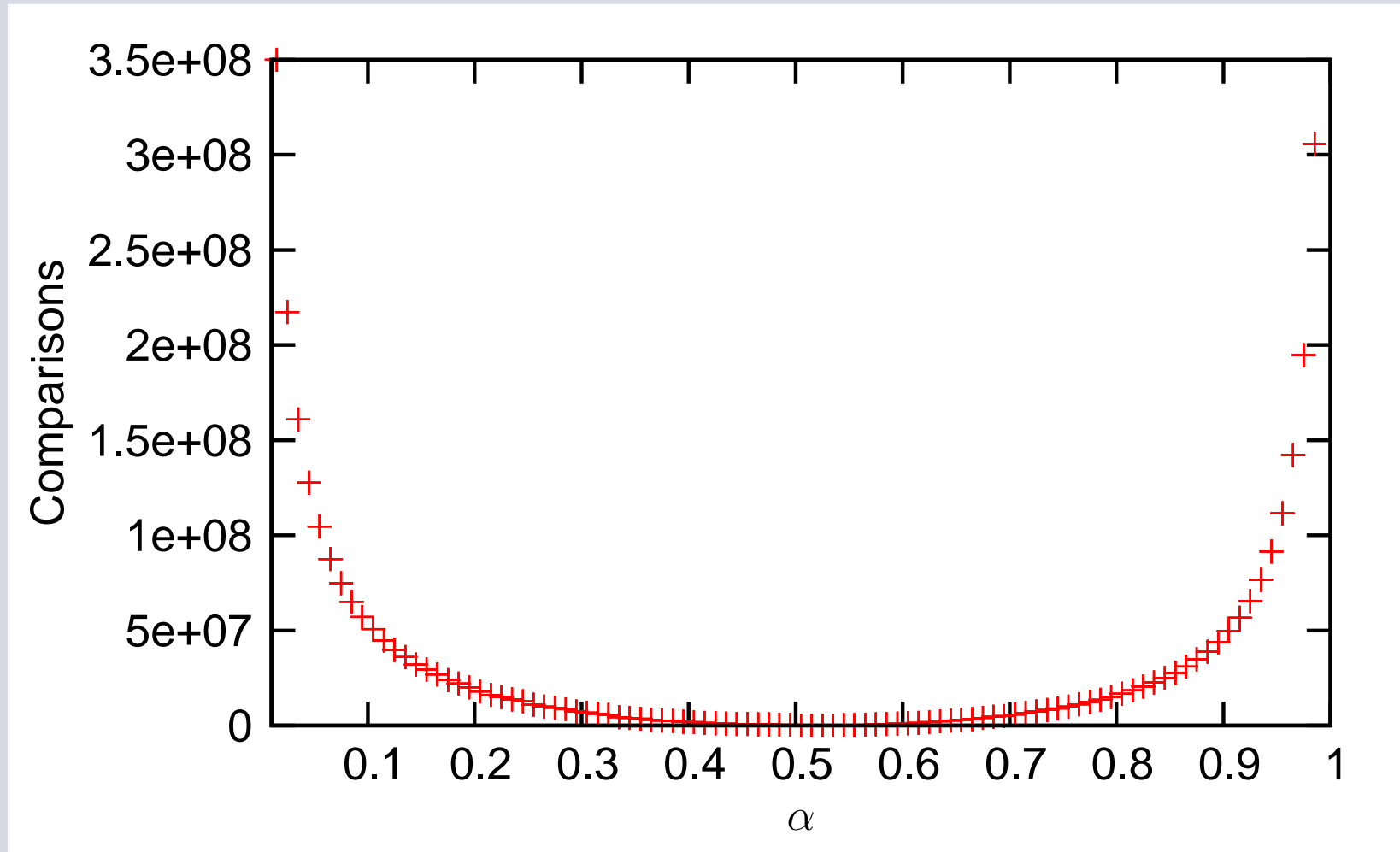
$$\leq \frac{1}{\underbrace{-\alpha \log_2 \alpha - (1 - \alpha) \log_2(1 - \alpha)}_{H(\alpha)}} \cdot \frac{n + 1}{n} \cdot \log_2(n + 1) - 2$$

Nievergelt and E. M. Reingold, 1972

$$1/H(\alpha)$$

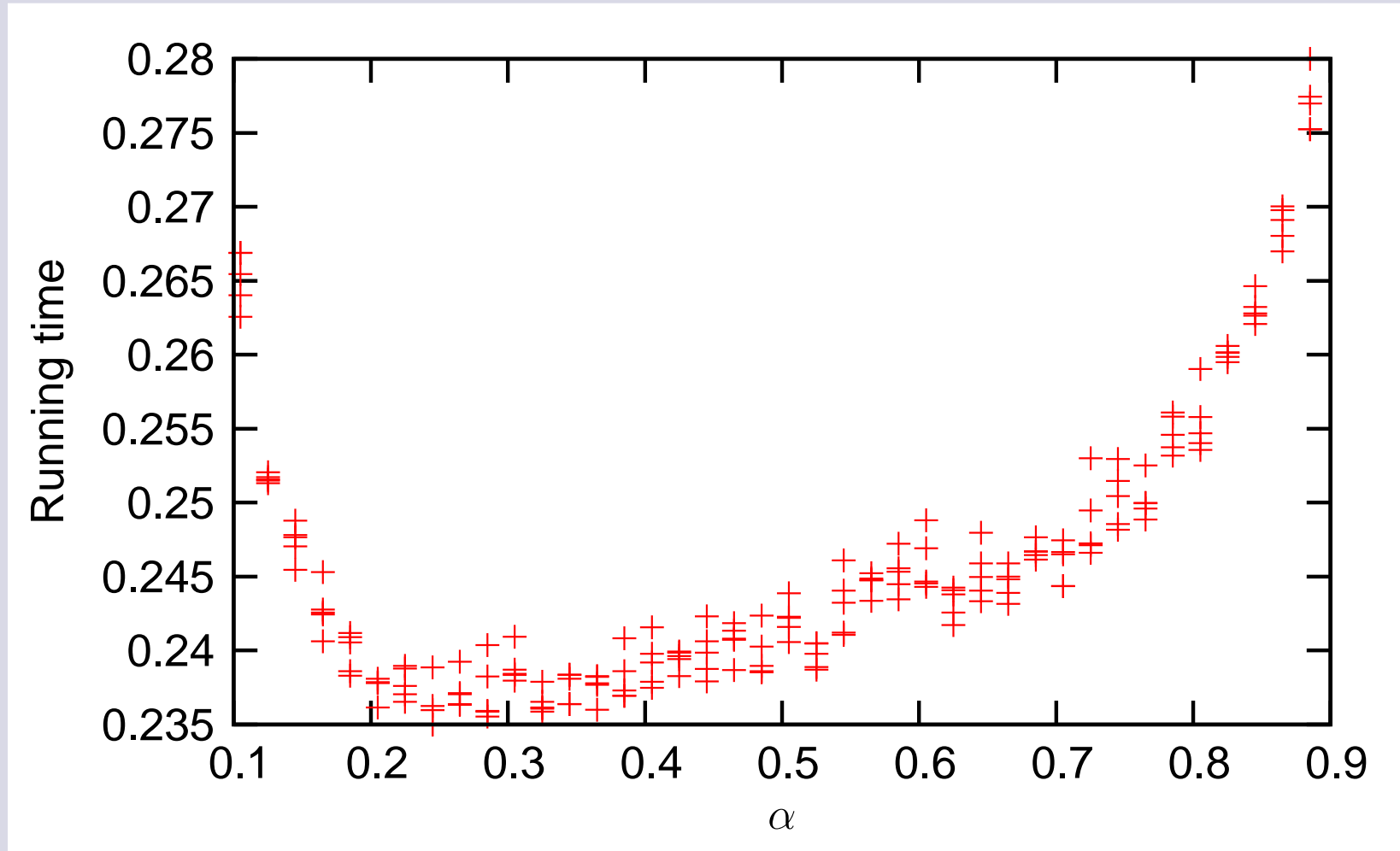


Comparisons



$$n = 50.000$$

Running Time



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

Conclusion

Skewed binary search trees

can beat

Perfectly balanced binary search trees !

Why ?

Why ?

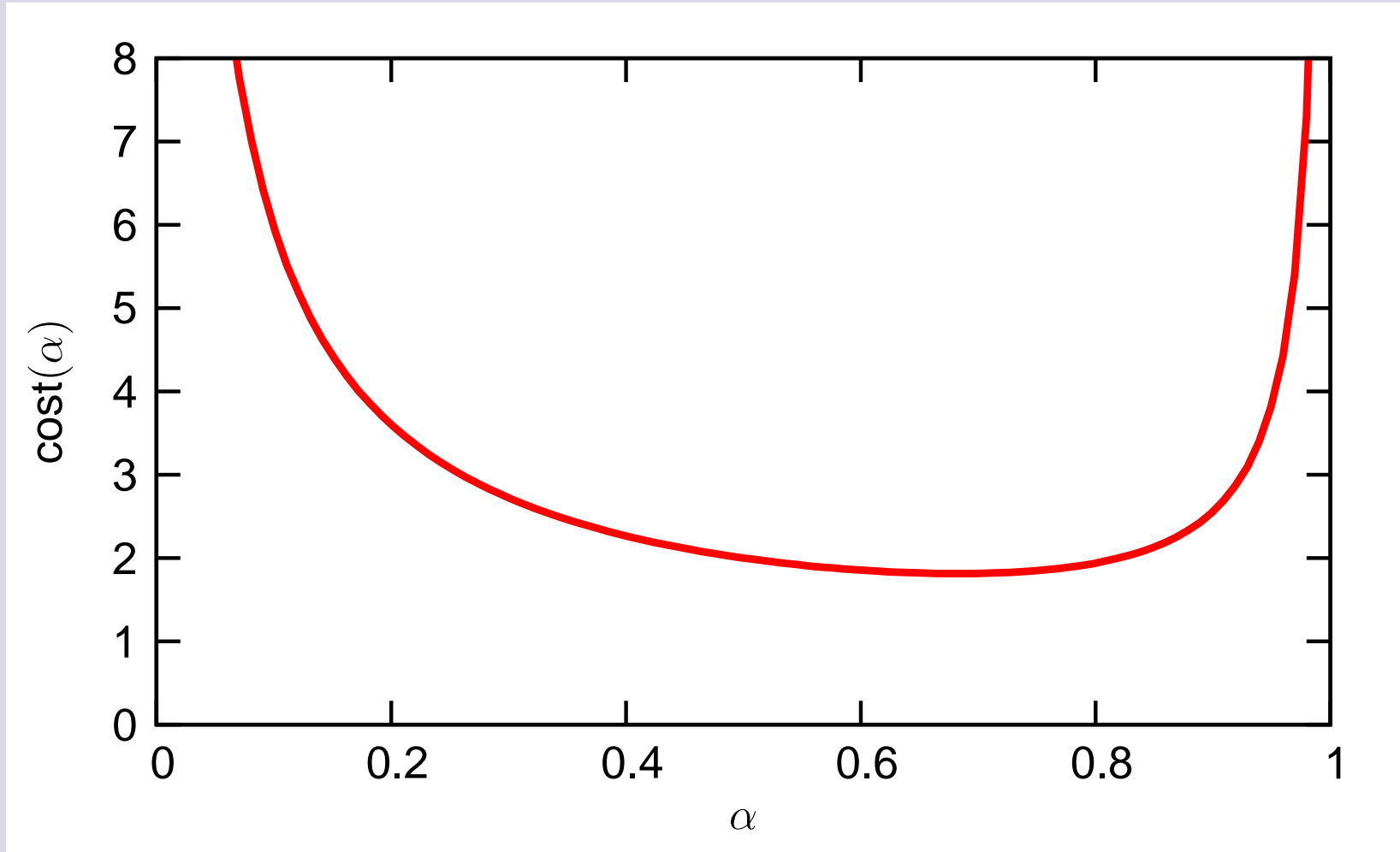
The costs going left and right are different !

Possible reasons

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

Expected Cost

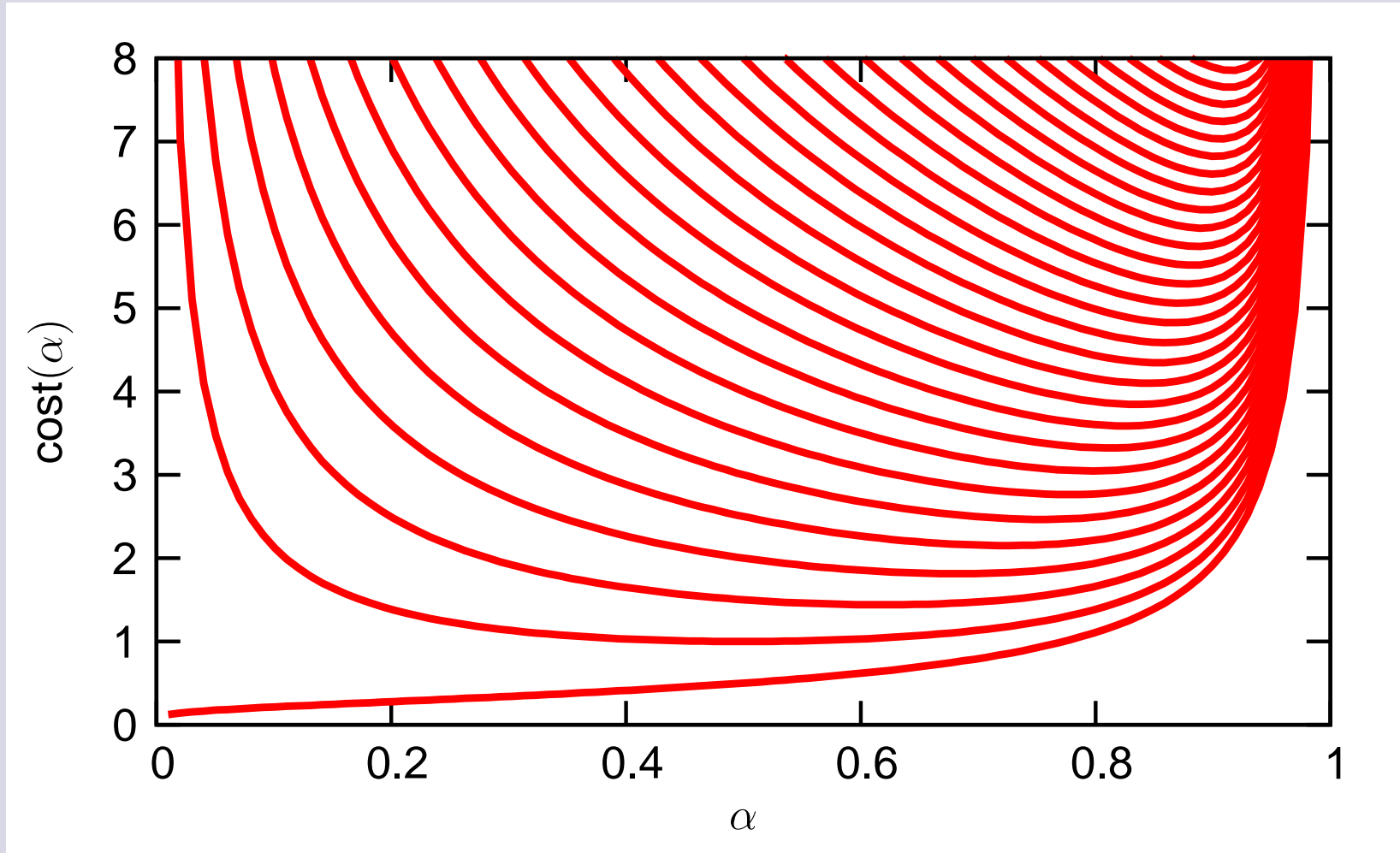
$$\text{cost}(\alpha) = (\alpha \cdot \{\text{left cost}\} + (1 - \alpha) \cdot \{\text{right cost}\}) / H(\alpha)$$



left cost = 1 and right cost = 3

Expected Cost

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



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

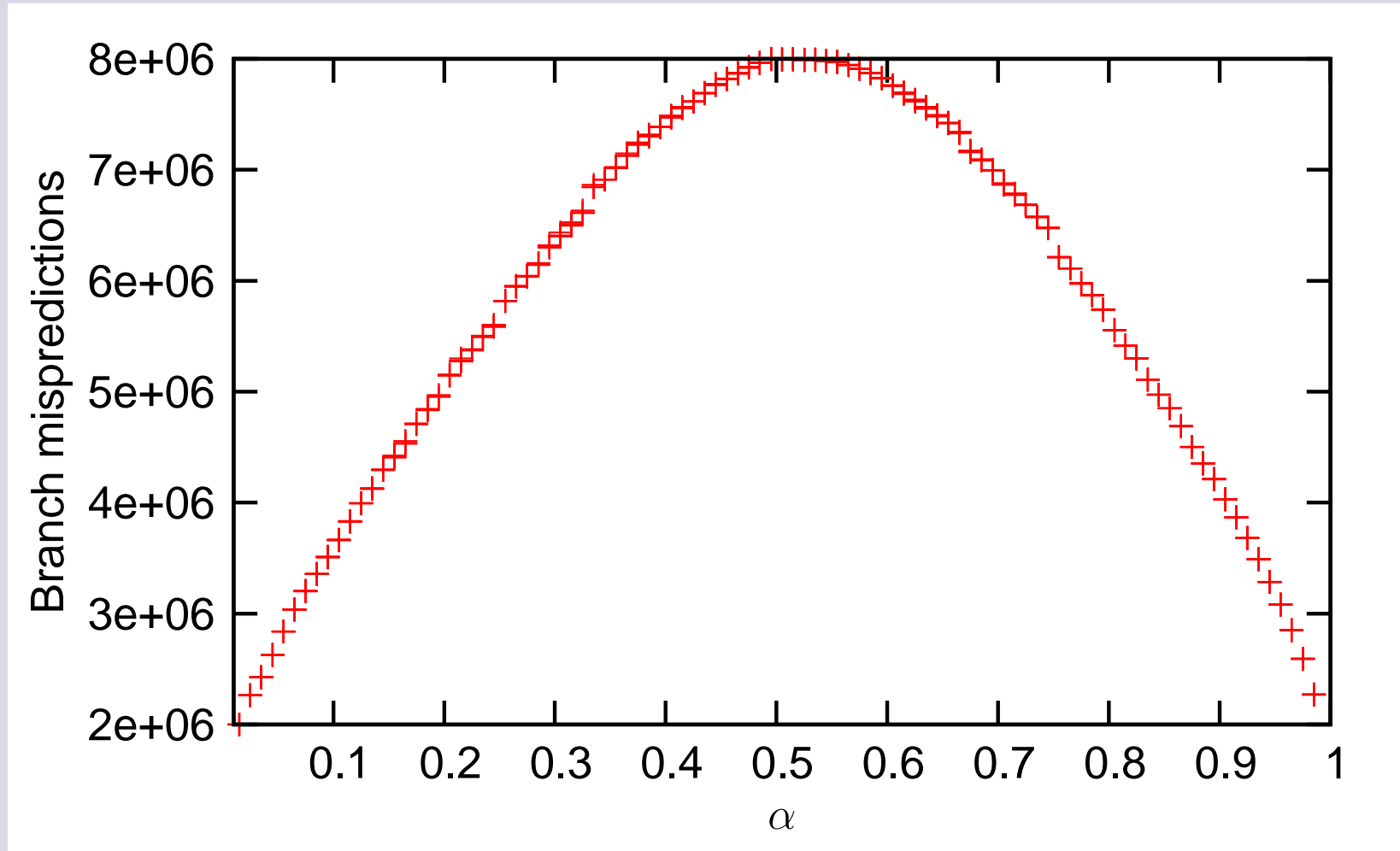
Experimental setup

- 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
- No unsuccessful searches

Search Code

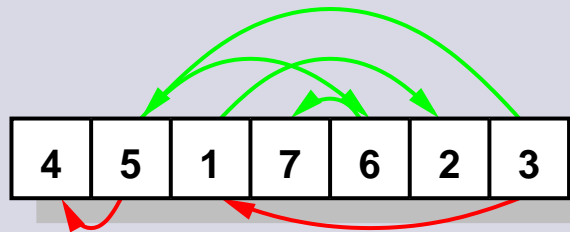
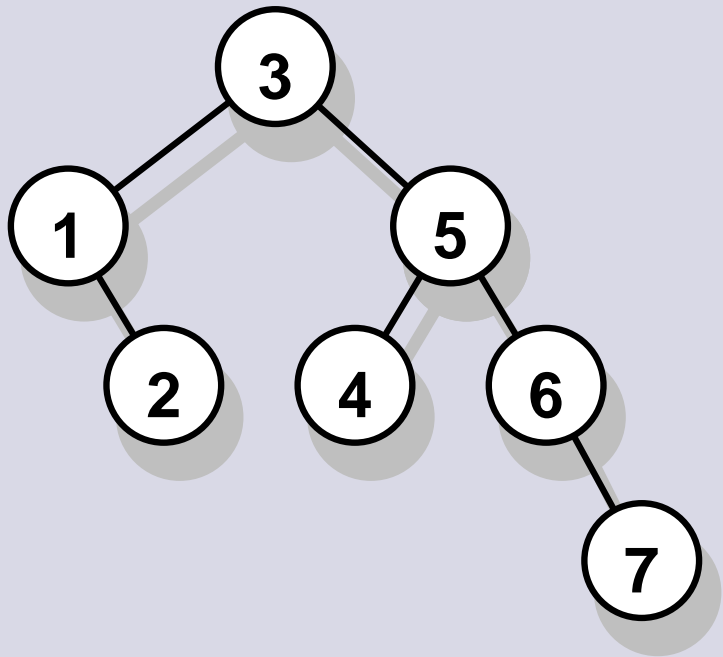
```
while(root!=NULLV)
{
    if(key==t[root].key)
        return root;
    if(key>t[root].key)
        root=t[root].right;
    else
        root=t[root].left;
}
```

Branch Mispredictions

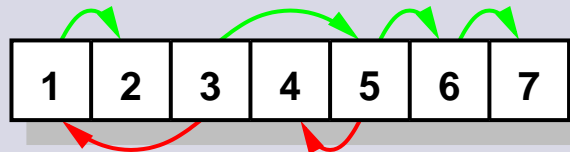


$$n = 50.000$$

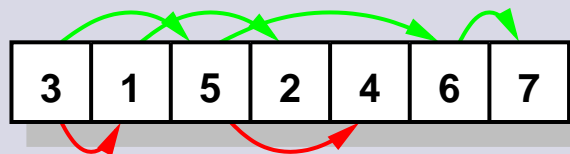
Simple Layouts



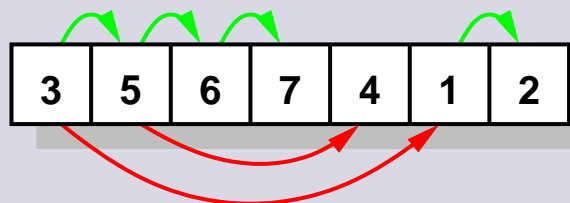
Random – $O\left(\frac{\log n}{H(\alpha)}\right)$ I/Os



Inorder – $O\left(\frac{\log n}{H(\alpha)} - \log B\right)$ I/Os

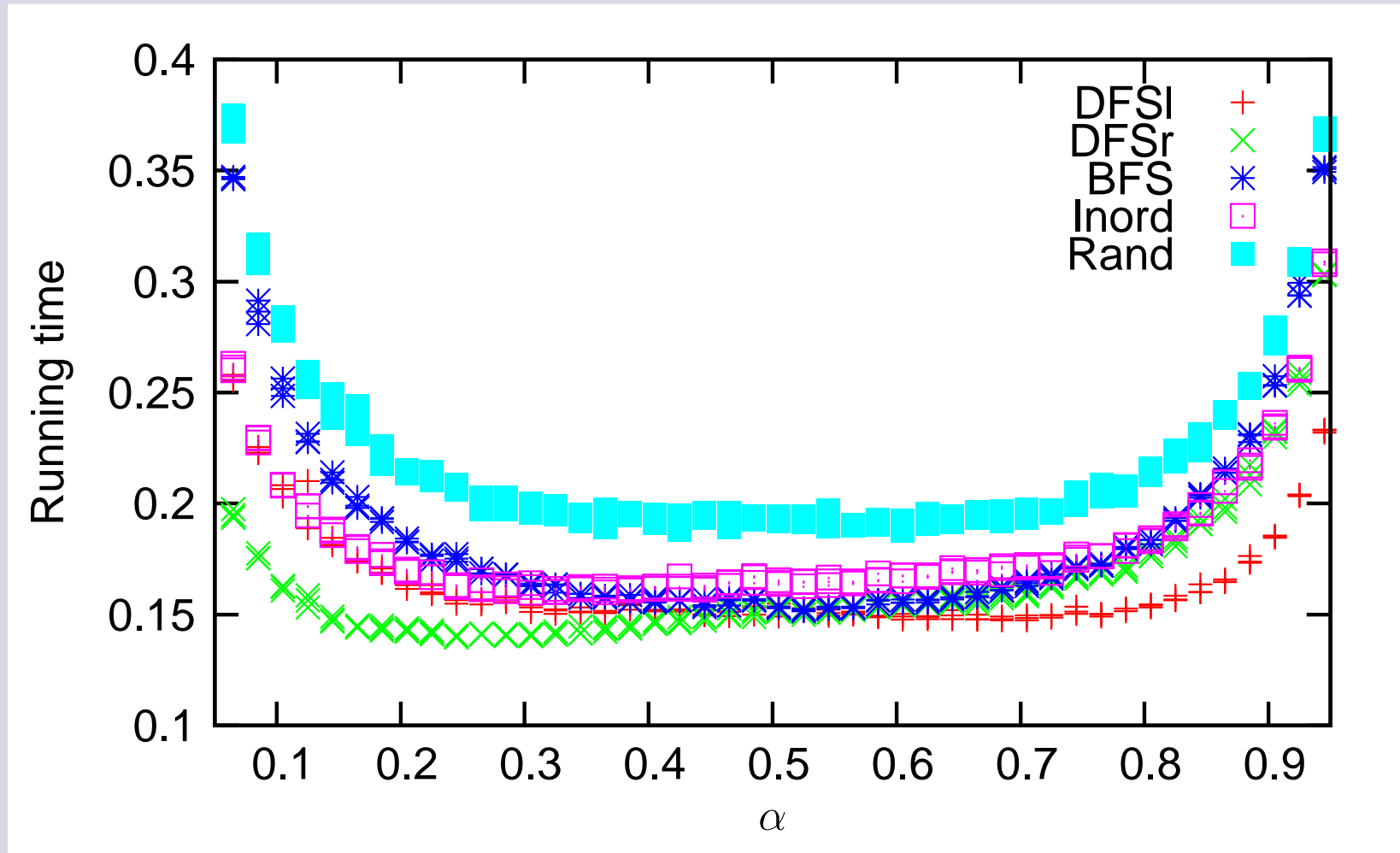


BFS – $O\left(\frac{\log n}{H(\alpha)} - \log B\right)$ I/Os



DFSr – $O\left(\frac{\alpha + (1-\alpha)/B}{H(\alpha)} \cdot \log n\right)$ I/Os.

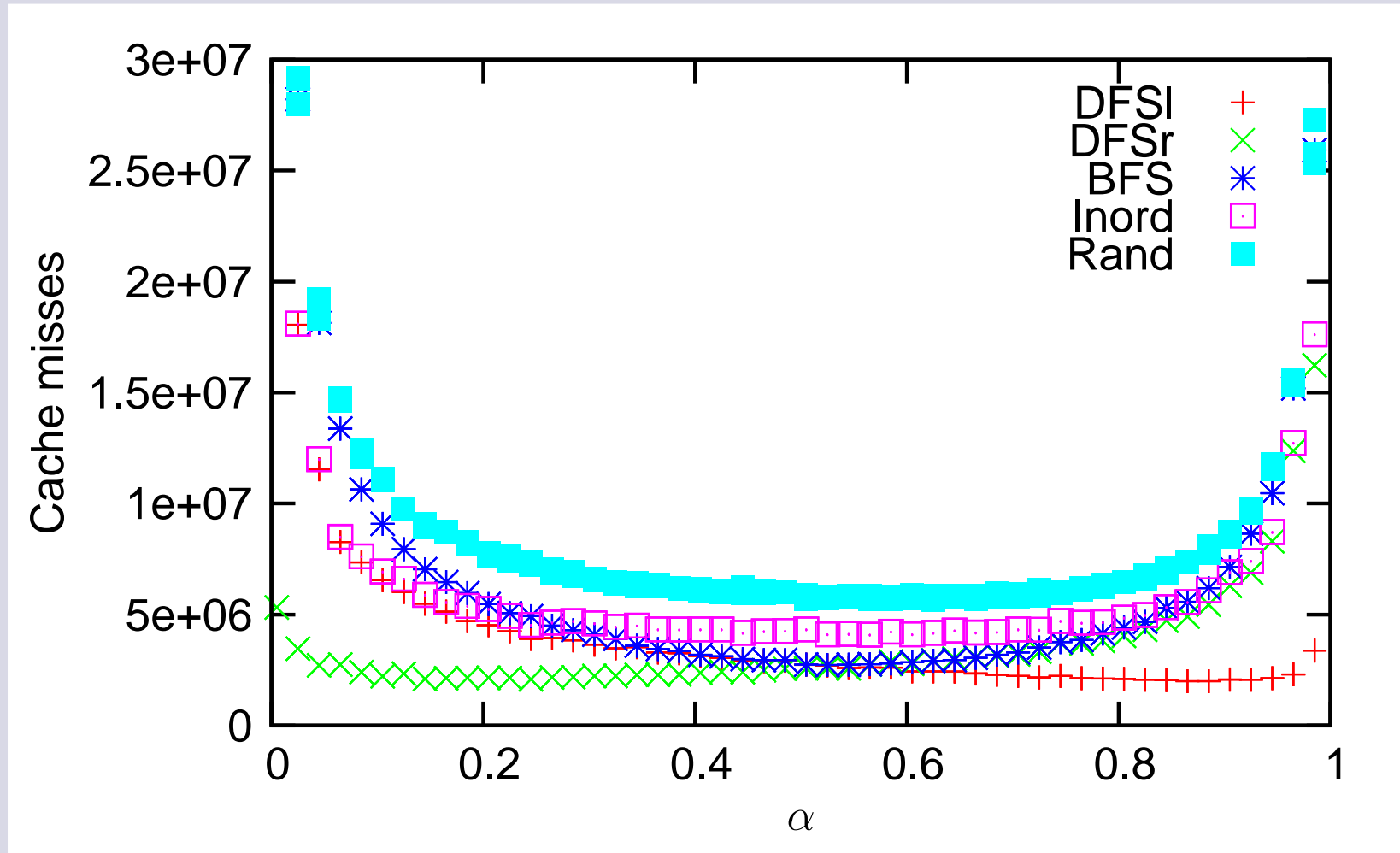
Running Time for Simple Layouts



DFS < Inorder < BFS < Random

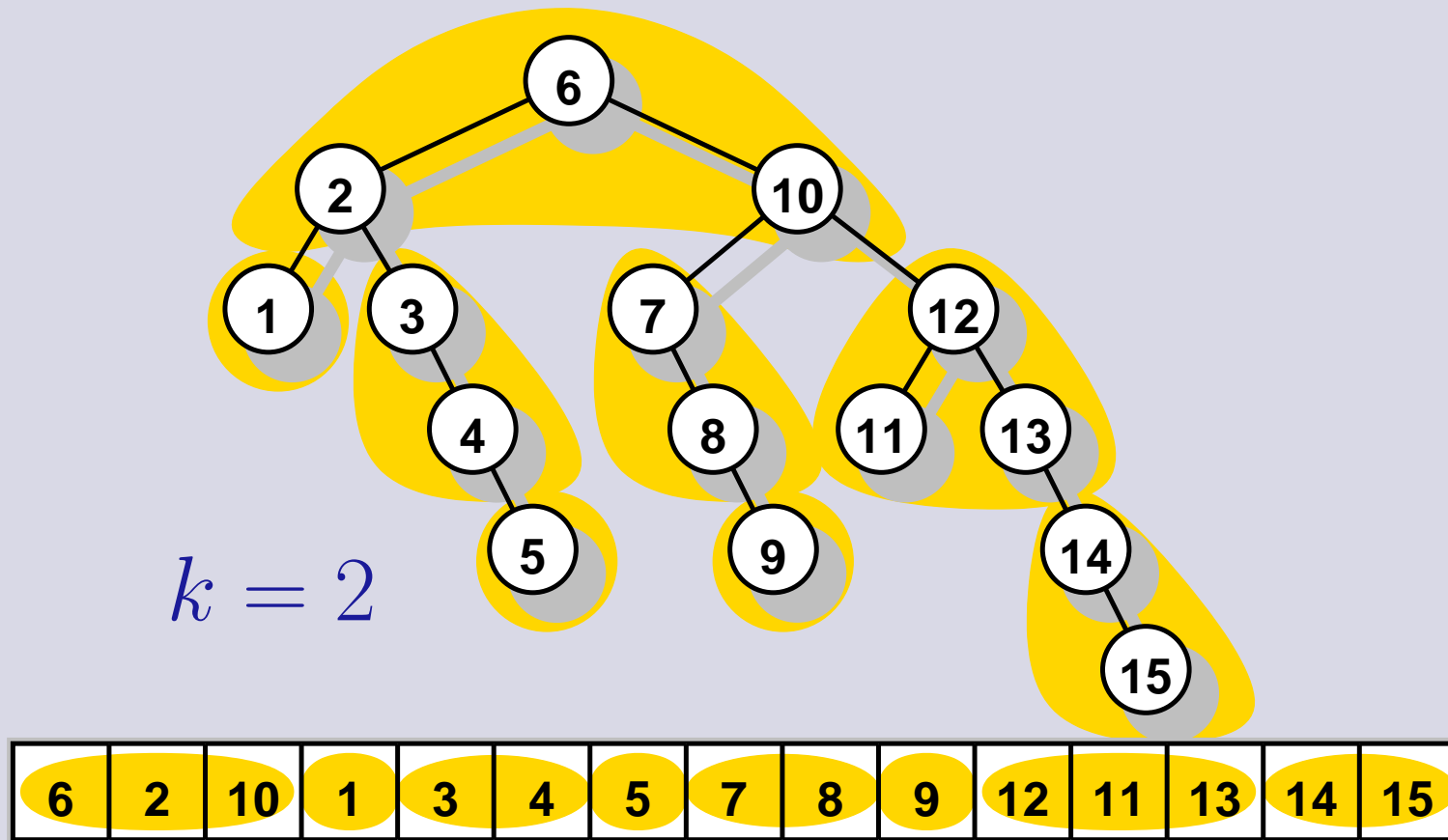
DFS achieves the best performance for $\alpha \approx 0.2$!

Cache Faults for Simple Layouts



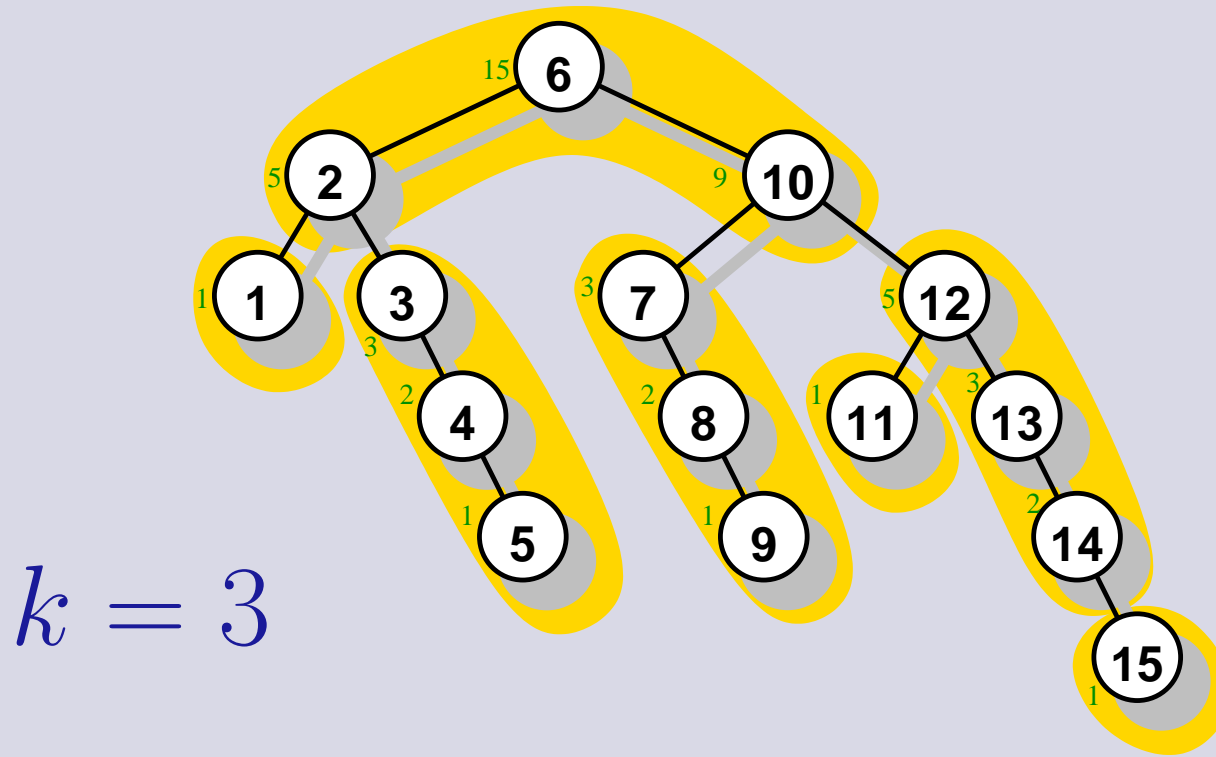
DFS \approx expected left cost = 1 and right cost = $1/B$.

Blocked Layouts — k -level blocking

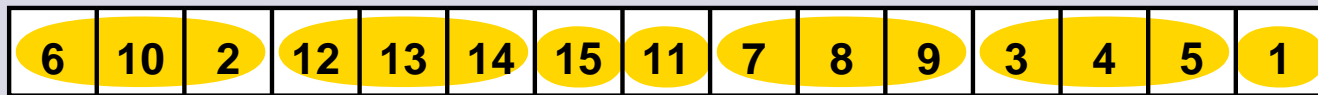


- layout the nodes of the first k levels
- recurse on subtrees
- a search uses $O(\log_B n / H(\alpha))$ I/Os

Blocked Layouts — pqDFS k

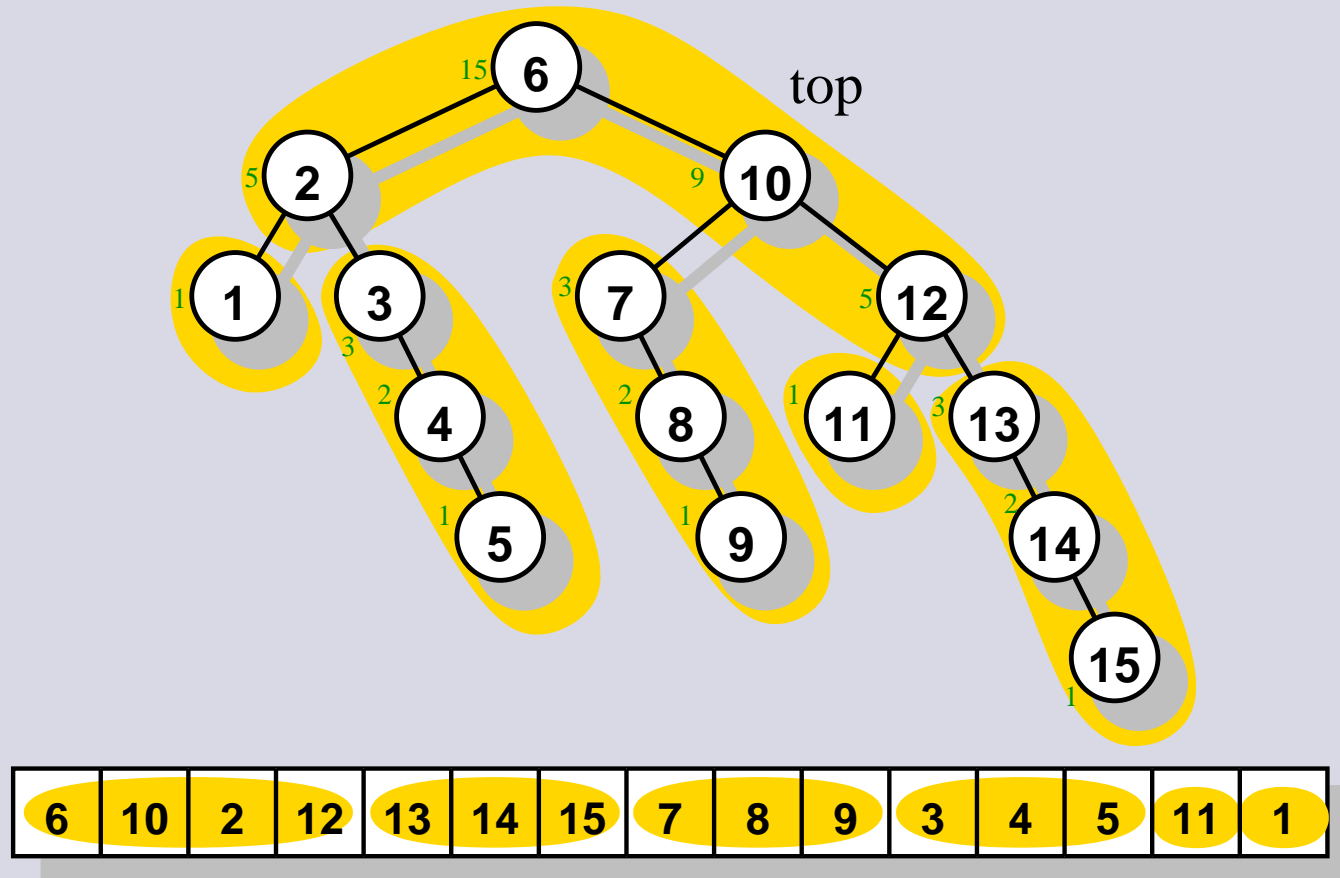


$$k = 3$$



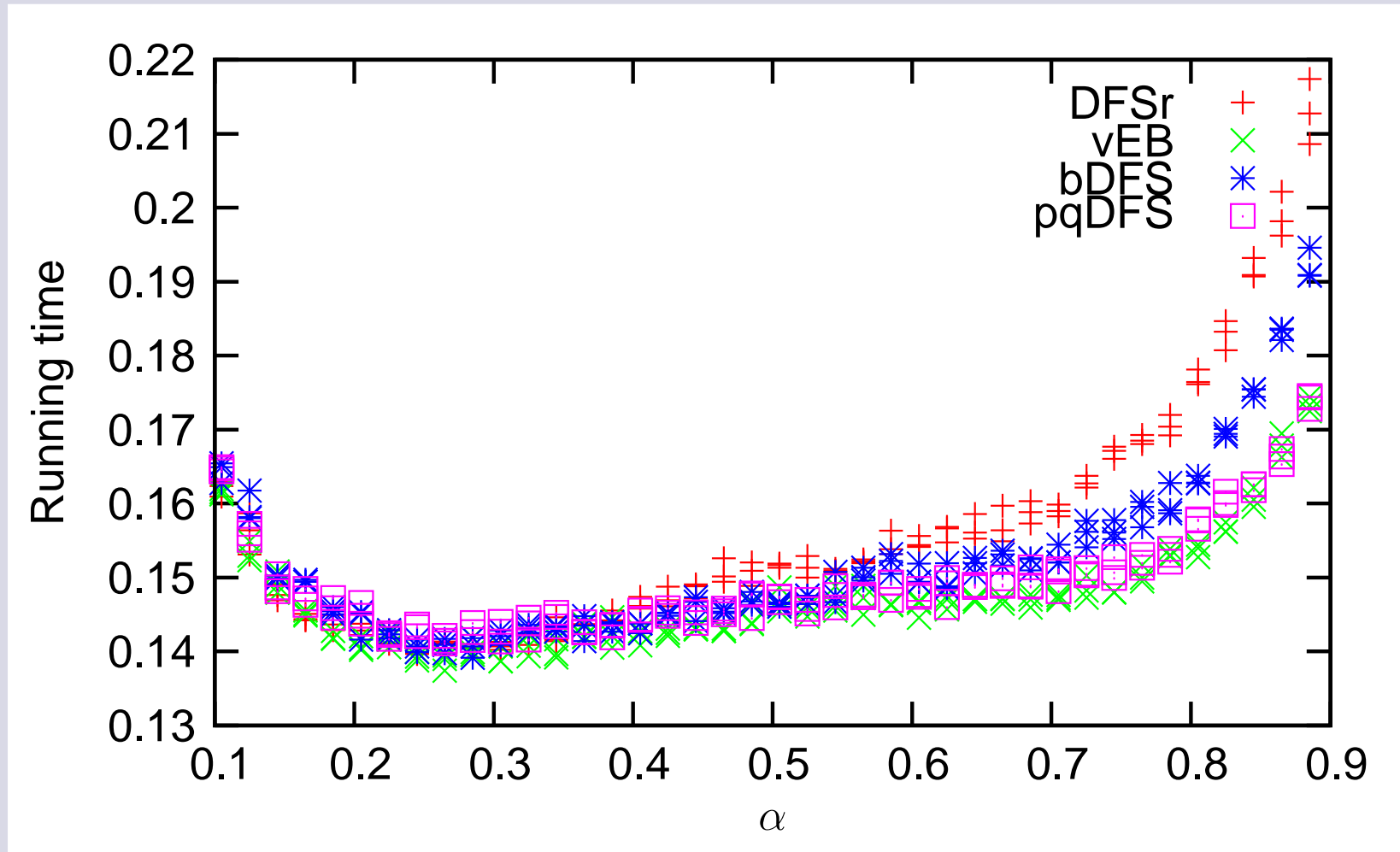
- layout the k heaviest nodes in order of decreasing size
- recurse on subtrees in order of decreasing size
- a search uses $O(\log_{B\alpha+1} n)$ I/Os

Blocked Layouts — veb



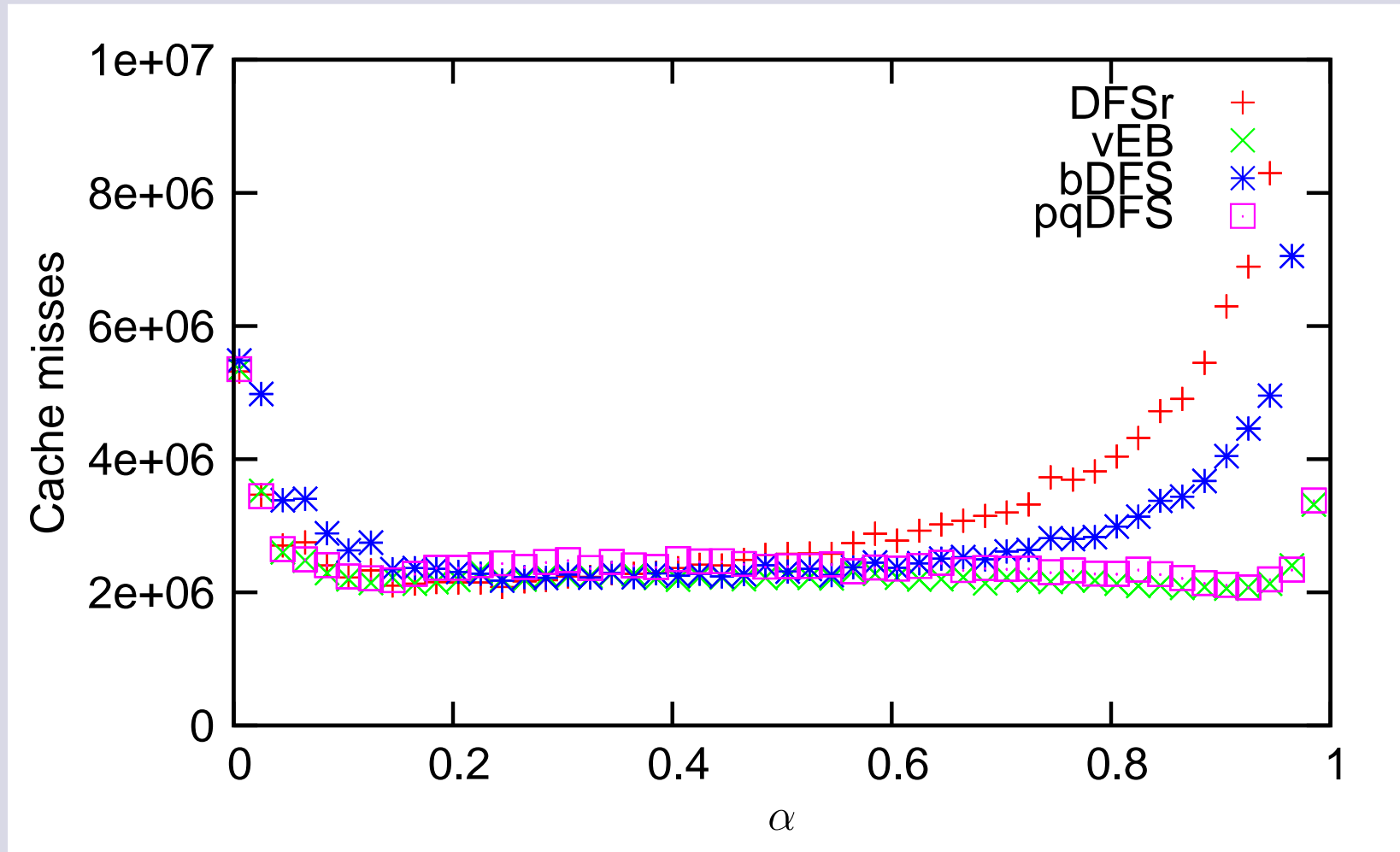
- $top = \lceil \sqrt{n} \rceil$ heaviest nodes
- recurse on top and bottom trees in order of decreasing size
- a search uses $O(\log_{B^{\alpha+1}} n)$ I/Os

Running Time for Blocked Layouts



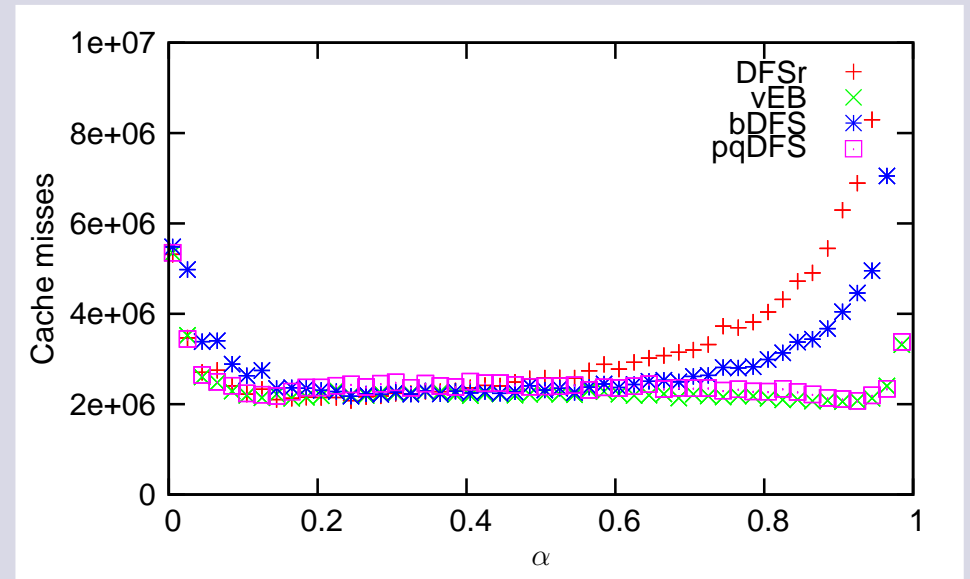
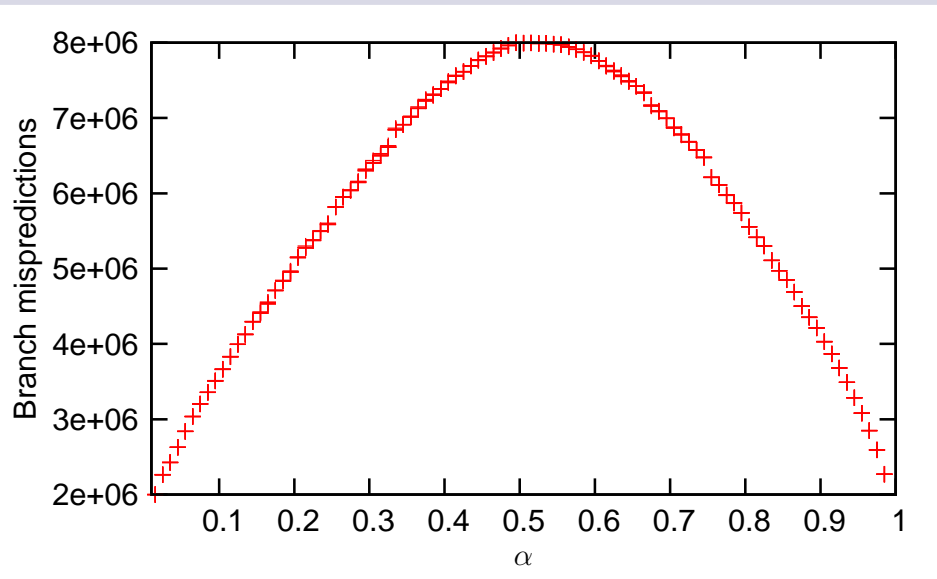
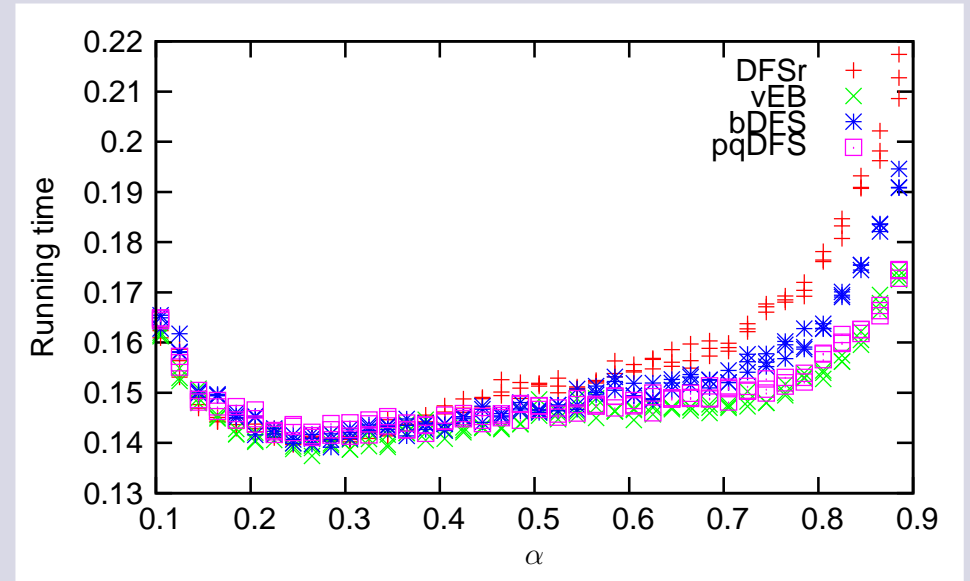
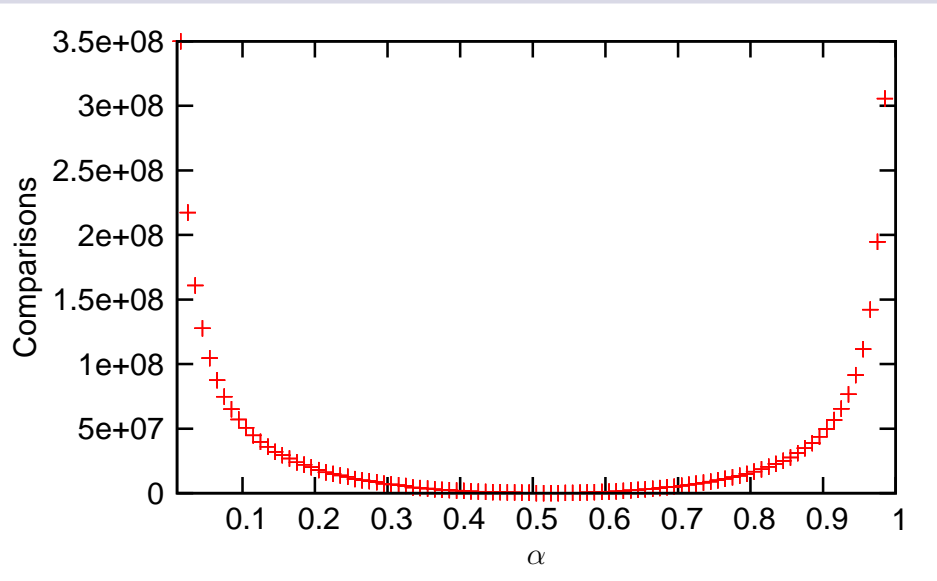
vEB achieves the fastest running time for $\alpha \approx .25$

Cache Faults for Blocked Layouts



vEB achieves the smallest number of cache faults

Experimental Summary



Conclusion

Skewed binary search trees

beat

Perfectly balanced binary search trees

because

The costs going left and right are different !