# Content

- **1. Why Advanced Data Structure?**
  - 1.1. Data Structure and Algorithm
  - 1.2. Data Structure and Data-intensive Application
- **2. Principles of Data Structure**
  - **2.1. Why and How O(log n)-access Time?**
  - **2.2. Amortized/Competitive Analysis and Dynamic Data Structure**
  - 2.3. Randomized Data Structure
  - 2.4. Augmented Data Structure
  - 2.5. Compact Data Structure
  - 2.6. Distributed Data Structure
- **3. Application**
  - 3.1. Data Streams/XML/RDF Storage Structure
  - 3.2. P2P Overlay Structure
  - 3.3. Indexing Moving Objects
  - 3.4. What's More?

# Why and How O(log n) Access Time?

Zhihong Chong

http://cse.seu.edu.cn/people/zhchong/

# Outline

1. First try: $O(n^{(3/2)})$ sorting algorithm
2. From binary search to binary search tree; from list to skip list
3. From binary heap to binomial heap
4. From AVL Trees to B-trees, red-black tree……

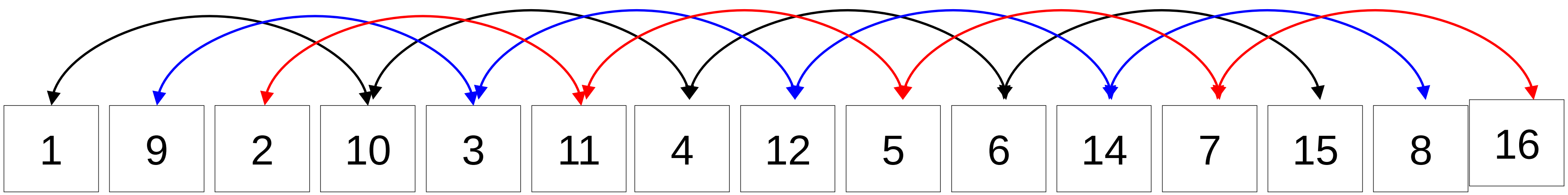
# 排序方法的一个下界

- 逆序
  - *一对逆序 $i<j, a[i]>a[j]$*
    - ***34，8，64，5***
    - *（34，8），（34，5），（8，5）*
  - *一个事实：如果一个序列的逆序个数为零，那么这个序列是从小到大的序列*
  - *另外一个事实：交换两个相邻的逆序对仅仅使一个序列的逆序个数减少1*
- **定理1：随机序列的平均逆序对数为n(n-1)/4**
- **定理2：交换相邻逆序的算法平均需要**$\Omega(n^2)$

# First try: from $O(n^2)$ to $O(n^{1.5})$

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 35 | 80 | 75 | 34 | 45 | 62 | 57 | 55 | 90 | 85 | 60 | 90 | 70 | 75 | 65 |

| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | 34 | 45 | 60 | 35 | 55 | 62 | 57 | 60 | 65 | 70 | 75 | 75 | 85 | 80 | 90 |




Insertion sort of 16/3 ~= 5 items, items are = 1, 10, 4, 6, 15

Insertion sort of 16/3 ~= 5 items, items are = 9, 3, 12, 14, 8

Insertion sort of 16/3 ~= 5 items

# First try: from $O(n^2)$ to $O(n^{1.5})$

| | | |
|---|---|---|
| Original | 32 95 16 82 24 66 35 19 75 54 40 43 93 68 | |
| After 5-sort | 32 35 16 68 24 40 43 19 75 54 66 95 93 82 | 6 swaps |
| After 3-sort | 32 19 16 43 24 40 54 35 75 68 66 95 93 82 | 5 swaps |
| After 1-sort | 16 19 24 32 35 40 43 54 66 68 75 82 93 95 | 15 swaps |

- Voi

```
•       int j, i;
•       int gap;
•       for (gap = a.size() /2; gap > 0; gap /=2)
•       {
•               for (i=gap; i < a.size(); i++)
•               {
•                       int tmp = a[i];
•                       for (j=i; j>=gap && tmp < a[j-gap]; j -= gap)
•                               a[j] = a[j-gap];
•                       a[j] = tmp;
•               }
•       }
• }
```

# An $O(n^2)$ sequence

- gap=n/2, n/4, ...
  - Insertion sort for each gap
  - For each gap, $(n/gap)^2 \times gap$ comparisons
  - Total cost is $n^2(\sum 1/gap) = O(n^2)$

# An $O(n^{1.5})$ sequence

1. $h_1=1, h_2=3, h_3=7, \ldots h_k=2^k-1$, let gap=$h_k$, because Hibbard suggested the sequence.

2. **For $h_k>N^{1/2}$,** we follow the previous analysis

3. **For $h_k<=N^{1/2}$**

- $h_k$-sorted ,
- For gap=$h_{k,}$ preserve $h_{k+1}$-sorted, $h_{k+2}$-sorted
- a[p-i]<a[p] if i is a multiple of $h_{k+1}$ or $h_{k+2}$,
- For any number $i>=(h_{k+1}-1)(h_{k+2}-1)=8h_k^2+4h_k$ can be expressed as a linear combination of $h_{k+1}$ and $h_{k+2}$

# An Example

- **$h_1$=1, $h_2$=3,$h_3$=7,$h_4$=15….**
- **k=3, $h_3$=3, ($h_4$-sorted and $h_5$-sorted)**
- **($h_4$-1)*($h_5$-1)=52=1*7+3*15**
- **A[100]<=A[107]<=A[122]<=A[137]<=A[152]**

# $O(n^{(3/2)})$的由来

$$\sum_{i=1}^{t} N^2/h_i = O(N^2 \sum_{i=1}^{t} 1/h_i) = O(N^2)$$

$$\boxed{h_k * (N/h_k)^2}$$

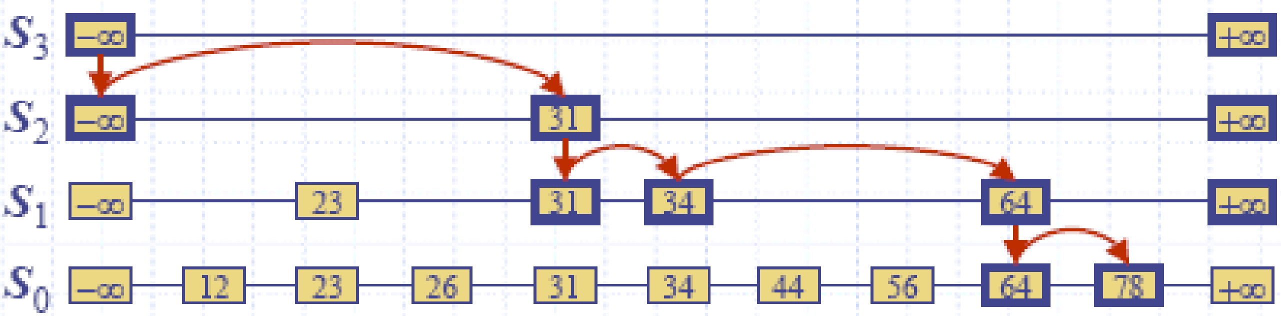
since $\sum_{i=1}^{t} 1/h_i < 2$

$$\boxed{h_k * (\,(h_{k+1}-1)\ (h_{k+2}-1)\ /h_k)^2}$$

# From sorted list to link list, skip list

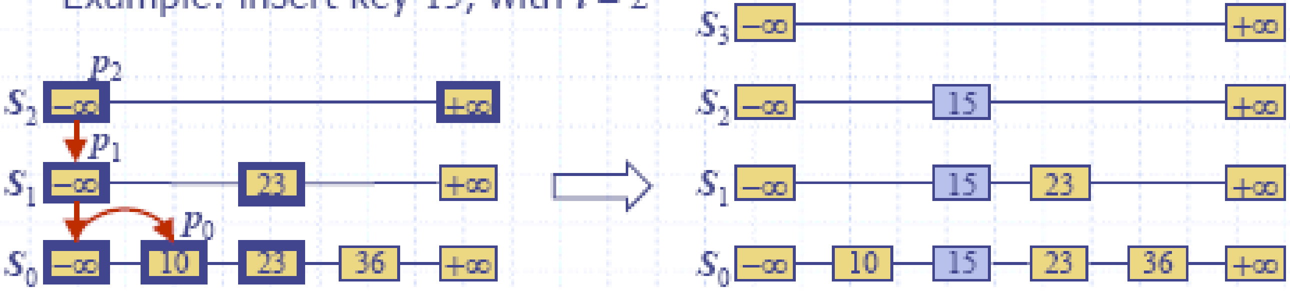
## Search

- We search for a key $x$ in a a skip list as follows:
  - We start at the first position of the top list
  - At the current position $p$, we compare $x$ with $y \leftarrow key(after(p))$
    - $x = y$: we return $element(after(p))$
    - $x > y$: we "scan forward"
    - $x < y$: we "drop down"
  - If we try to drop down past the bottom list, we return $NO\_SUCH\_KEY$
- Example: search for 78



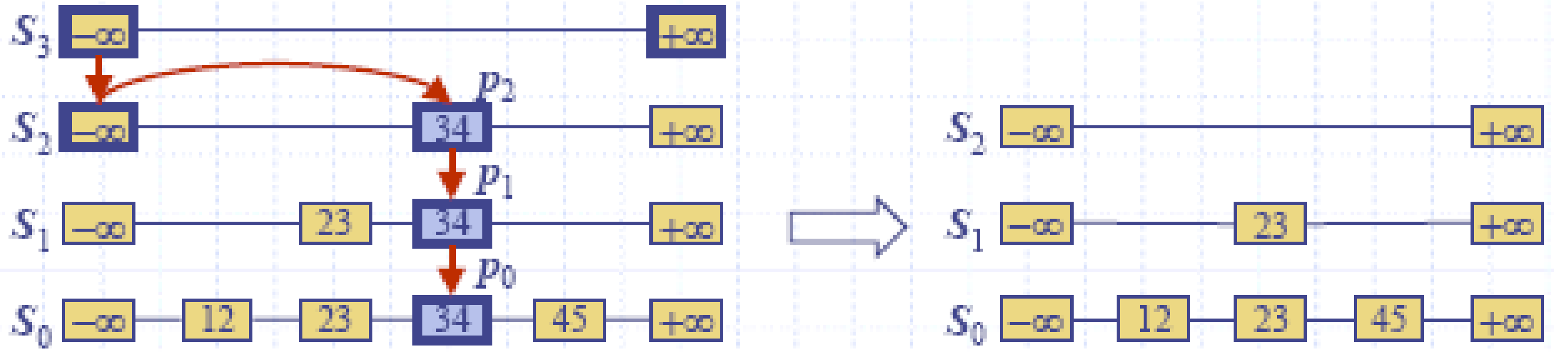
# Insertion

- To insert an item $(x, o)$ into a skip list, we use a randomized algorithm:
  - We repeatedly toss a coin until we get tails, and we denote with $i$ the number of times the coin came up heads
  - If $i \geq h$, we add to the skip list new lists $S_{h+1}, \dots, S_{i+1}$, each containing only the two special keys
  - We search for $x$ in the skip list and find the positions $p_0, p_1, \dots, p_i$ of the items with largest key less than $x$ in each list $S_0, S_1, \dots, S_i$
  - For $j \leftarrow 0, \dots, i$, we insert item $(x, o)$ into list $S_j$ after position $p_j$
- Example: insert key 15, with $i = 2$



# Deletion

- To remove an item with key $x$ from a skip list, we proceed as follows:
  - We search for $x$ in the skip list and find the positions $p_0, p_1, \ldots, p_i$ of the items with key $x$, where position $p_j$ is in list $S_j$
  - We remove positions $p_0, p_1, \ldots, p_i$ from the lists $S_0, S_1, \ldots, S_i$
  - We remove all but one list containing only the two special keys
- Example: remove key 34



# Space Usage

- The space used by a skip list depends on the random bits used by each invocation of the insertion algorithm
- We use the following two basic probabilistic facts:

  Fact 1: The probability of getting $i$ consecutive heads when flipping a coin is $1/2^i$

  Fact 2: If each of $n$ items is present in a set with probability $p$, the expected size of the set is $np$

- Consider a skip list with $n$ items
  - By Fact 1, we insert an item in list $S_i$ with probability $1/2^i$
  - By Fact 2, the expected size of list $S_i$ is $n/2^i$
- The expected number of nodes used by the skip list is

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

- Thus, the expected space usage of a skip list with $n$ items is $O(n)$

# Height

- The running time of the search an insertion algorithms is affected by the height $h$ of the skip list
- We show that with high probability, a skip list with $n$ items has height $O(\log n)$
- We use the following additional probabilistic fact:

  Fact 3: If each of $n$ events has probability $p$, the probability that at least one event occurs is at most $np$

- Consider a skip list with $n$ items
  - By Fact 1, we insert an item in list $S_i$ with probability $1/2^i$
  - By Fact 3, the probability that list $S_i$ has at least one item is at most $n/2^i$
- By picking $i = 3\log n$, we have that the probability that $S_{3\log n}$ has at least one item is at most

$$n/2^{3\log n} = n/n^3 = 1/n^2$$

- Thus a skip list with $n$ items has height at most $3\log n$ with probability at least $1 - 1/n^2$

# Search and Update Times

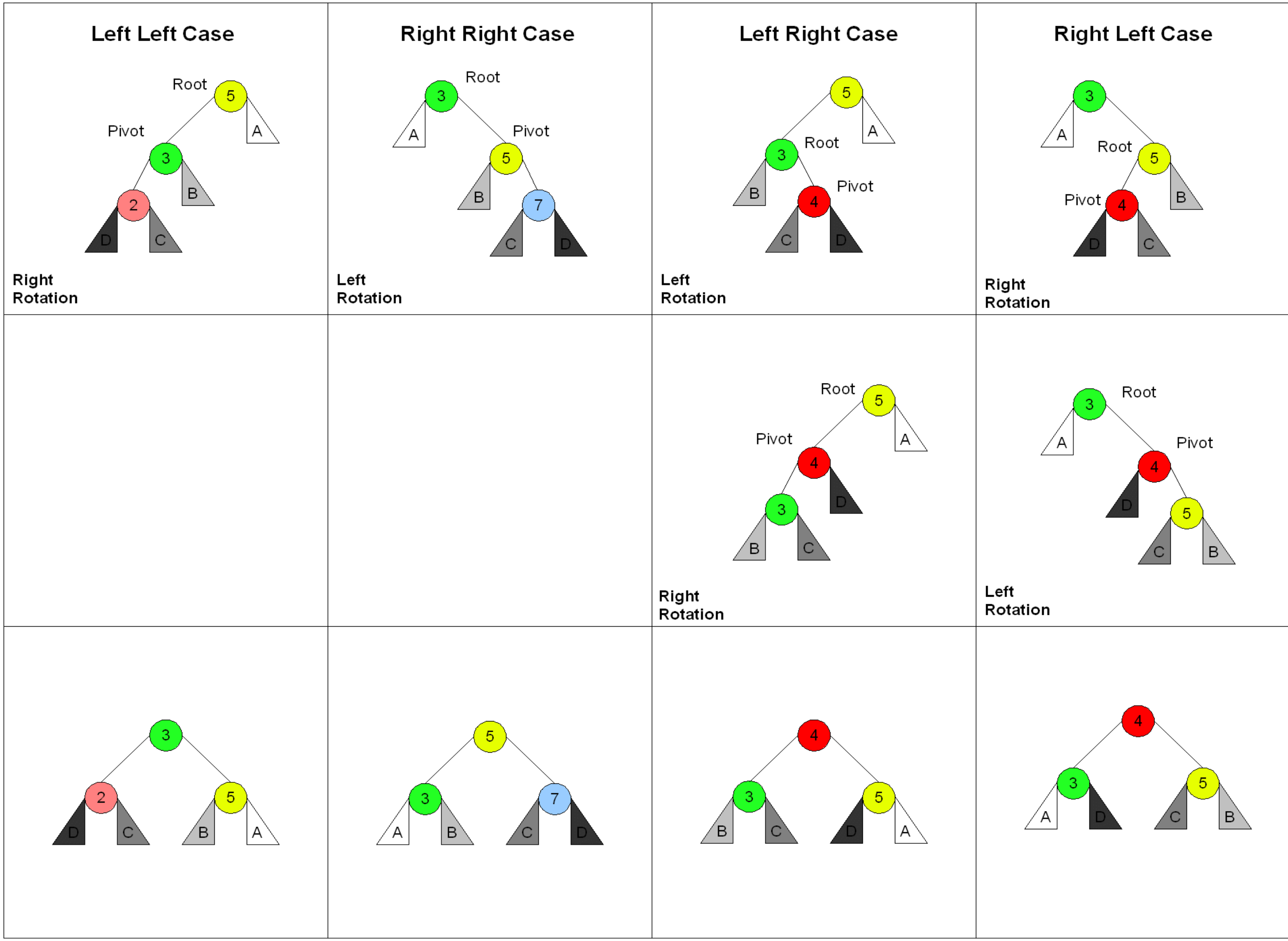
- The search time in a skip list is proportional to
  - the number of drop-down steps, plus
  - the number of scan-forward steps
- The drop-down steps are bounded by the height of the skip list and thus are $O(\log n)$ with high probability
- To analyze the scan-forward steps, we use yet another probabilistic fact:

  Fact 4: The expected number of coin tosses required in order to get tails is 2

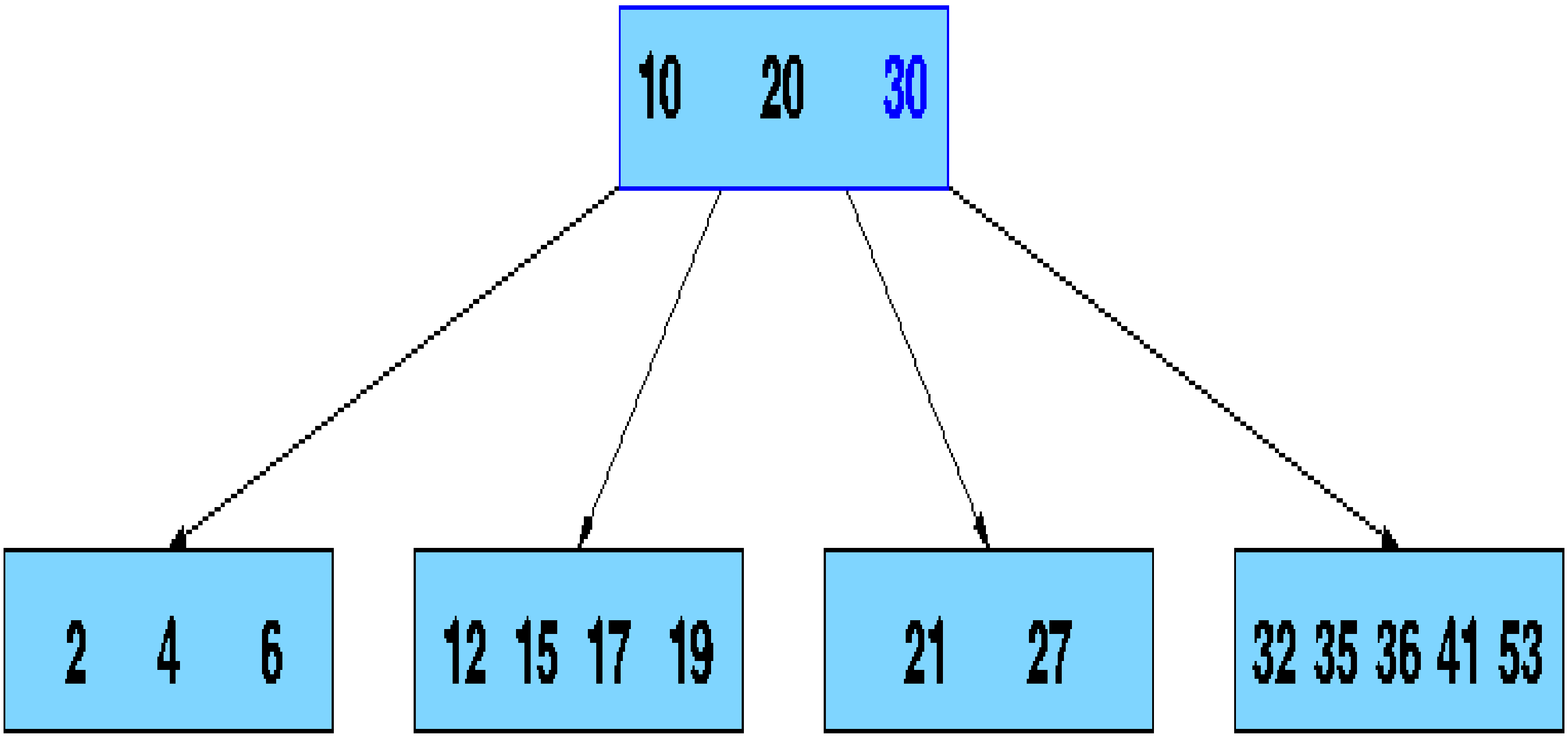
- When we scan forward in a list, the destination key does not belong to a higher list
  - A scan-forward step is associated with a former coin toss that gave tails
- By Fact 4, in each list the expected number of scan-forward steps is 2
- Thus, the expected number of scan-forward steps is $O(\log n)$
- We conclude that a search in a skip list takes $O(\log n)$ expected time
- The analysis of insertion and deletion gives similar results

There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

**Root** is the initial parent before a rotation and **Pivot** is the child to take the root's place.



B-Tree: Minimization Factor t = 3, Minimum Degree = 2, Maximum Degree = 5



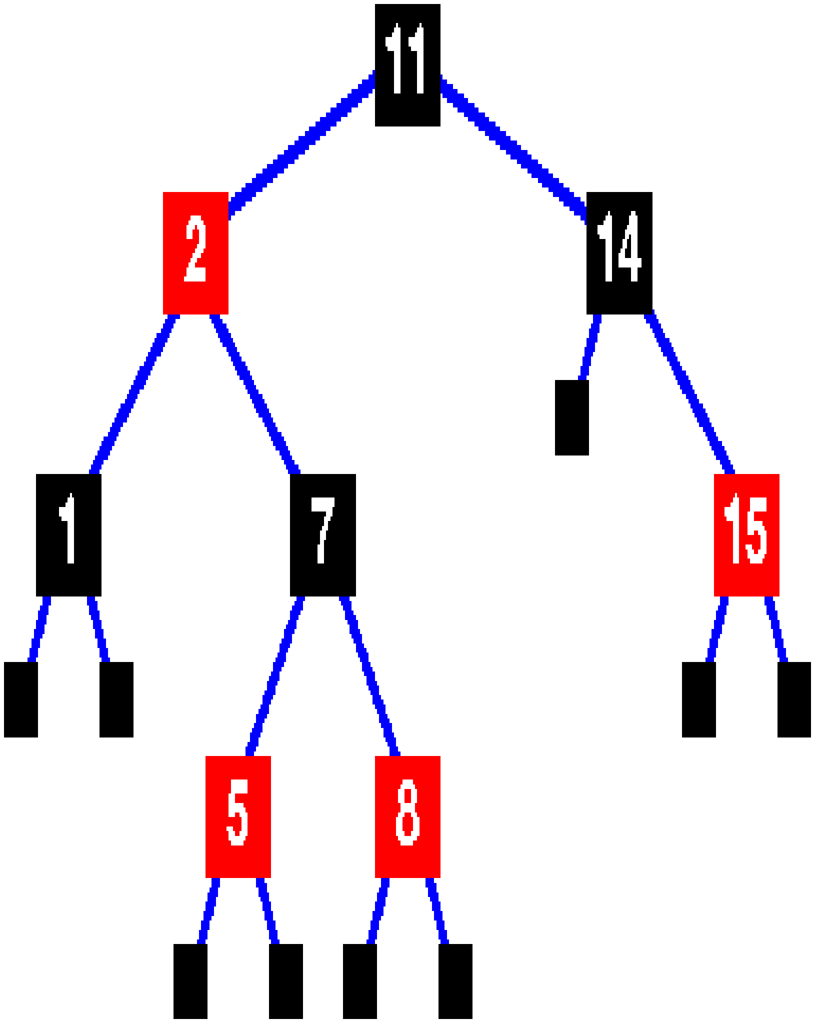

Search(21)

# From AVL Trees to B-trees

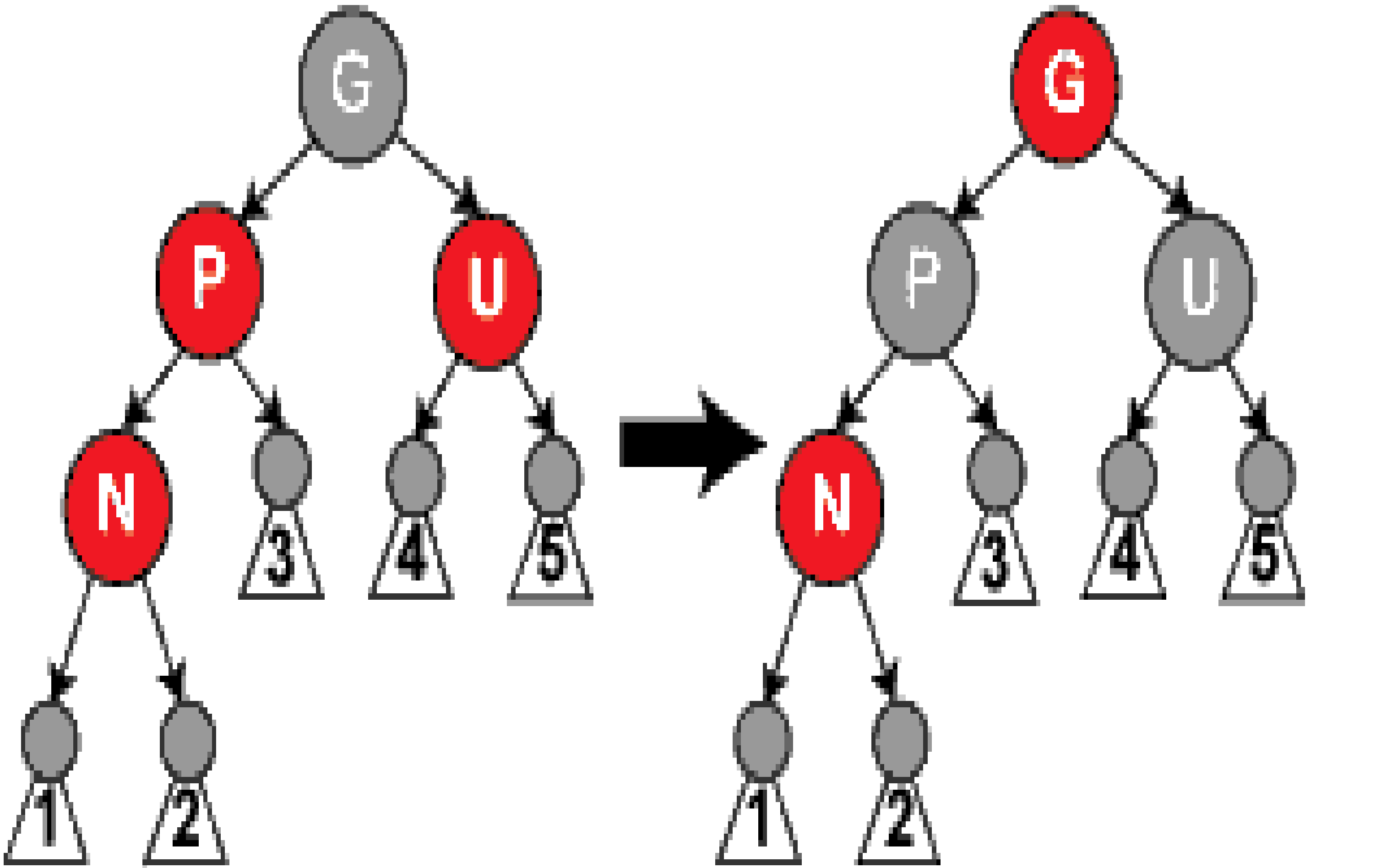
1. The data items are sorted at leaves
2. The nonleaf nodes stores up to M-1 key to guide the searching
3. The root is either a leaf or has between two and M children
4. All nonleaf nodes except the root have between $\lceil M/2 \rceil$ and M children
5. All leaves are at the same depth and have between L/2 and L data items

# And Red Black-tree

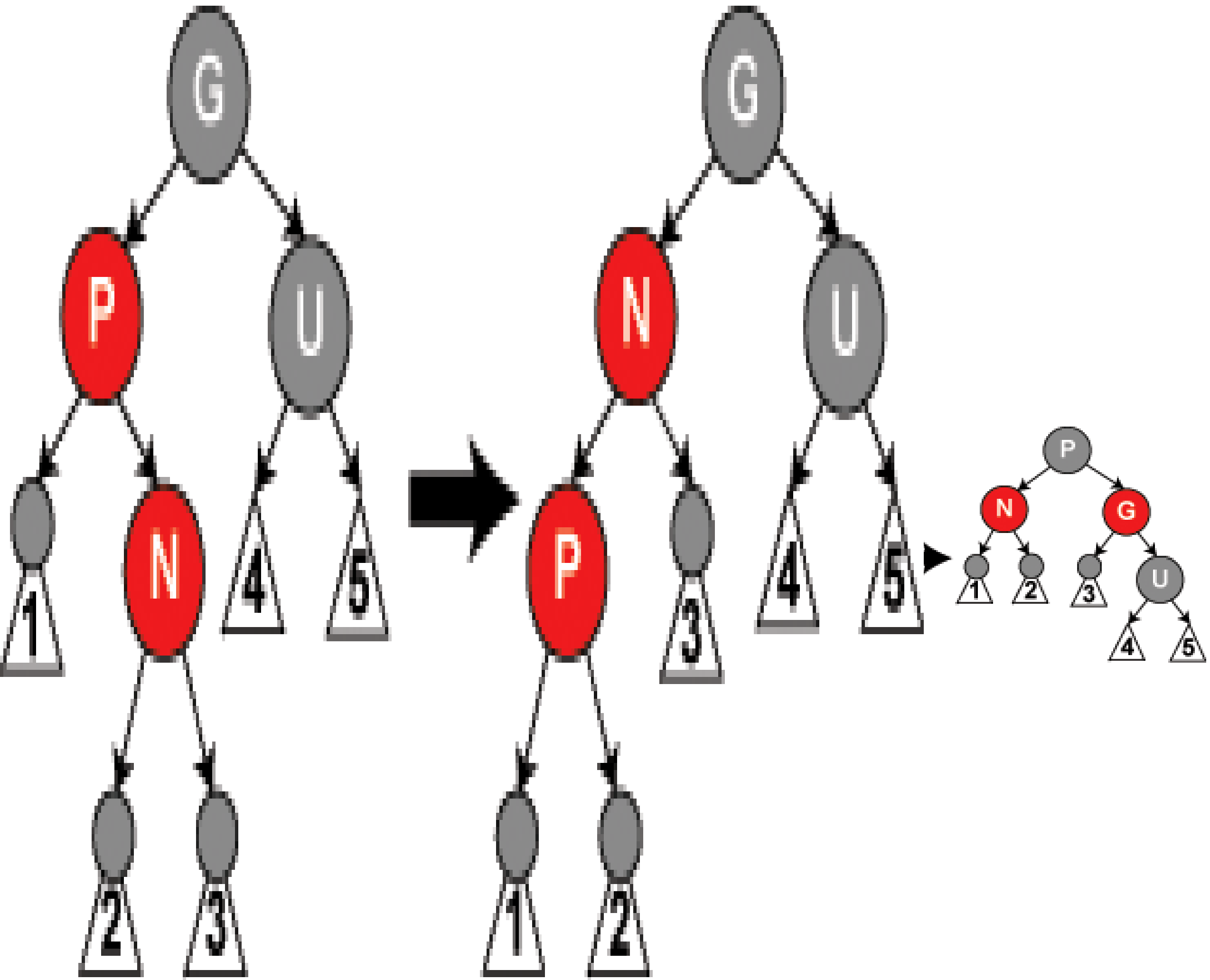
1. A node is either red or black.
2. The root is black
3. All leaves are black.
4. Both children of every red node are black.
5. Every simple path from a given node to any of its descendant leaves contains **the same number of black nodes.**



# Insertion

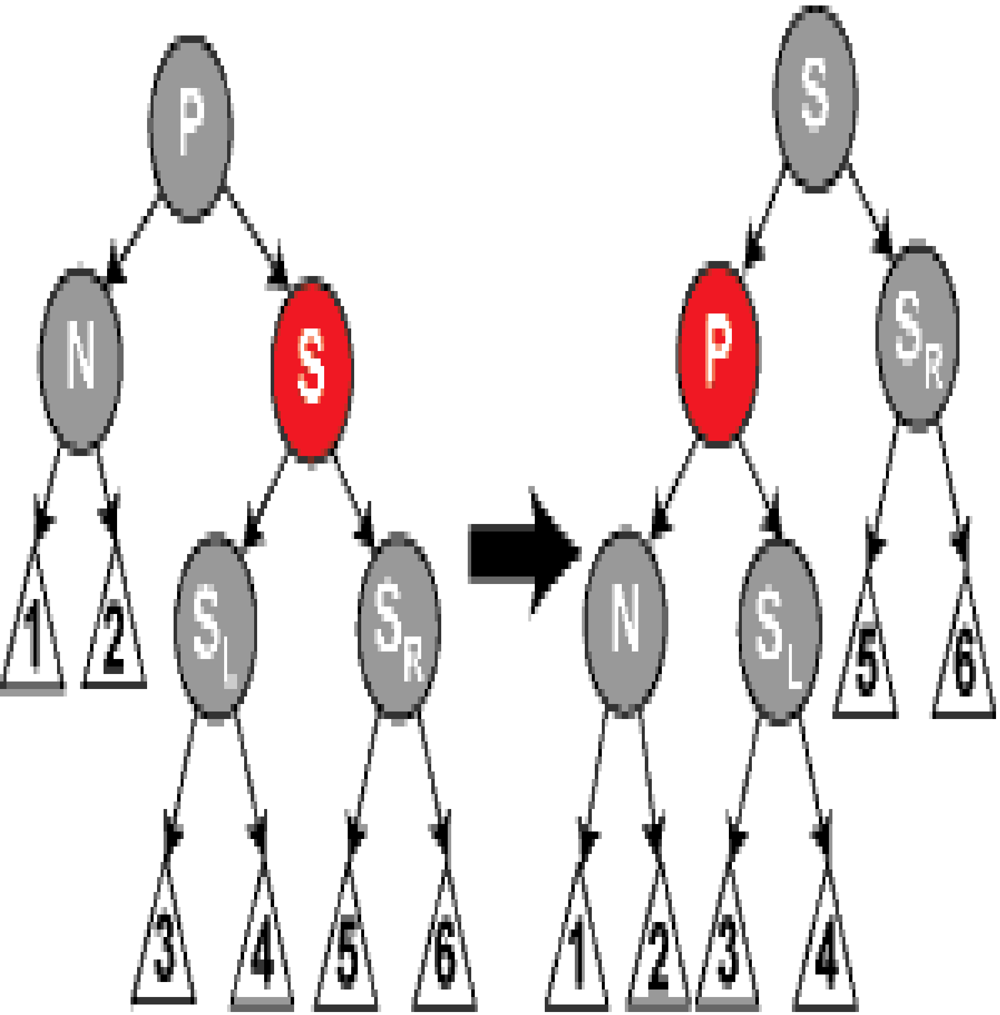


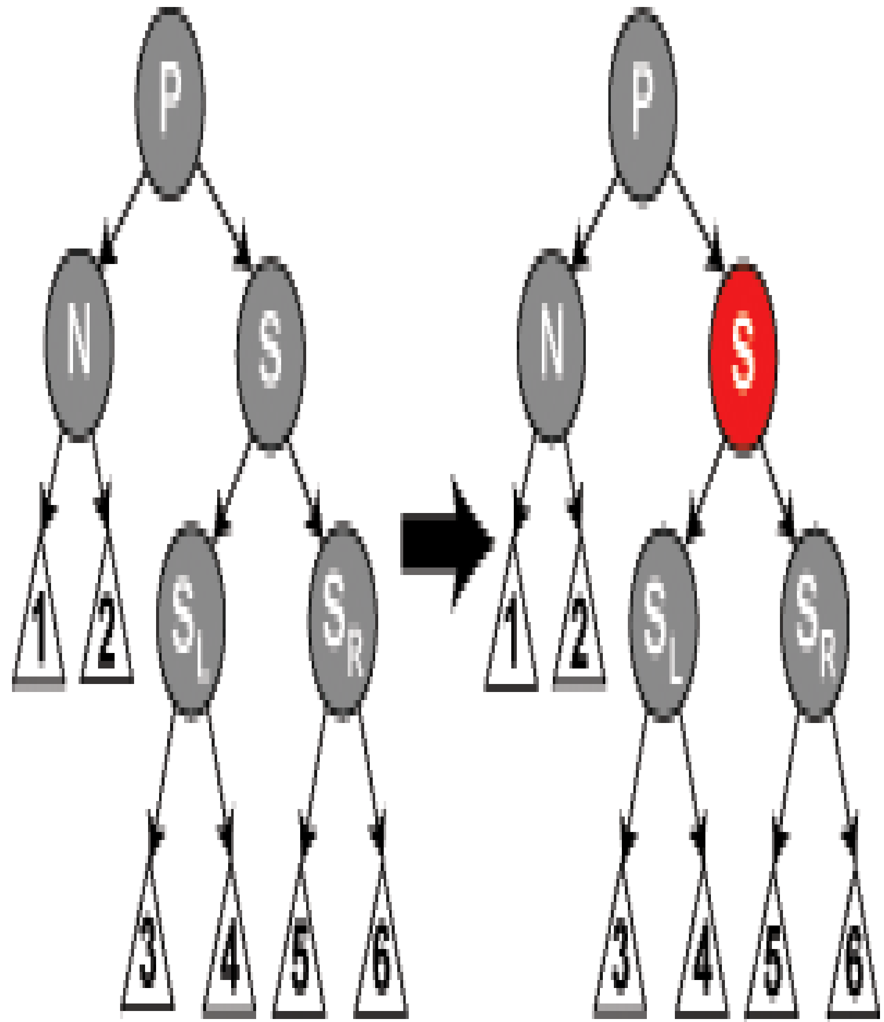
# Insertion

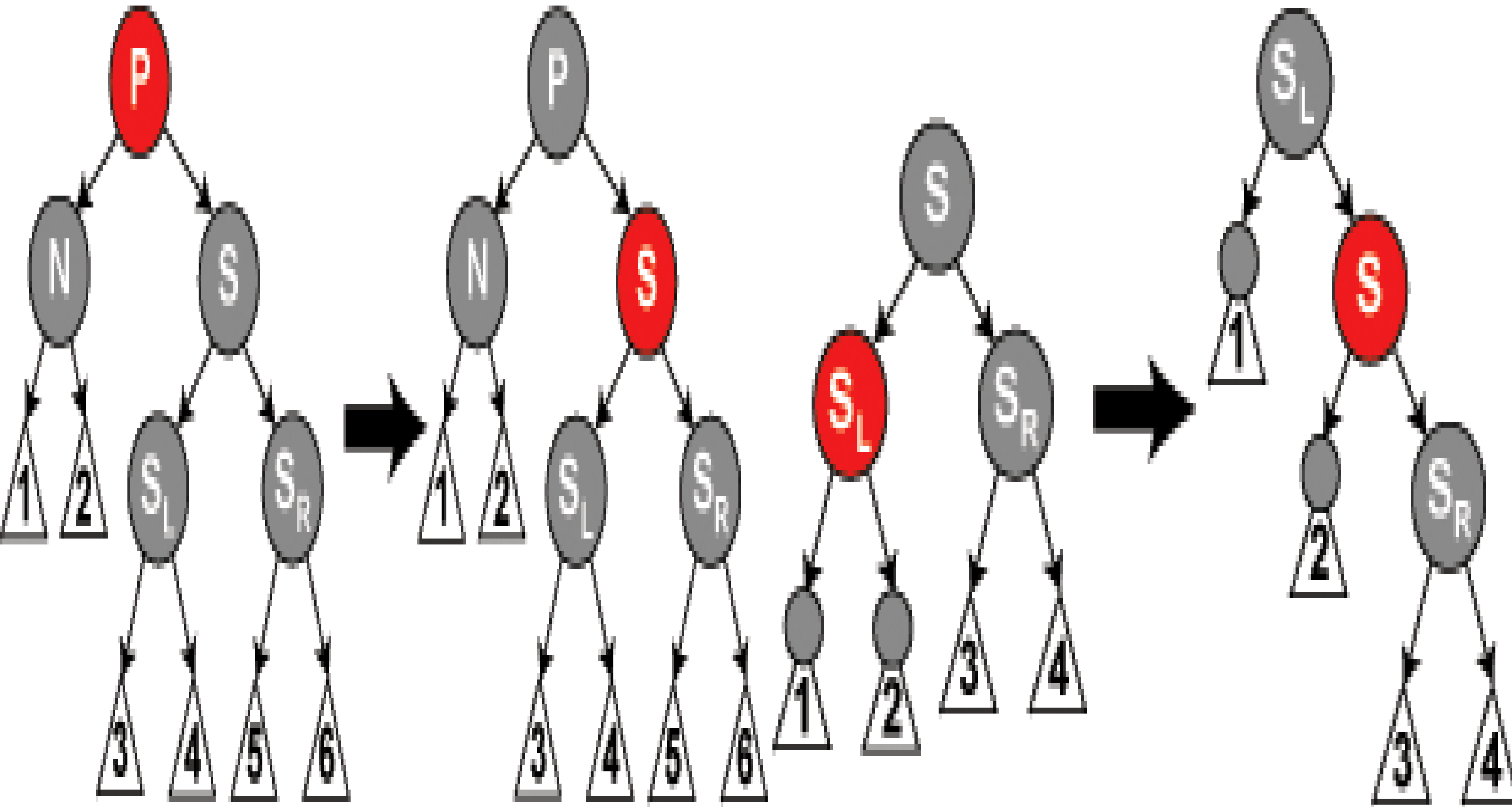


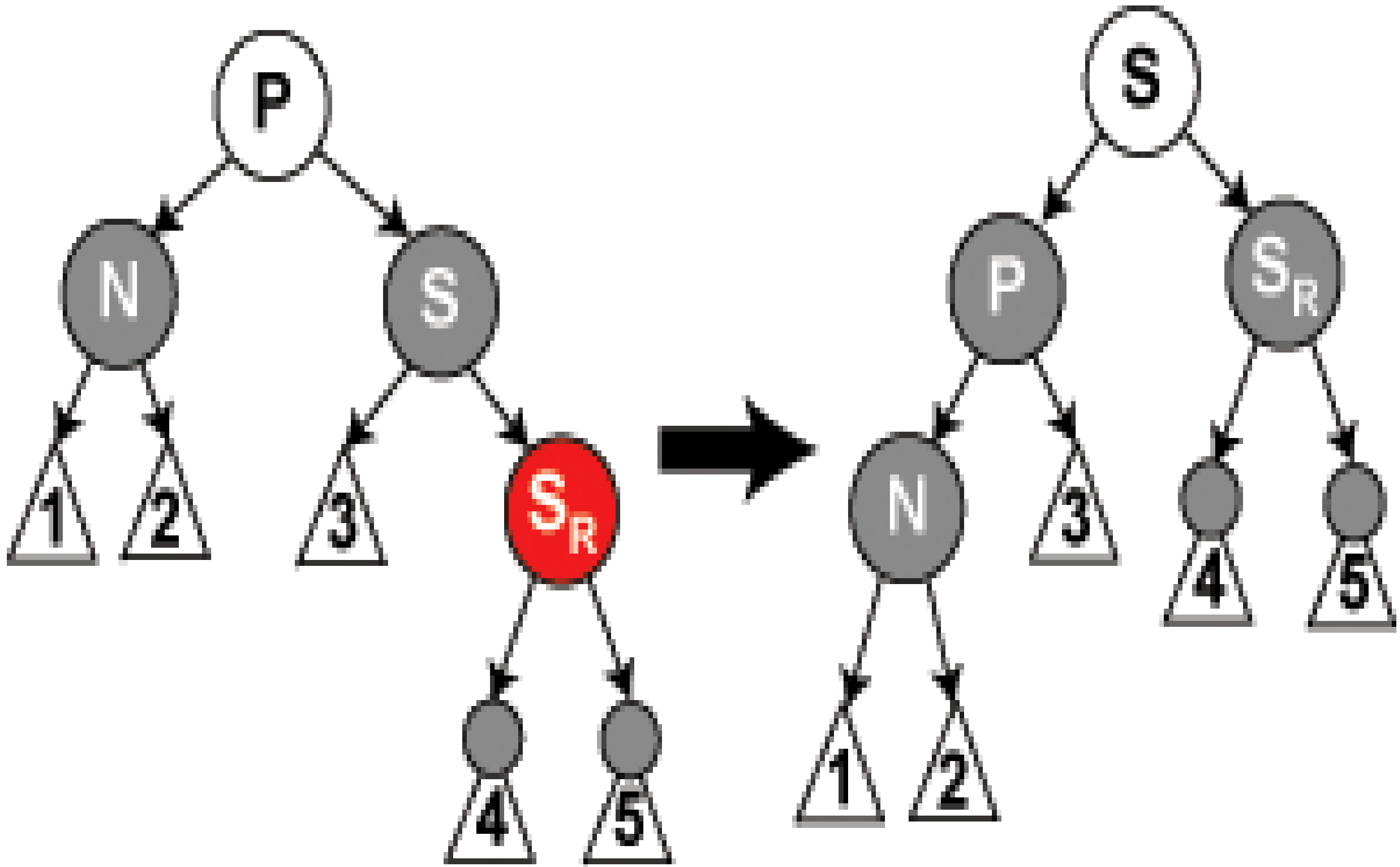
# Deletion

1. this reduces to the problem of deleting a node with at most one non-leaf child.
    - deleting a red node
    - the deleted node is black and its child is red
    - both the node to be deleted and its child are black

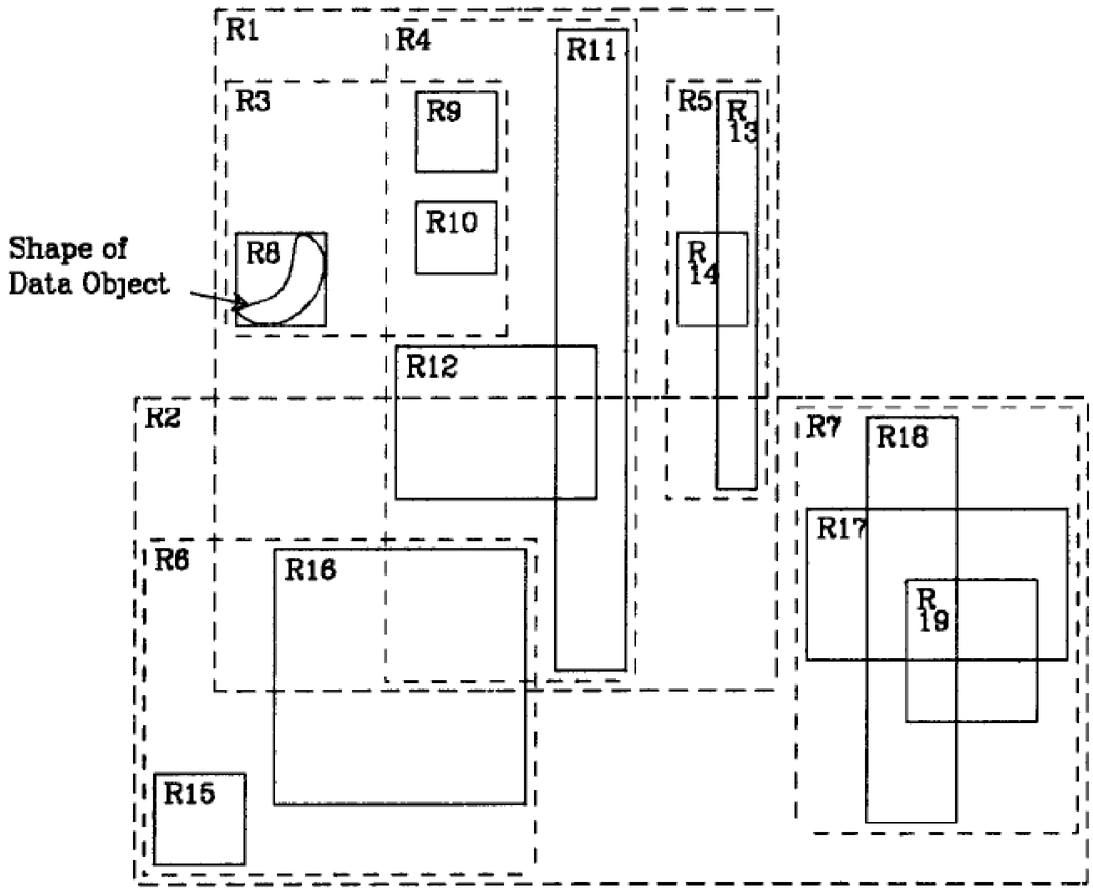

P
N
S
S
L
S
R
1
2
3
4
5
6
S
P
S
R
N
S
L
5
6
1
2
3
4


P
N
S
1
2
S
L
S
R
3
4
5
6
P
N
S
1
2
S
L
S
R
3
4
5
6

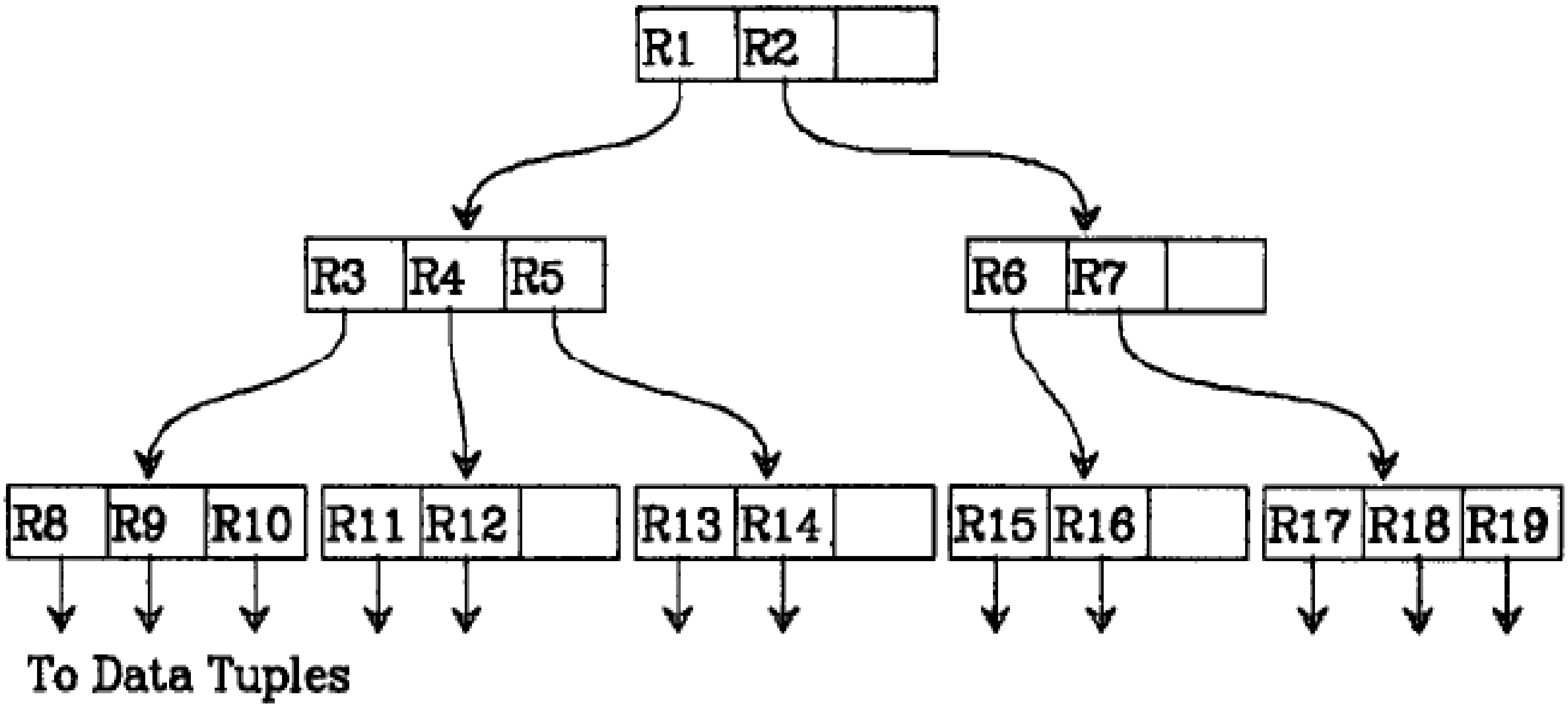

P
N
S
$S_L$
$S_R$
1
2
3
4
5
6


P
N
S
1
2
3
$S_R$
4
5
S
P
$S_R$
N
3
4
5
1
2

# And R-tree

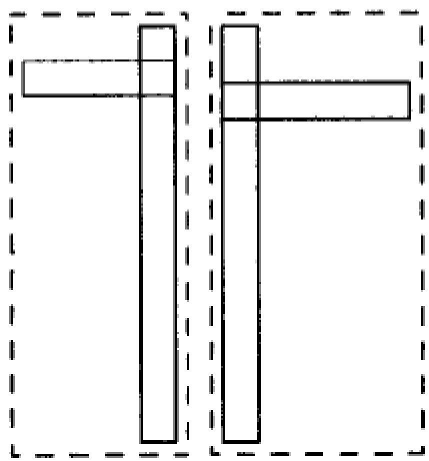


R1
R4
R11
R3
R9
R5
R
13
R10
Shape of
Data Object
R8
R
14
R12
R2
R7
R18
R17
R6
R16
R
19
R15

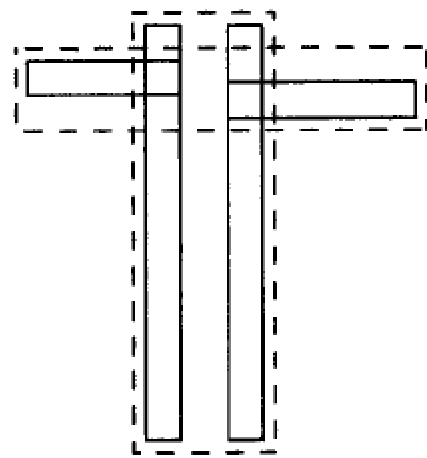



# R-TREES: A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING

Antonin Guttman
University of California
Berkeley



Bad split



Good split

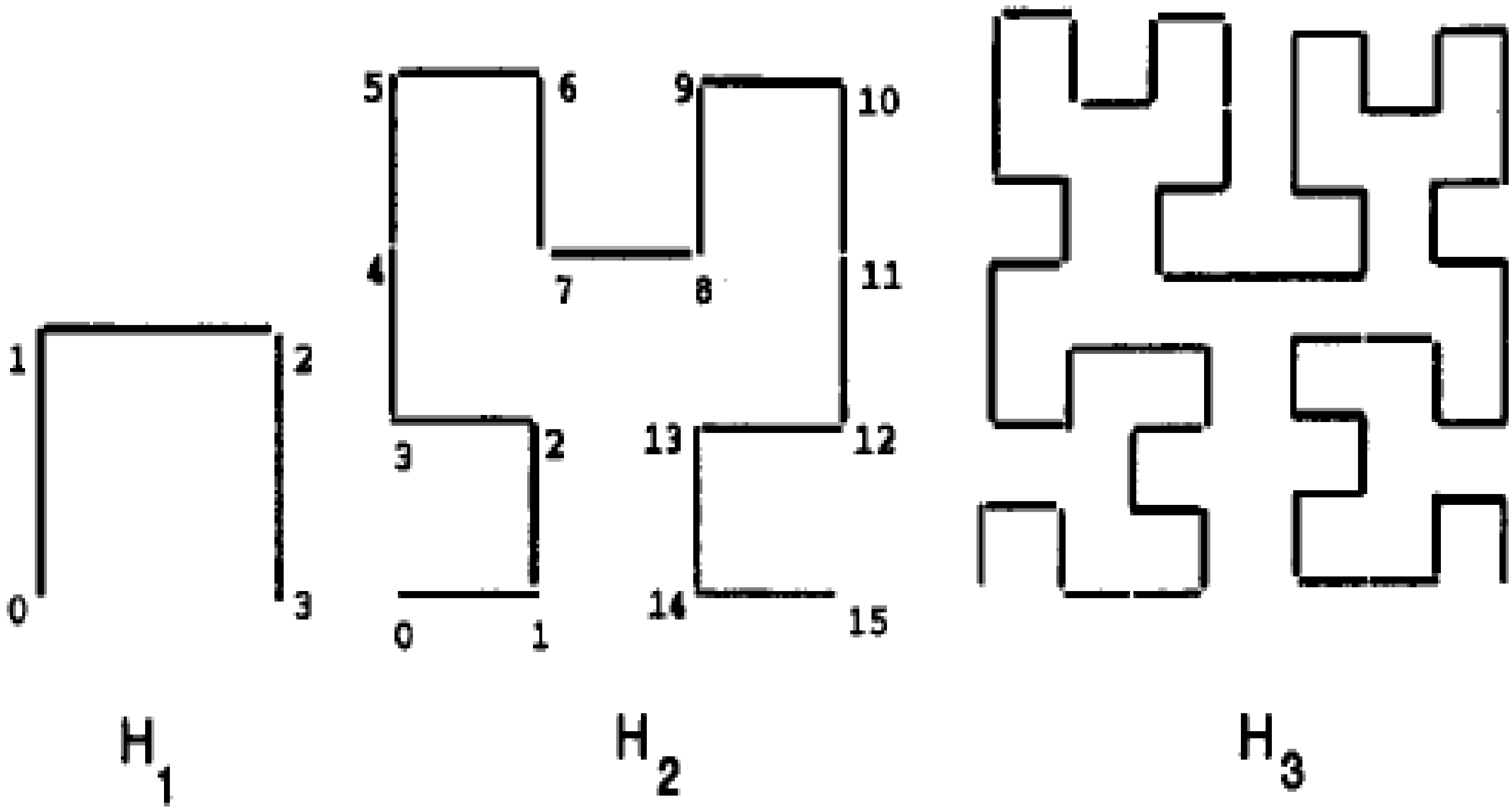
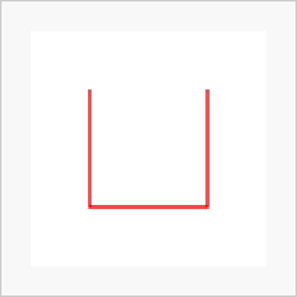
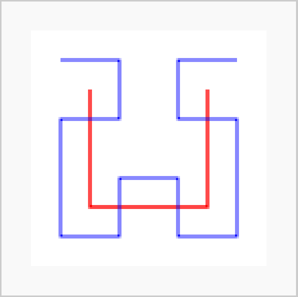
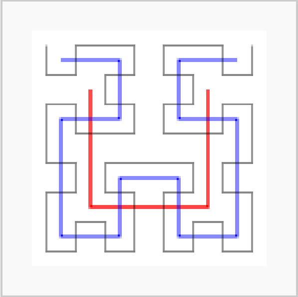
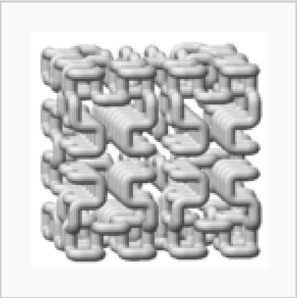


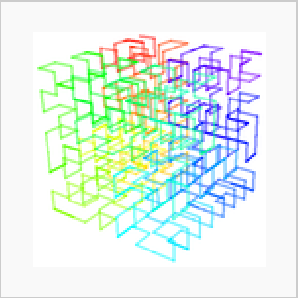

Figure 1: Hilbert Curves of order 1, 2 and 3

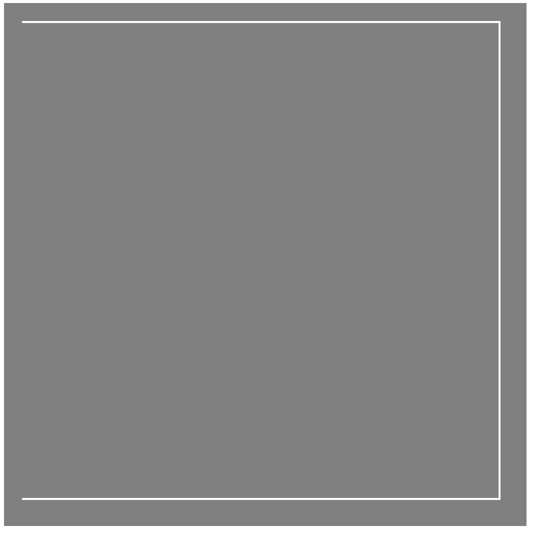

Hilbert curve, first order


Hilbert curves, first and second orders


Hilbert curves, first to third orders


Hilbert curve in three dimensions


3-D Hilbert curve with color showing progression



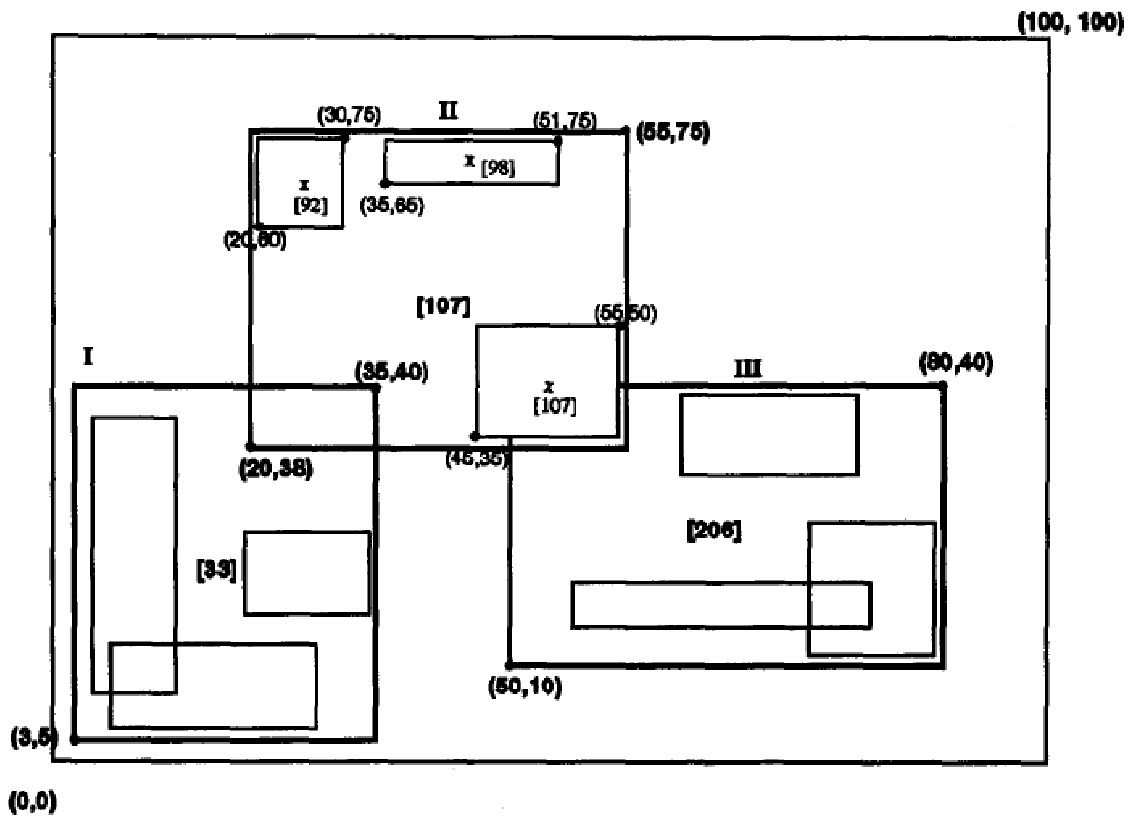



Figure 2: *Data rectangles organized in a Hilbert R-tree*

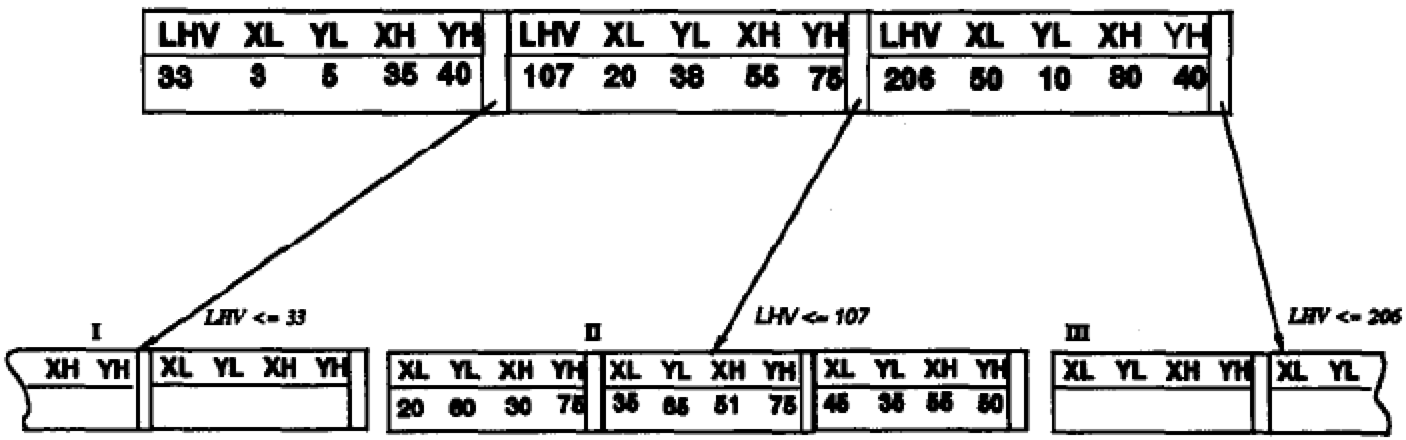
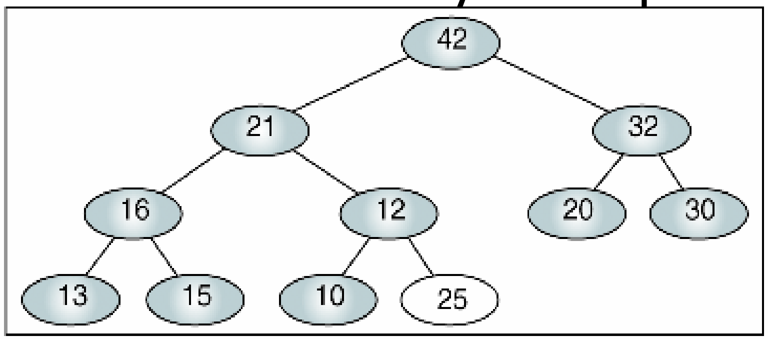


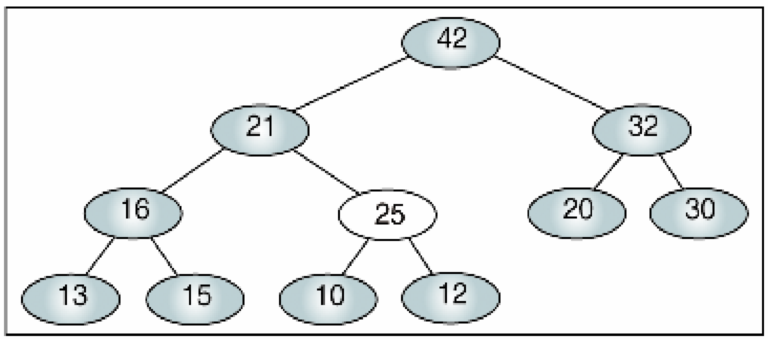

Figure 3: *The file structure for the previous Hilbert R-tree*

# From binary heap to binomial heap

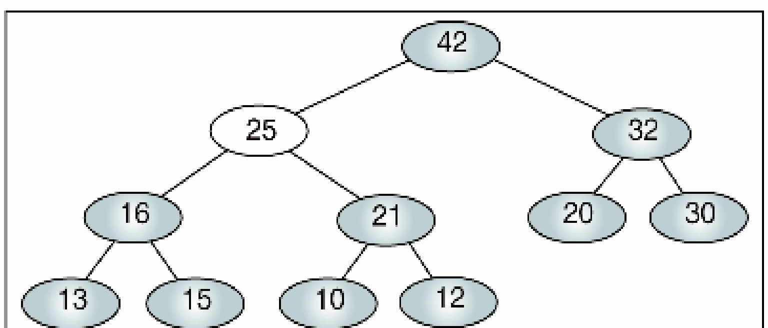
- Re-heap up




(a) Original tree: not a heap

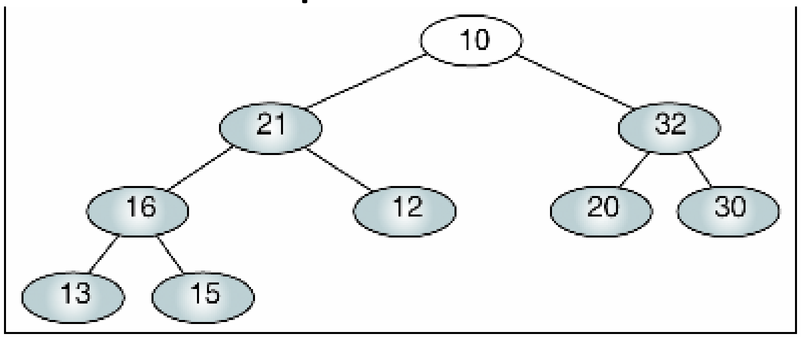



(b) Last element (25) moved up

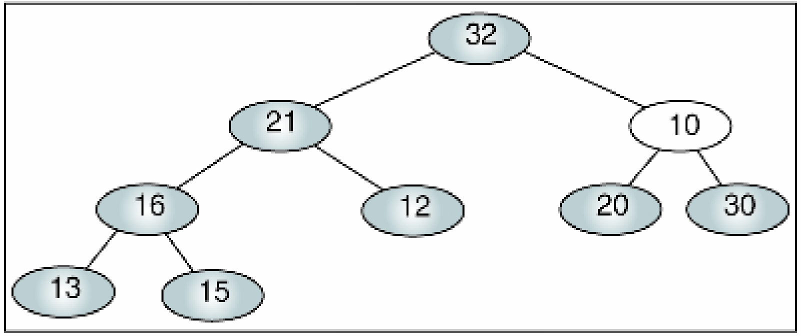



(c) Moved up again: tree is a heap

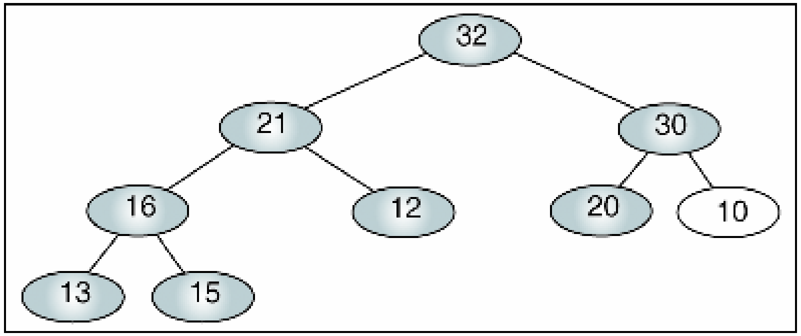
- Re-heap down



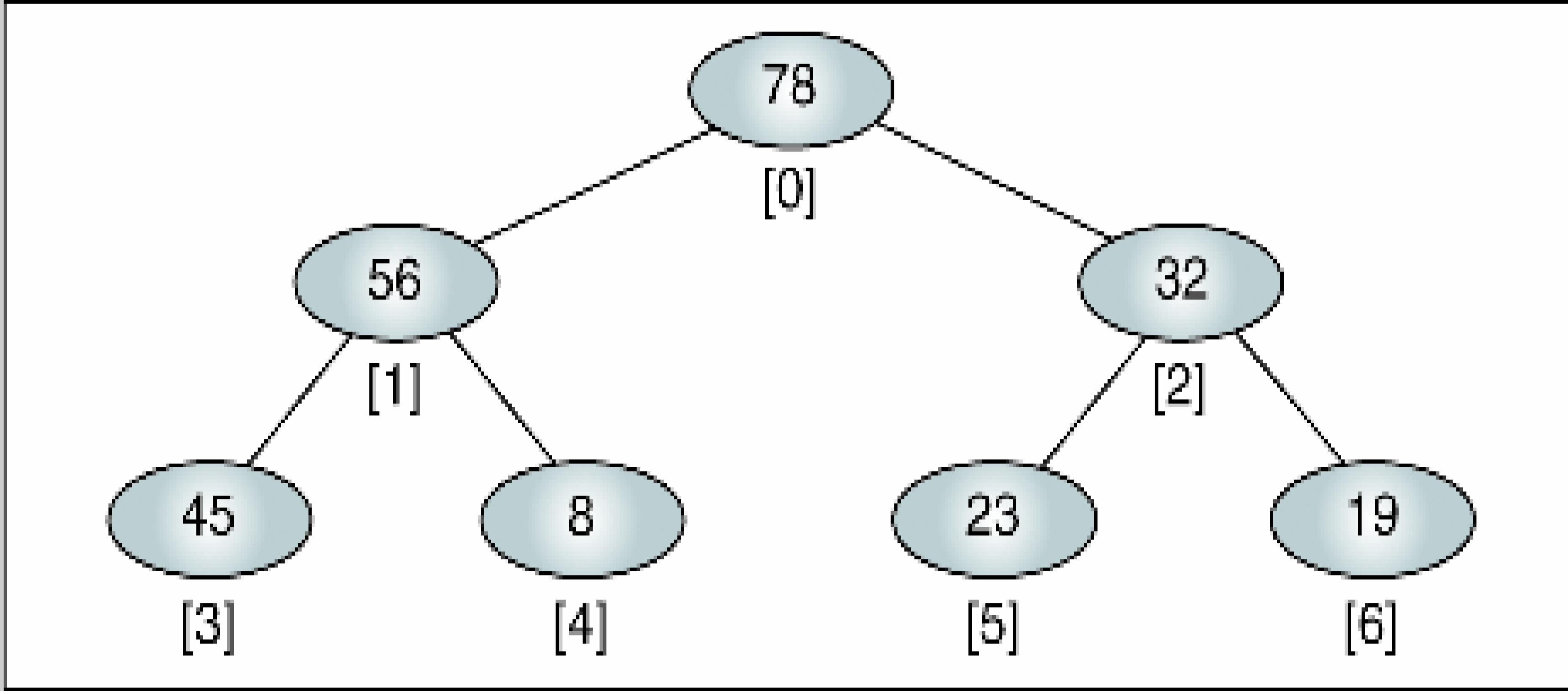

(a) Original tree: not a heap



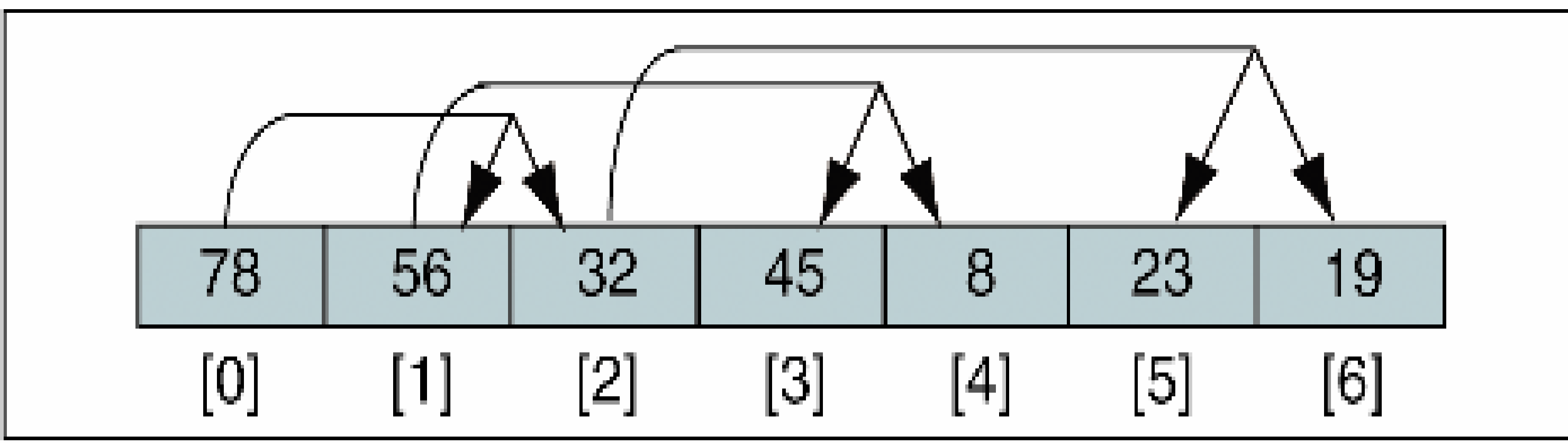

(b) Root moved down (right)




(c) Moved down again: tree is a heap



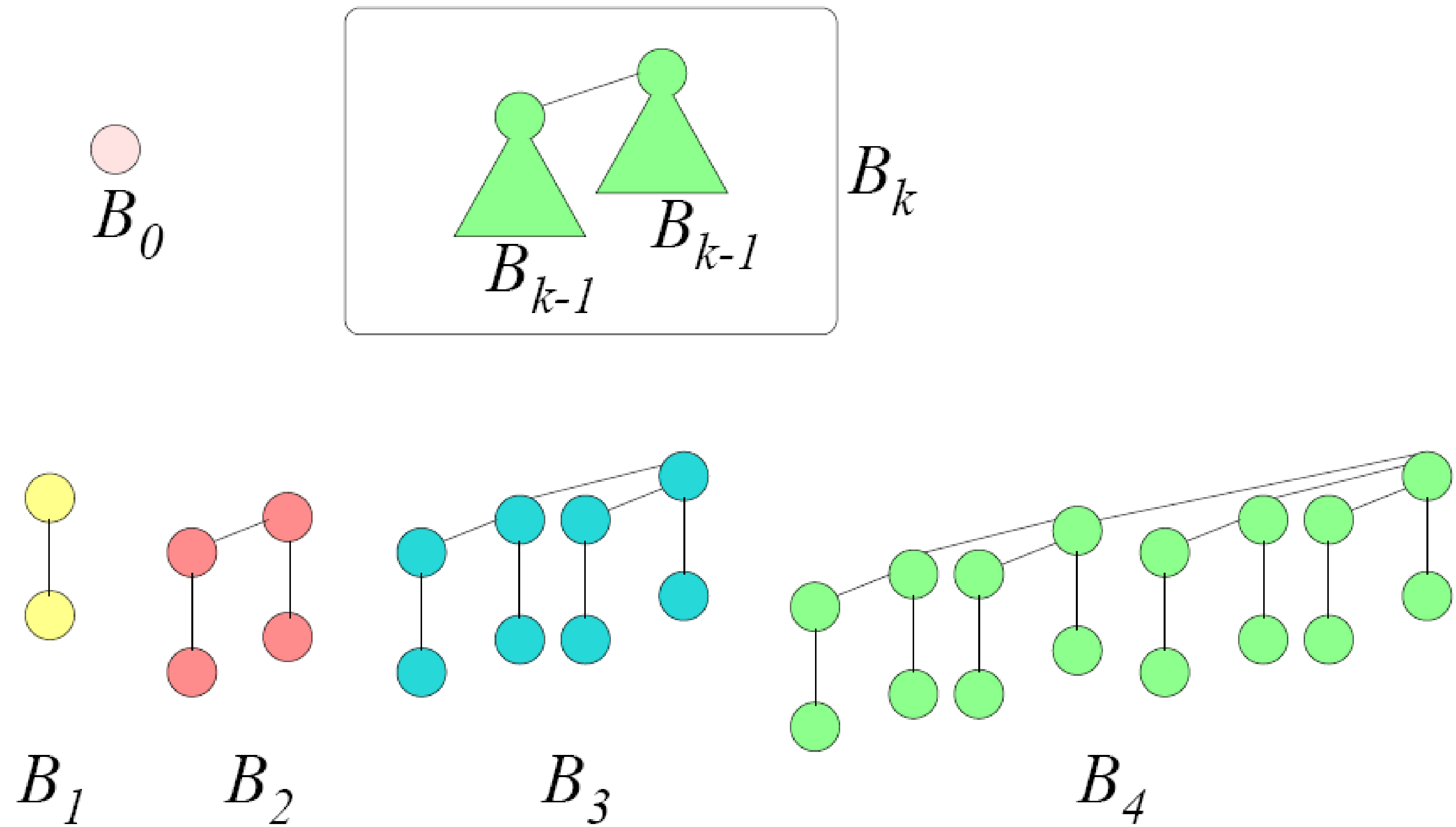

(a) Heap in its logical form




(b) Heap in an array

A binomial tree $B_k$ is an ordered tree defined recursively.

- $B_0$ consists of a single node.
- For $k \geq 1$, $B_k$ is a pair of $B_{k-1}$ trees, where the root of one $B_{k-1}$ becomes the leftmost child of the other.

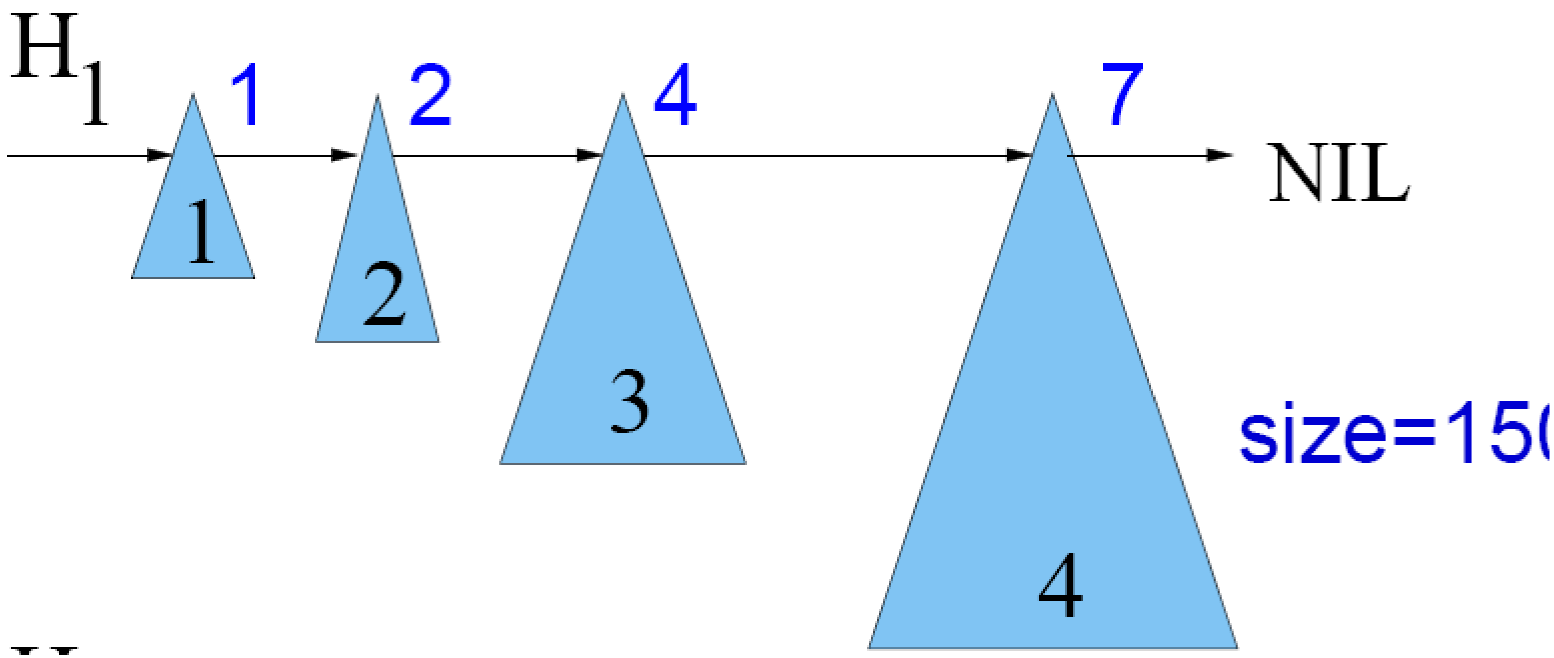


$B_0$
$B_{k-1}$
$B_{k-1}$
$B_k$
$B_1$
$B_2$
$B_3$
$B_4$

**Lemma A** For all integers $k \geq 0$, the following properties hold:

1. $B_k$ has $2^k$ nodes.
2. $B_k$ has height $k$.
3. For $i = 0, \ldots, k$, $B_k$ has exactly $\binom{k}{i}$ nodes at depth $i$.
4. The root of $B_k$ has degree $k$ and all other nodes in $B_k$ have degree smaller than $k$.
5. If $k \geq 1$, then the children of the root of $B_k$ are $B_{k-1}, B_{k-2}, \cdots, B_0$ from left to right.

**Corollary B** The maximum degree of an $n$-node binomial tree is $\lg n$.


H1
1
2
4
7
NIL
1
2
3
4
size=15(

## Implementation of a Binomial Heap

Keep at each node the following pieces of information:

- a field $key$ for its key,
- a field $degree$ for the number of children,
- a pointer $child$, which points to the leftmost-child,
- a pointer $sibling$, which points to the right-sibling, and
- a pointer $p$, which points to the parent.

The roots of the trees are connected so that the sizes of the connected trees are in decreasing order. Also, for a heap $H$, $head[H]$ points to the head of the list.

- How count the cost of n insertions?
    - Amortized analysis(分摊代价)

# An O()