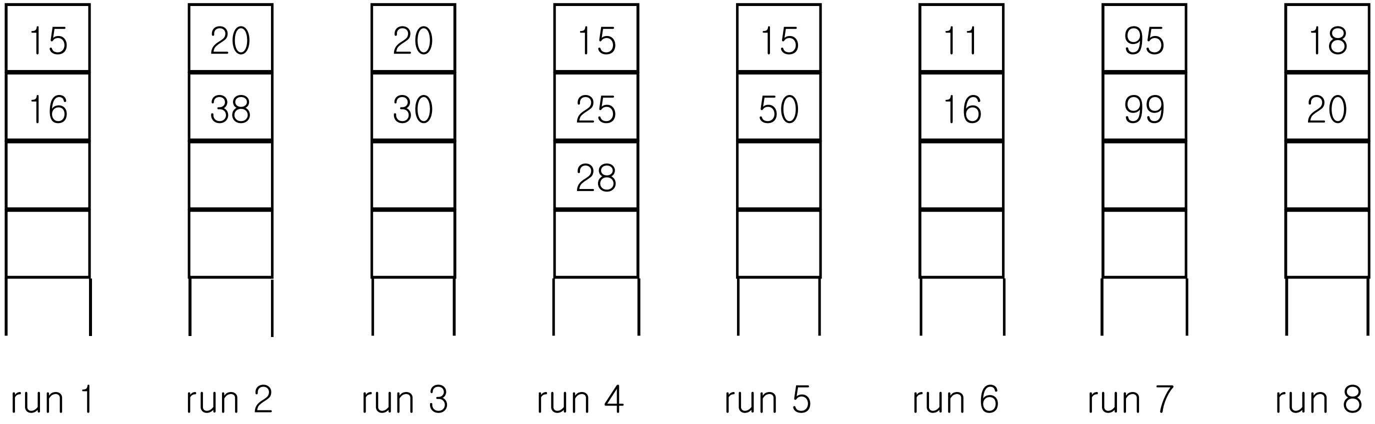
# Selection Trees

Prof. Ki-Hoon Lee
Dept. of Computer Engineering
Kwangwoon University

# Introduction

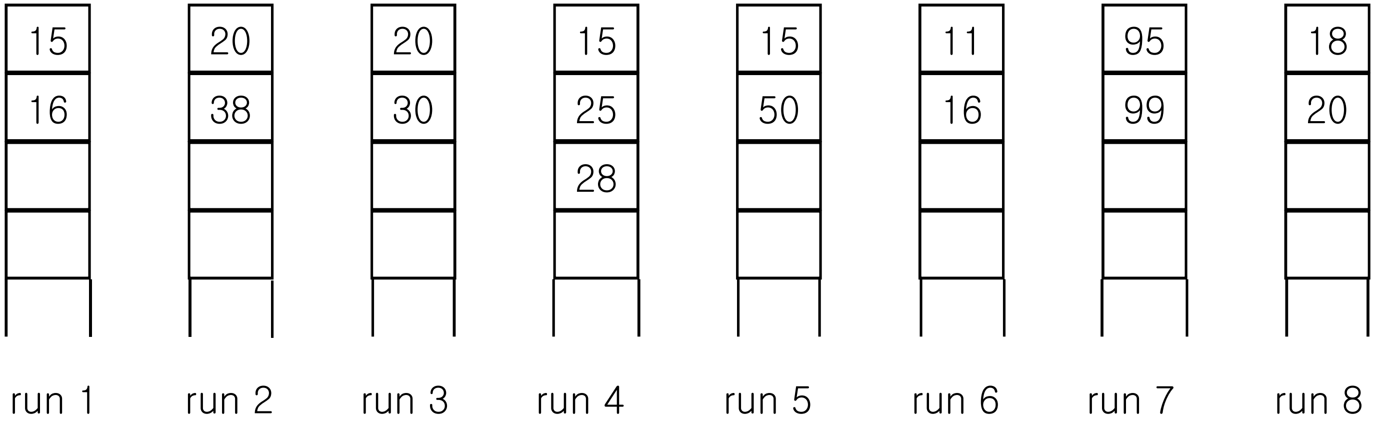
- Suppose we have *k* ordered sequences, called *runs*, that are to be merged into a single ordered sequence
  - Each run is in non-decreasing order of the key
- The merging task can be accomplished by repeatedly outputting the record with the smallest key



| run 1 | run 2 | run 3 | run 4 | run 5 | run 6 | run 7 | run 8 |
|---|---|---|---|---|---|---|---|
| 15 | 20 | 20 | 15 | 15 | 11 | 95 | 18 |
| 16 | 38 | 30 | 25 | 50 | 16 | 99 | 20 |
|  |  |  | 28 |  |  |  |  |

# Selection Trees (cont.)

- The smallest has to be found from $k$ possibilities, and it could be the leading record in any of the $k$ runs
- The most direct way to merge $k$ runs is to make $k - 1$ comparisons to determine the next record to output

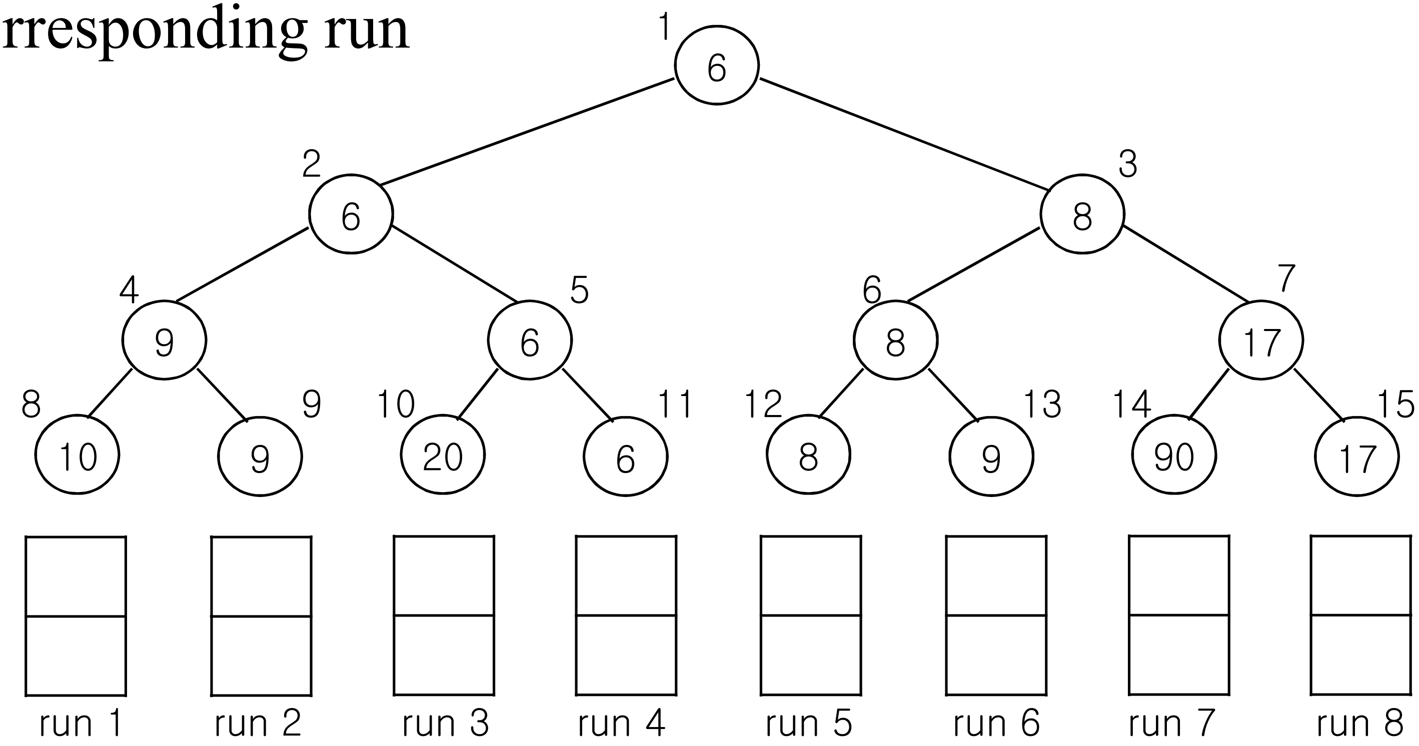


# Selection Trees (cont.)

- For $k > 2$, we can reduce the number of comparisons needed to find the next smallest element by using the *selection tree*
- There are two kinds of selection trees
  - Winner trees
  - Loser trees
- Selection trees are also called *tournament trees*

# Winner Trees

- A min (max) *winner tree* is a complete binary tree in which each internal node represents the smaller (larger) of its two children
- Each leaf node represents the first node in the corresponding run



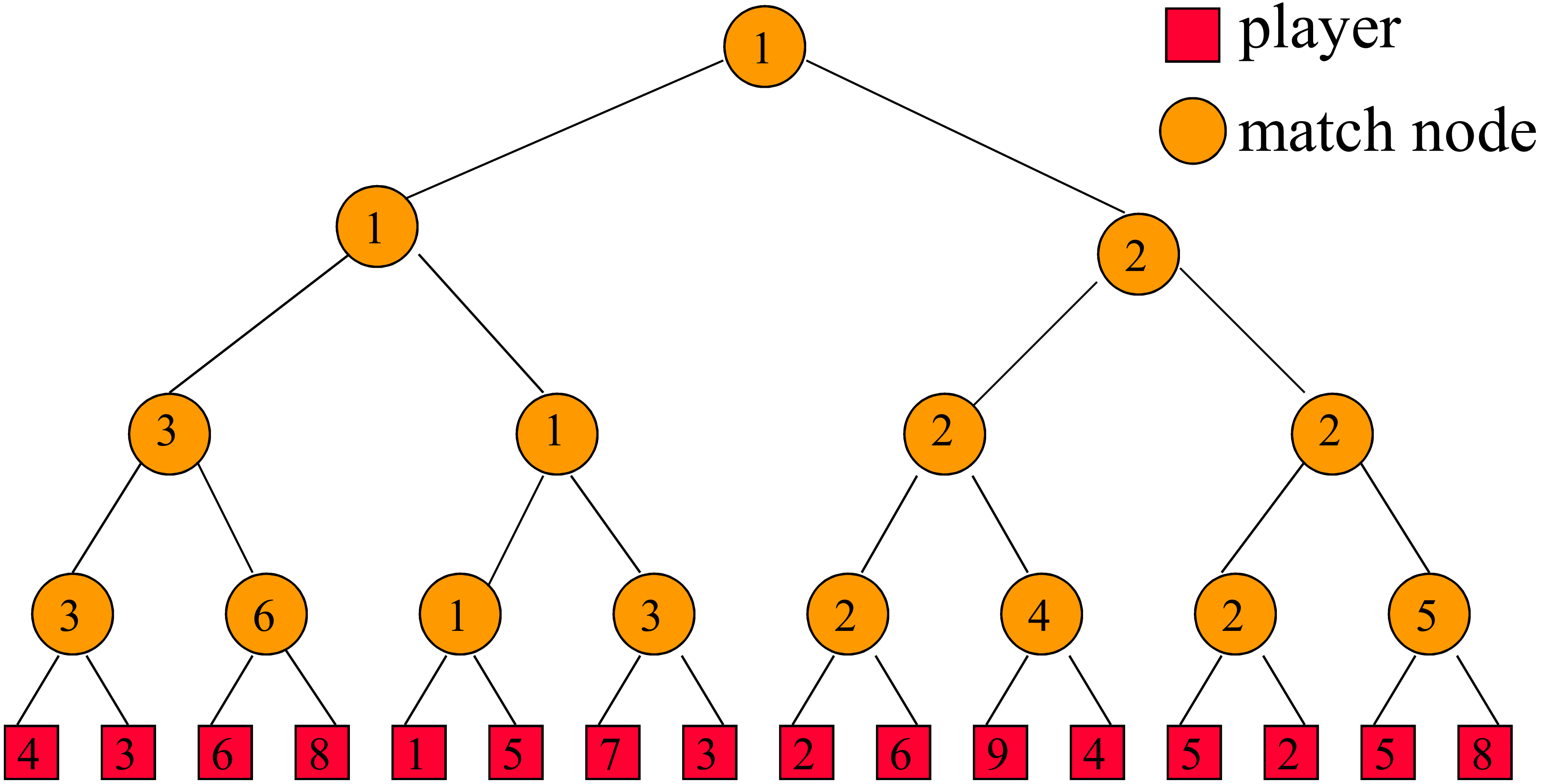
# Winner Trees (cont.)

- A winner tree has $k$ leaf nodes and $k - 1$ internal nodes
- Lemma 5.3: For any nonempty binary tree, $T$, if $n_0$ is the number of leaf nodes and $n_2$ the number of nodes of degree 2, then $n_0 = n_2 + 1$

# Winner Trees (cont.)

- Like the heap, a winner tree is a complete binary tree that is most efficiently stored using an array
- The construction of the winner tree may be compared to the playing of a tournament
  - Leaf nodes represent tournament players
  - Each internal node represents a match played between its two children; the winner of the match is stored at the internal node

# Winner Tree for 16 Players

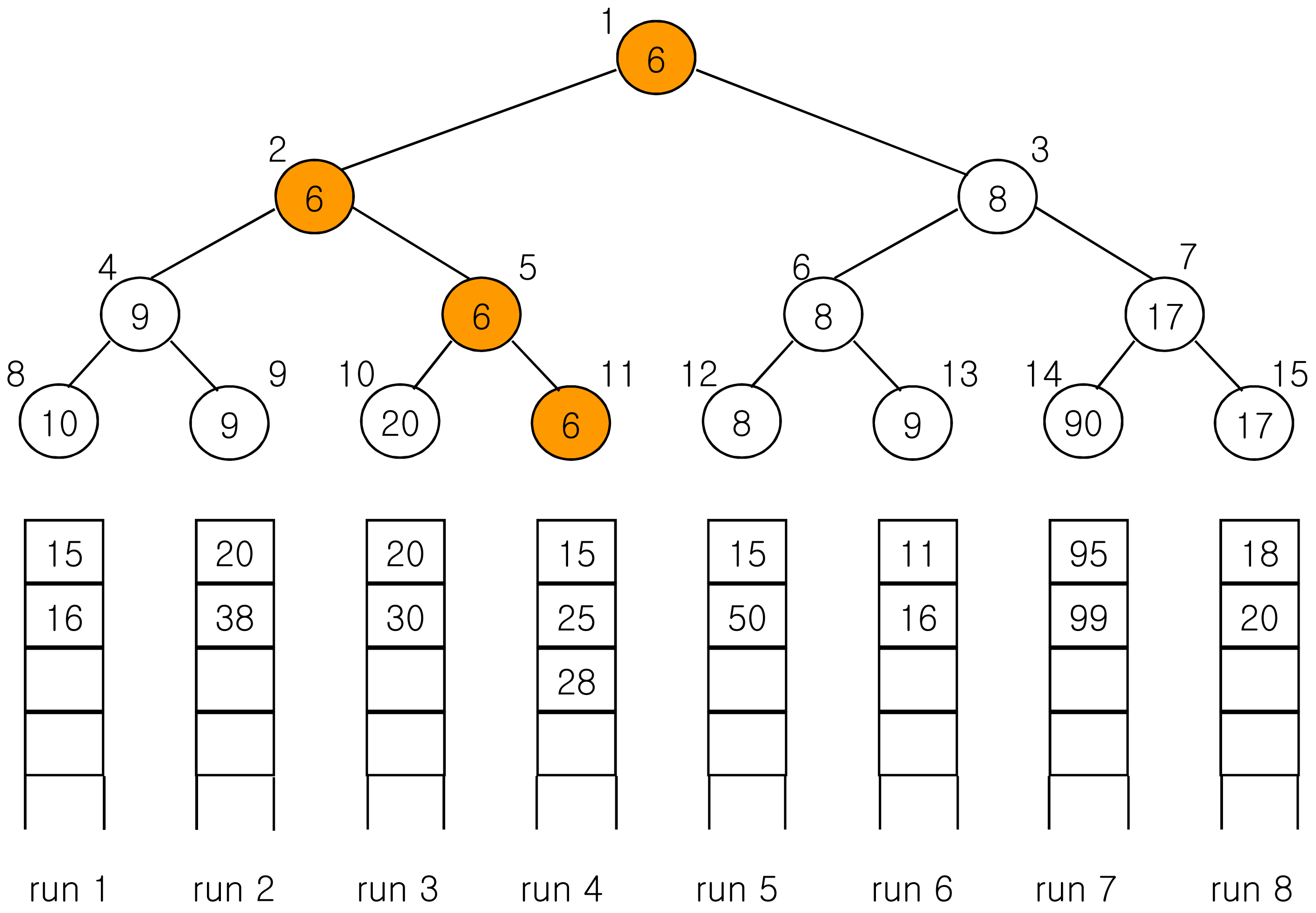


Smaller element wins => min winner tree.

# Winner Trees Operations

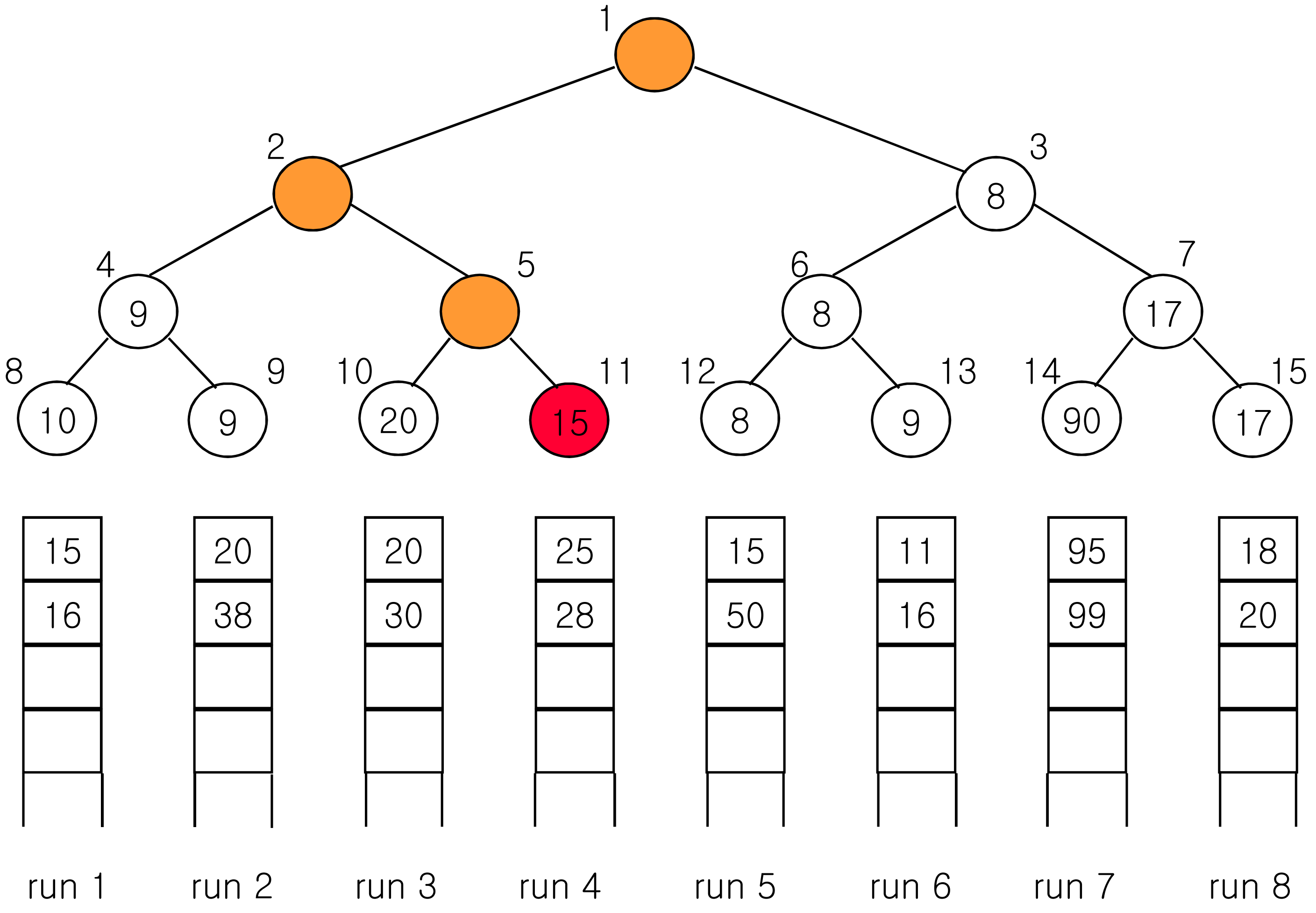
- Initialize
  - $k - 1$ match nodes
  - O($k$) time to initialize $k$-player winner tree
- Get winner
  - O(1) time
- Remove winner and replay
  - Remove winner and insert the next record from the run corresponding to the winner at the leaf corresponding to the winner
  - Replay tournament on the path from the leaf to the root
  - O(log $k$) time

# Remove Winner and Replay



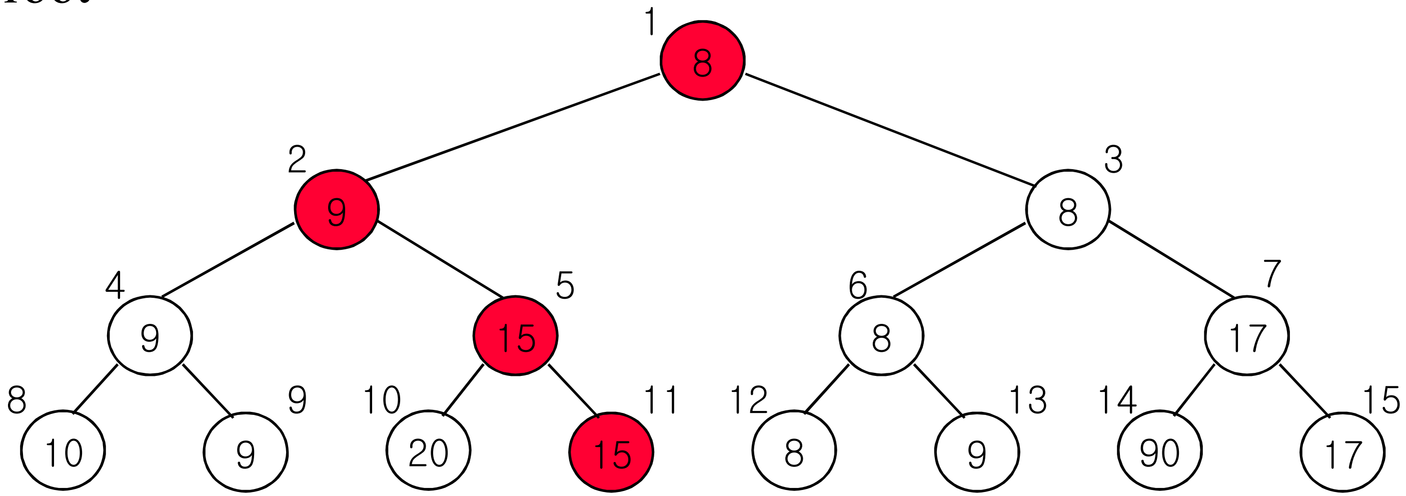
# Remove Winner and Replay (cont.)

Remove winner and insert the next record from the run corresponding to the winner at the leaf corresponding to the winner



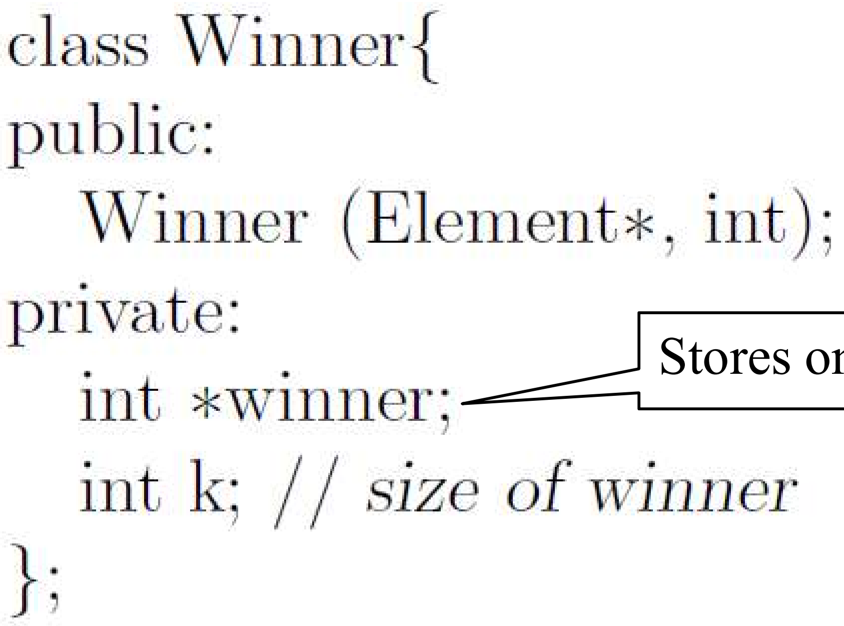
# Remove Winner and Replay (cont.)

Replay tournament on the path from the leaf corresponding to the winner to the root



| run 1 | run 2 | run 3 | run 4 | run 5 | run 6 | run 7 | run 8 |
|---|---|---|---|---|---|---|---|
| 15 | 20 | 20 | 25 | 15 | 11 | 95 | 18 |
| 16 | 38 | 30 | 28 | 50 | 16 | 99 | 20 |
| | | | | | | | |
| | | | | | | | |

# Class Definition for Winner Trees



```
class Winner{
public:
   Winner (Element*, int);
private:
   int *winner;
   int k; // size of winner
};
```

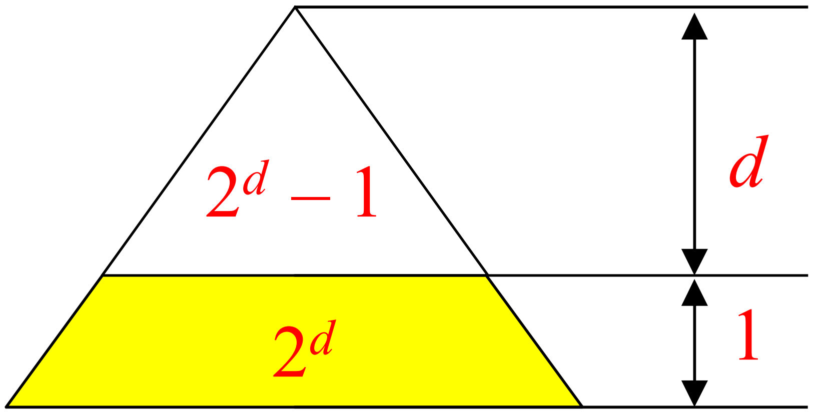
Stores only *k*-1 internal nodes

# Properties of Binary Trees

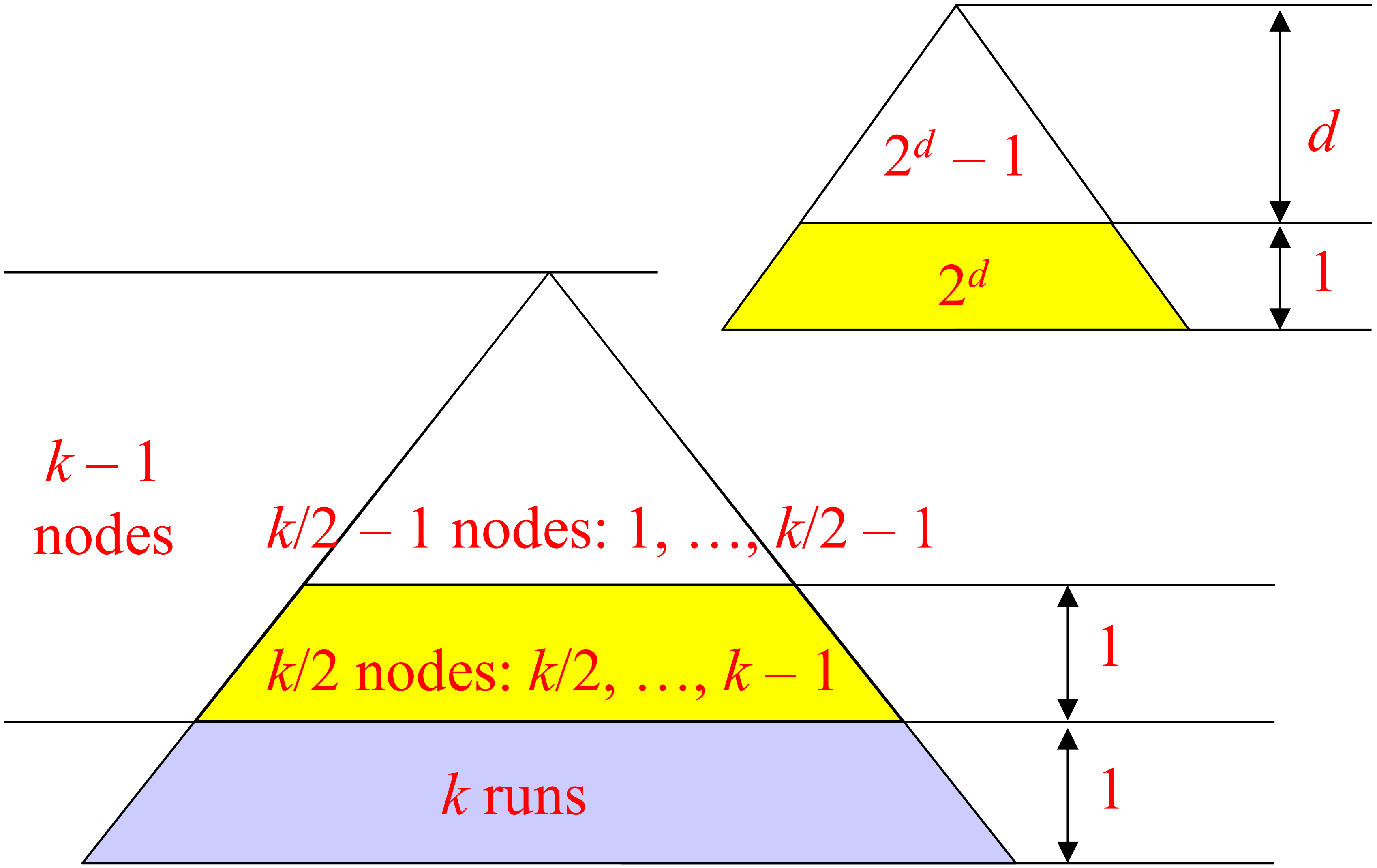
- The maximum number of nodes in a binary tree of depth $d$ is $2^d - 1$

$$\sum_{i=1}^{d} 2^{i-1} = 2^d - 1$$

- The maximum number of nodes on level $d + 1$ of a binary tree is $2^d$

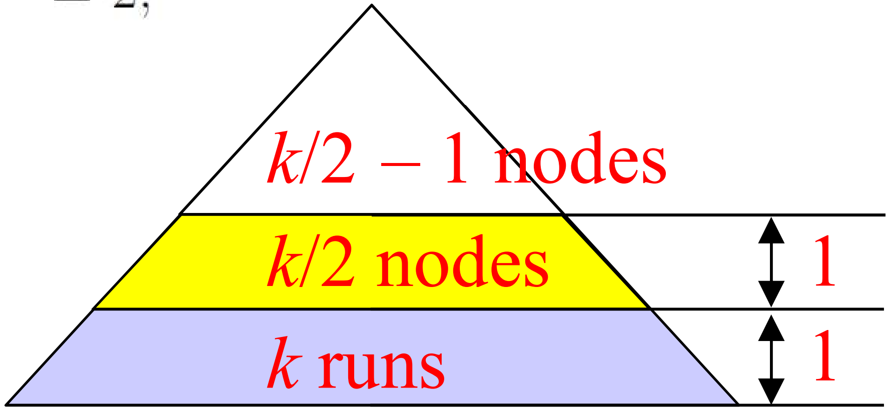


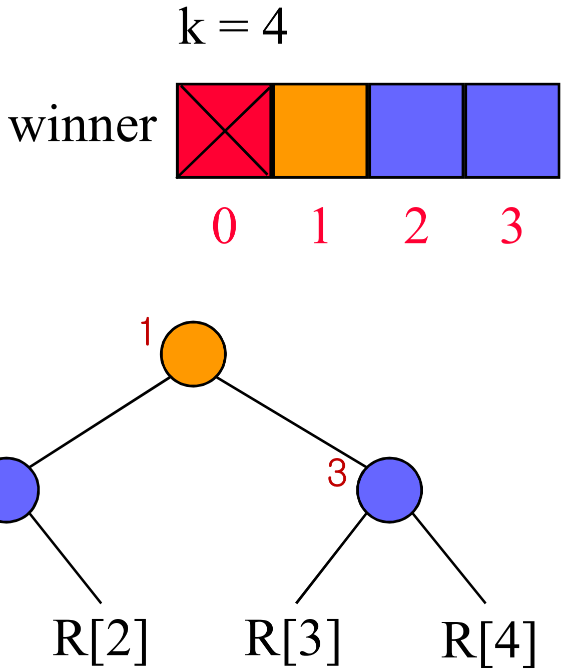
# Properties of Binary Trees (cont.)


$2^d - 1$
$d$
$2^d$
1
$k - 1$ nodes
$k/2 - 1$ nodes: 1, …, $k/2 - 1$
$k/2$ nodes: $k/2$, …, $k - 1$
1
$k$ runs
1

# Construct a Winner Tree of *k* Runs

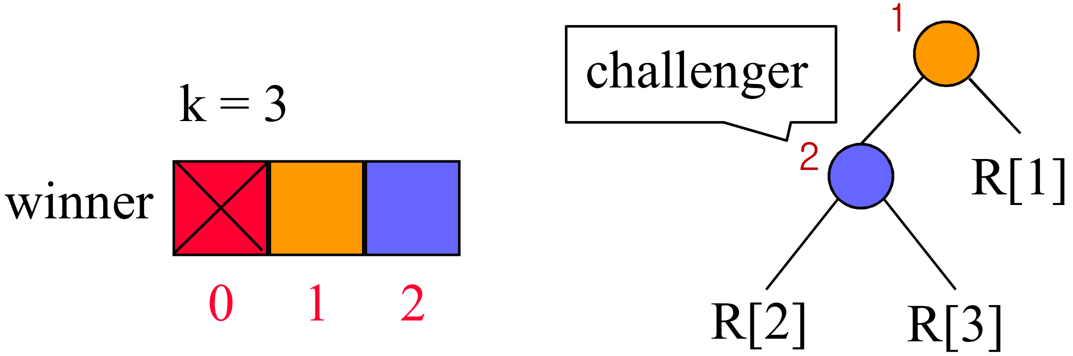
```
Winner::Winner(Element ∗R, int sz = TreeSize)
{
    k = sz;
    winner = new int [k]; // Don't want to use winner[0]
    for (int i = 1; i < k; i++) winner[i] = −1;

    int j = k;
    for (i = k−1; i ≥ k/2 && j ≠ 1 ; i−−) {
    // Play a tournament at each leaf of the tree
        if (R[j].key > R[j−1].key) winner[i] = j−1;
        else winner[i] = j;
        j −= 2;
    }
```





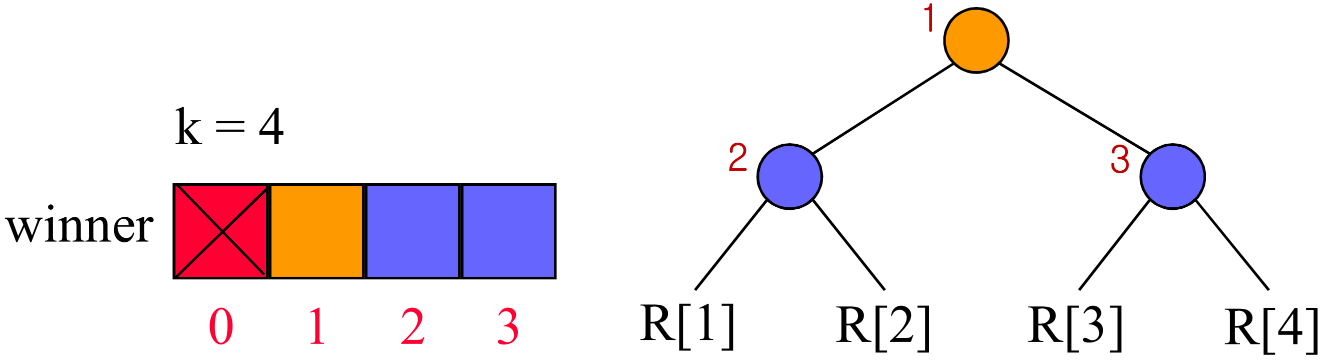
# Construct a Winner Tree of *k* Runs (cont.)

```
if (j == 1) {
// The tree contains a node which has one child.
// Play a tournament at this node
  winner[k/2] = 1;
  int challenger = winner[k/2 *2];
  if (R[challenger].key < R[1].key) winner[k/2] = challenger;
}
```



# Construct a Winner Tree of *k* Runs (cont.)

```
  for(i = k/2 − 1; i ≥ 1; i−−) {
  // Play a tournament at each internal node of the tree starting
  // from the bottom and moving towards the root
     j = 2∗i;
     if (R[winner[j]].key > R[winner[j+1]].key) winner[i] = winner[j+1];
     else winner[i] = winner[j];
  }
}
```



# Homework #4

**Due: 10/10일(수)** 수업시간까지 하드카피로 제출

Run의 개수 k가 다음과 같을 때 winner tree의 모양을 그려서 제출할 것

1. k = 5
2. k = 7
3. k = 11