

Review Data Structures

- Associative table
 - also known as map; dictionary
 - represented as list
 - access time: n
 - represented as tree
 - access time: $\log n$
 - represented as hashtable
 - collision
 - hash function
 - example: $\text{hash}(x) = x \bmod 17$
 - access time, worst case: n
 - access time, average case: slightly above 1

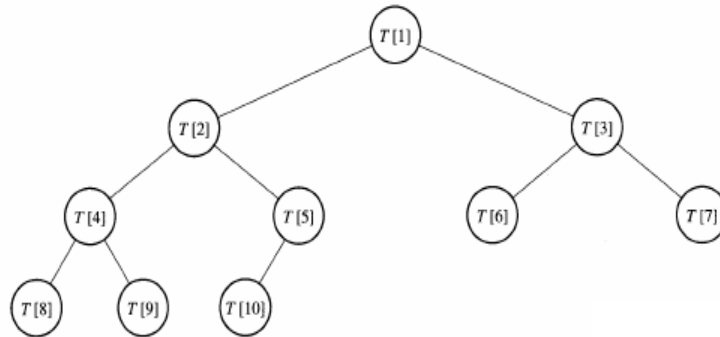
© 2005-06. 324-1-53

Review Data Structures

- Heap
 - a binary tree
 - essentially complete
 - can be stored in an array
 - with “heap property”:
 - value of each node is \geq values of child nodes
 - all nodes except leaves (and one special node)
 - have exactly two children
 - easy to keep sorted!
 - if a node is changed
 - if it becomes bigger: move up
 - if it becomes smaller: move down
 - heapsort algorithm

© 2005-06. 324-1-54

Heap



© 2005-06. 324-1-55

Heap

- Restoring heap condition: sift and percolate
 - sift: downward
 - percolate: upward

```
void sift( int k, int sizeOfHeap ) {           // k: index of node to sink
    while( 2*k <= sizeOfHeap ) {             // while we are not in lowest level
        int j = 2*k;
        if( j < sizeOfHeap && less( j, j+1 ) ) {
            j++;                               // go to right child if necessary
        }
        if( ! less( k, j ) ) {                 //heap condition is satisfied
            break;
        }
        exchange( k, j );
        k = j;                                 // go to next-lower level
    }
}
```

© 2005-06. 324-1-56

Heapsort

```

for( int k = n/2; k >= 1; k-- ) {           // make a heap
    sift( k, n );
}
while( n > 1 ) {                             // sort it
    exchange( 1, n );
    n--;
    sift( 1, n );
}

```

n: size of array (starting at 1)

- sorting works by successively removing largest element, and then fixing the heap
- runtime?

© 2005-06. 324-1-57

