Sorting - 2

CS 202 – Fundamental Structures of Computer Science II

Bilkent University

Computer Engineering Department

Heapsort

- For a binary heap, building the heap takes O(N) time.
- In a binary heap, deleteMin gives the smallest element.
  - Running time of O(logN).
- Basic strategy for heap sort given N elements
  - Build a binary heap from N elements (O(N))
  - Perform N deleteMin operations (N*logN)
    - Put the result of *deleteMin*s in a new array.
  - The new array is sorted.

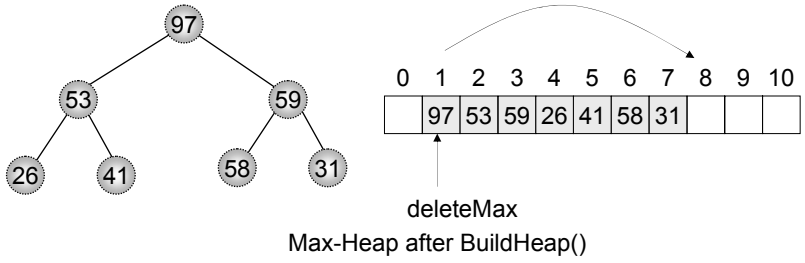
Heapsort

- The problem is that:
  - This strategy uses a new array
  - This means waste of space for large N.
- A way for using only one array.
  - When minimum item is deleted, put it to the end of the array.
  - After N deleteMins:
    - The original array will have the elements in sorted order from max to min.
  - If you want items in increasing order (from min to max), then use a Max-Heap.

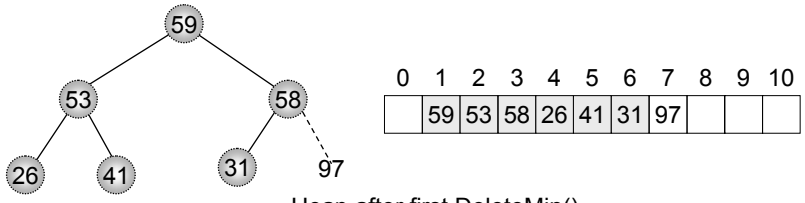
Example

- Assume the input that is to be sorted:
  - 31, 41, 59, 26, 53, 58, 97
  - N = 7.
- We need first to build a binary heap (a Max-Heap)
  - A max-Heap is heap that has the largest item in root.

Example



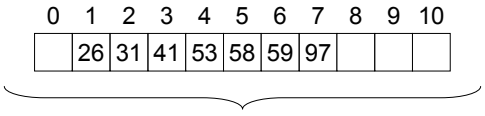
Max-Heap after BuildHeap()



Heap after first DeleteMin()

Example

- After 5 more deleteMins, the array will look like the following.



This is a sorted array

```
template <class Comparable>
      void heapsort( vector<Comparable> & a )
      {
/* 1*/     for( int i = a.size( ) / 2; i >= 0; i-- )  /* buildHeap */
/* 2*/         percDown( a, i, a.size( ) );

/* 3*/     for( int j = a.size( ) - 1; j > 0; j-- )
           {
/* 4*/         swap( a[ 0 ], a[ j ] );              /* deleteMax */
/* 5*/         percDown( a, 0, j );
           }
      }
```

i is running from N/2 to 0

j is running from N -1 to 1

Swap first and last elements

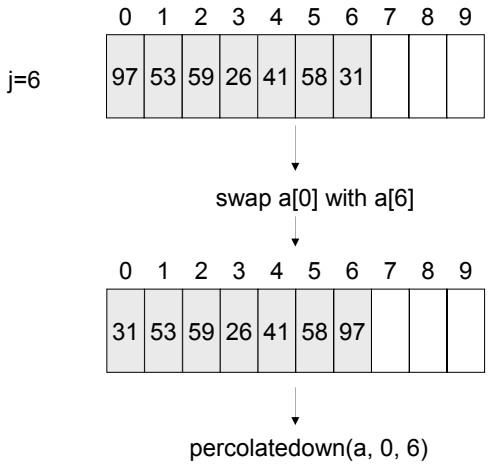

```
template <class Comparable>
      void percDown( vector<Comparable> & a, int i, int n )
      {
         int child;
         Comparable tmp;

/* 1*/    for( tmp = a[ i ]; leftChild( i ) < n; i = child )
         {
/* 2*/         child = leftChild( i );
/* 3*/         if( child != n - 1 && a[ child ] < a[ child + 1 ] )
/* 4*/            child++;
/* 5*/         if( tmp < a[ child ] )
/* 6*/            a[ i ] = a[ child ];
             else
/* 7*/            break;
         }
/* 8*/    a[ i ] = tmp;
      }
```

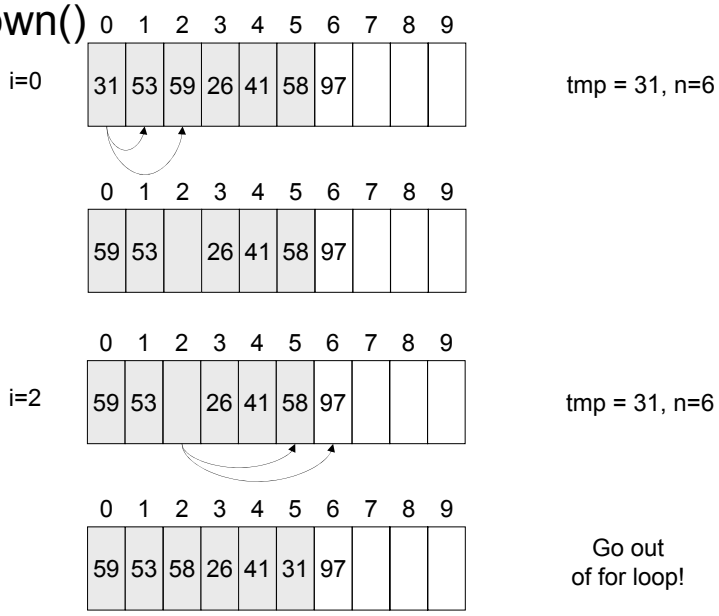
Point from where to start percolate down

Logical size of heap

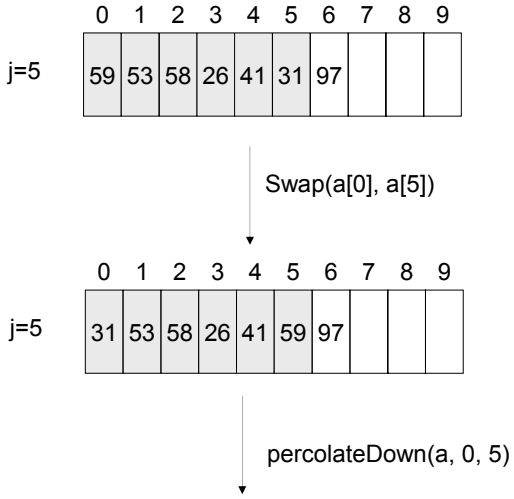
heapsort()



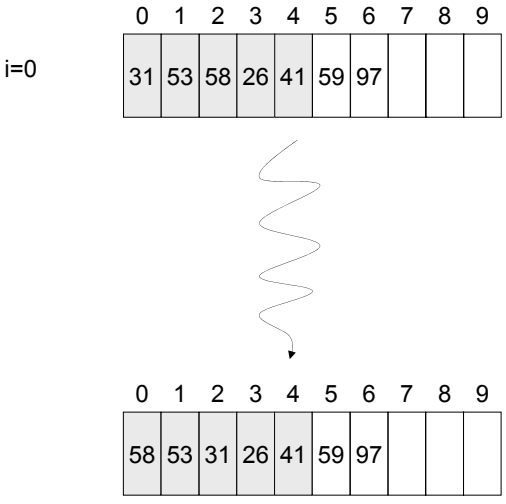
percDown()



heapsort()



percDown()

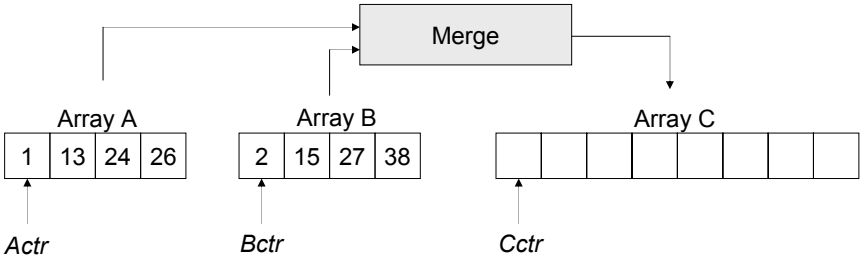


MergeSort

MergeSort

- Runs in O(NlogN) worst case running time.
- A recursive algorithm
- Based on merging to sorted lists
  - List 1: 2,3,5,9, 10
  - List 2: 1, 4, 6,7, 11
  - Merged List: 1,2,3, 4,5,6,7,10,11
- It is enough to do one pass over the *list 1* and *list 2*

Merging Algorithm

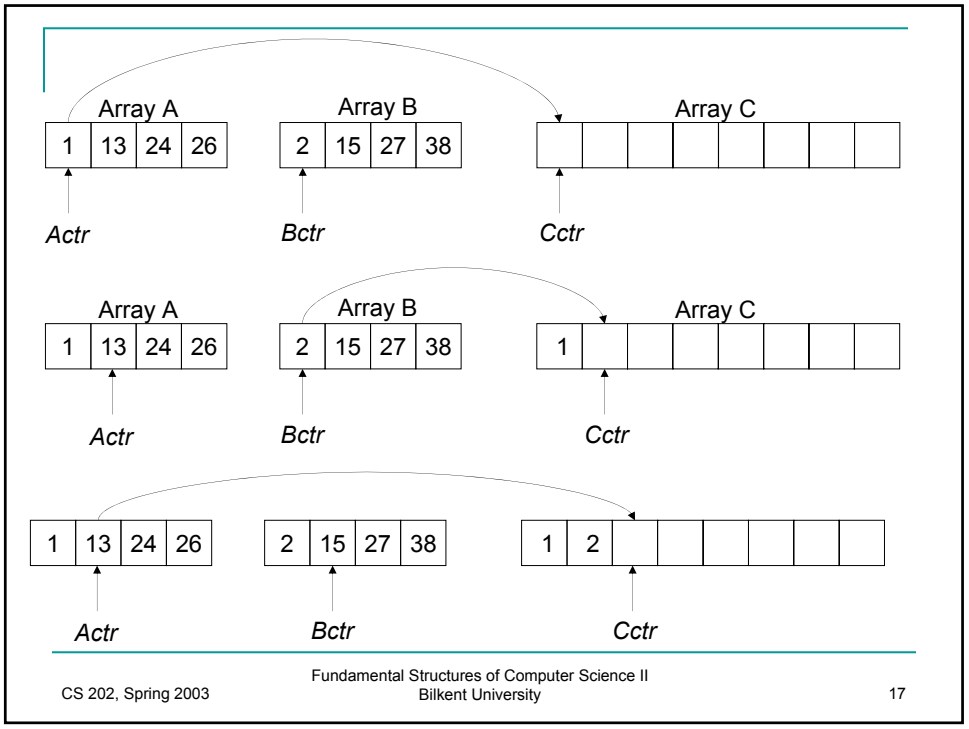


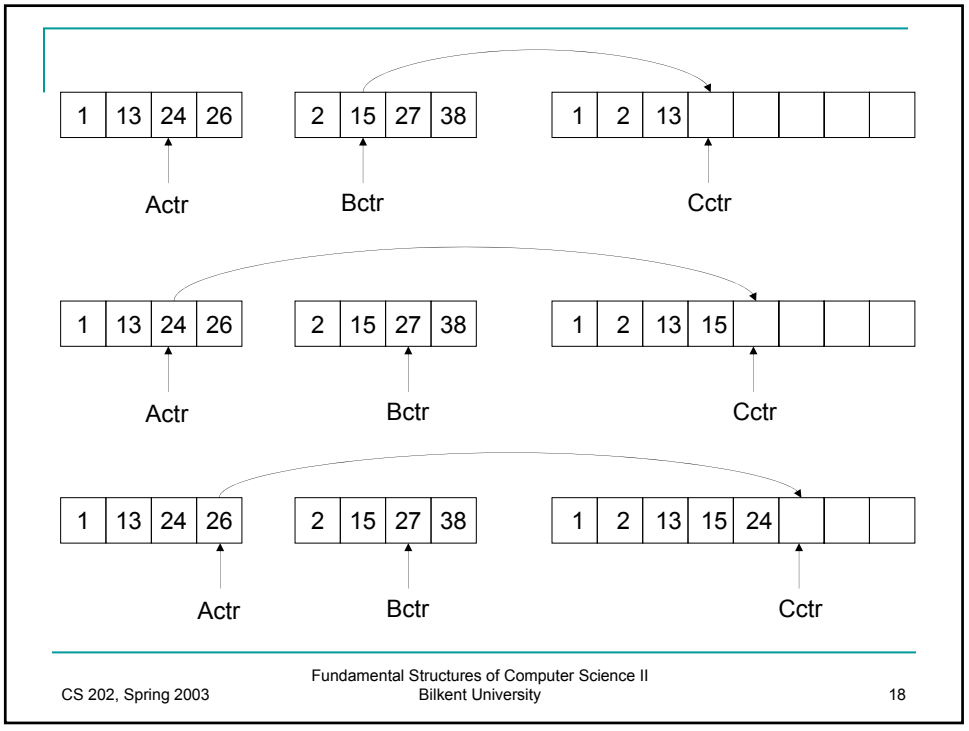
We maintain 3 counters: *Actr*, *Bctr*, *Cctr*

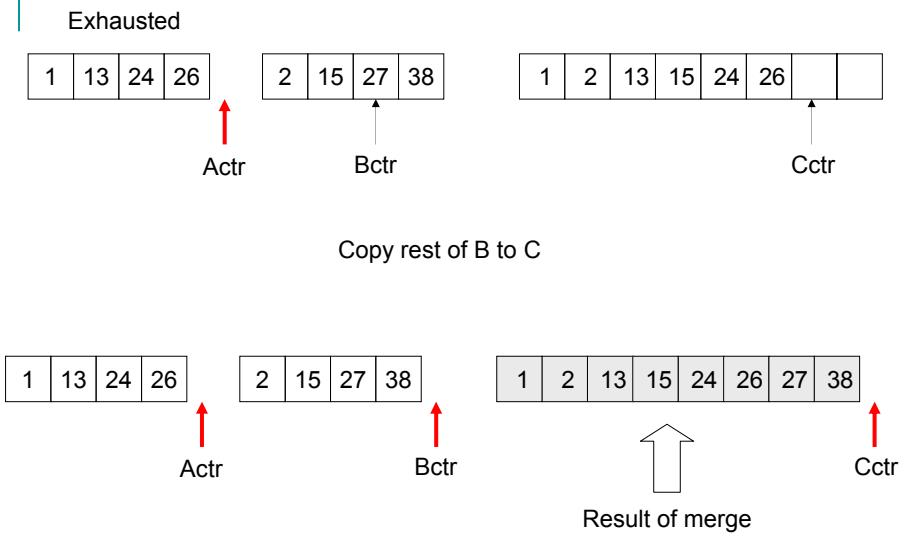
Pseudo-code for merge

```
While both Actr and Bctr did not reach to end of arrays
{
        if A[Actr]  < B[Bctr] then
                C[Cctr] = A[Actr]
                Actr++
        else
                C[Cctr] = B[Bctr]
                Bctr++
}

If Actr did not reach end of array A
        copy rest of A to C
If B ctr did not reach end of array B
        copy rest of B to C
```


Array A
1 13 24 26
Actr
Array B
2 15 27 38
Bctr
Array C
Cctr
Array A
1 13 24 26
Actr
Array B
2 15 27 38
Bctr
Array C
1
Cctr
1 13 24 26
Actr
2 15 27 38
Bctr
1 2
Cctr


1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13
Cctr
1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13 15
Cctr
1 13 24 26
Actr
2 15 27 38
Bctr
1 2 13 15 24
Cctr



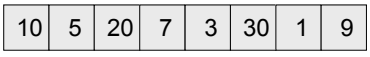
MergeSort Algorithm Description

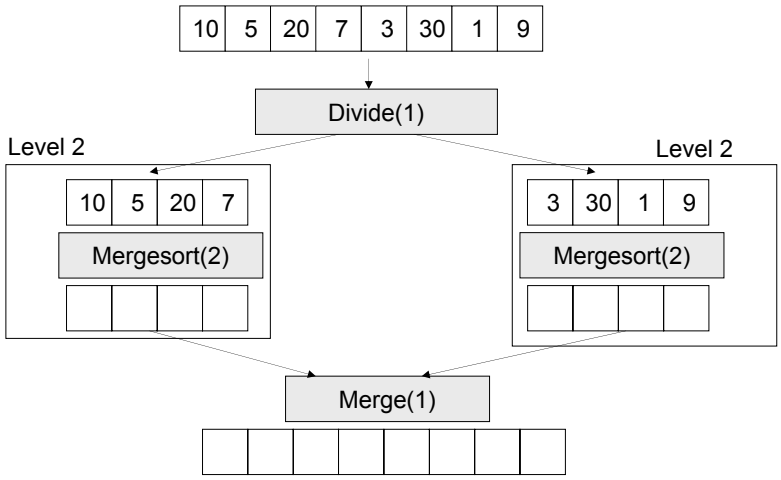
- Mergesort (A[1:N])
  1. If N = 1 then
     A. return
  2. Else
     A. Mergesort (A[1:N/2])
     B. Mergesort (A[N/2+1:N]
     C. Merge (A[1:N/2], A[N/2+1:N])
     D. return

example

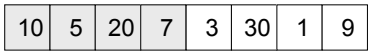
- Mergesort the following array
  - 10 5 20 7 3 30 1 9

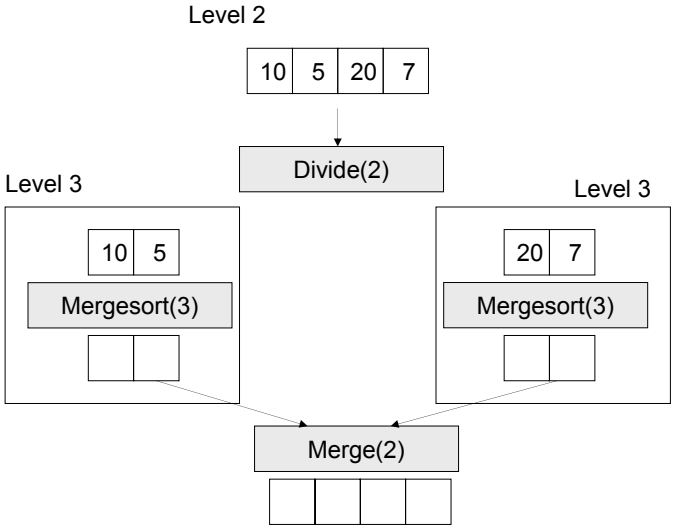
Level 1



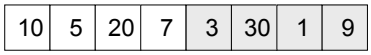


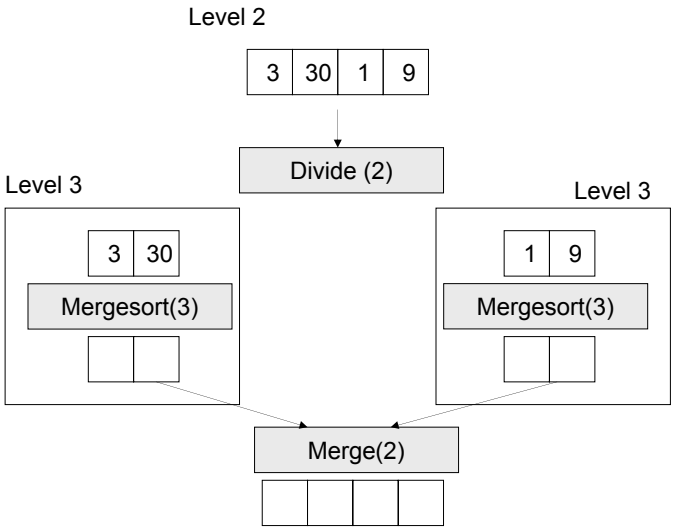
Level 2



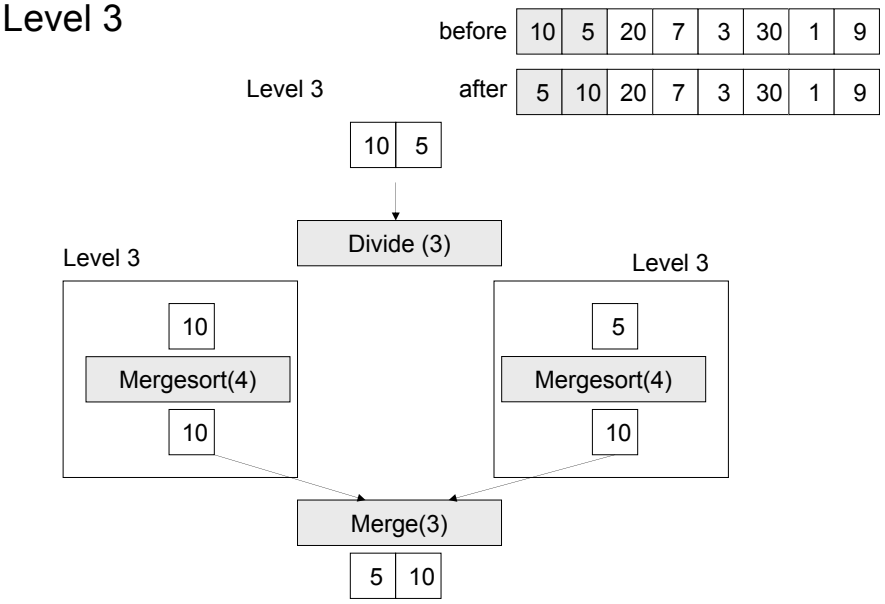


Level 2

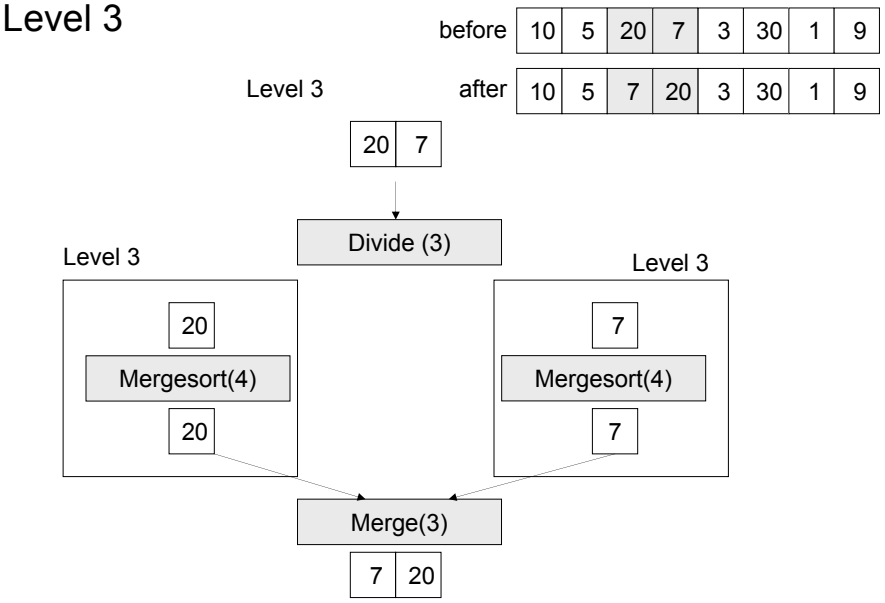




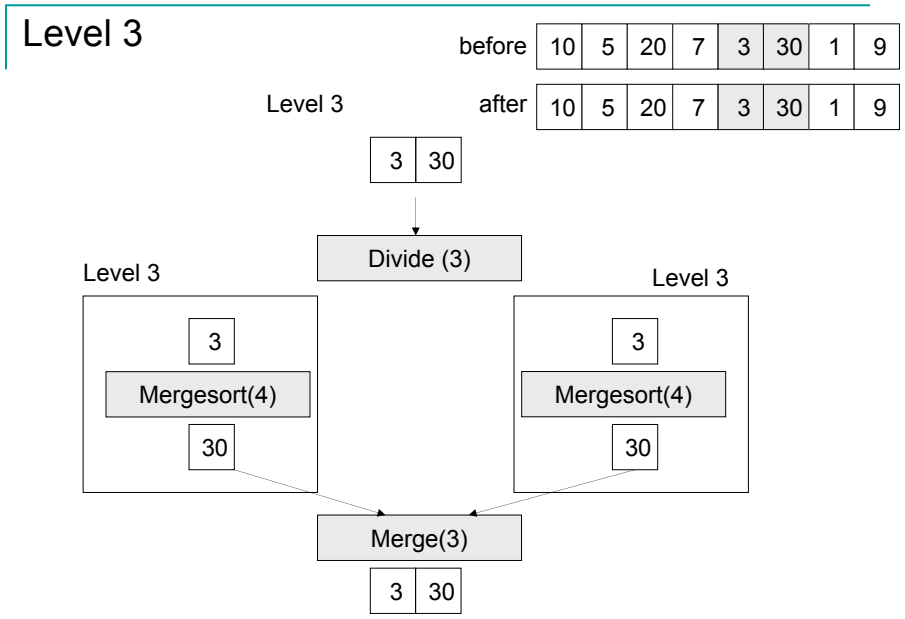
Level 3



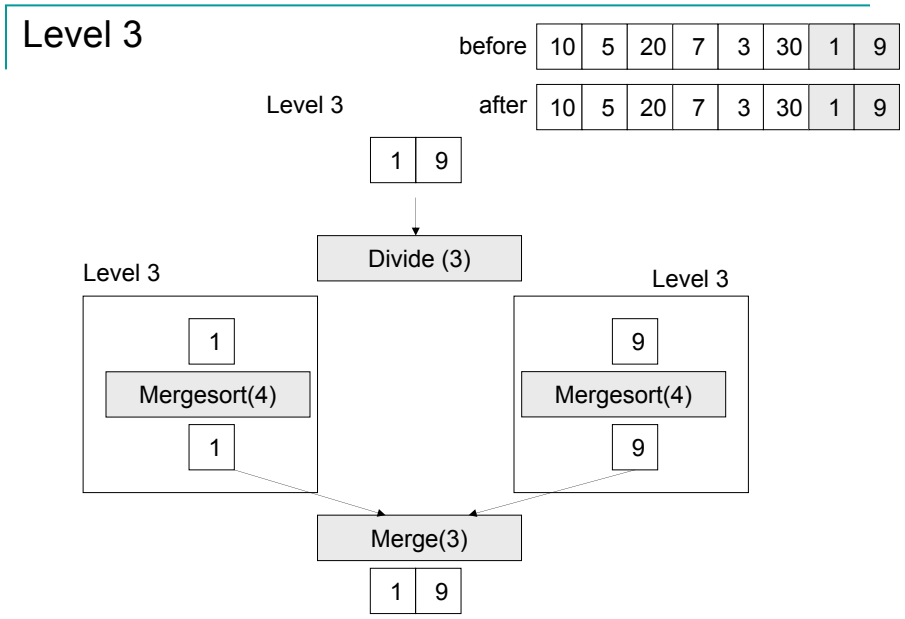
Level 3



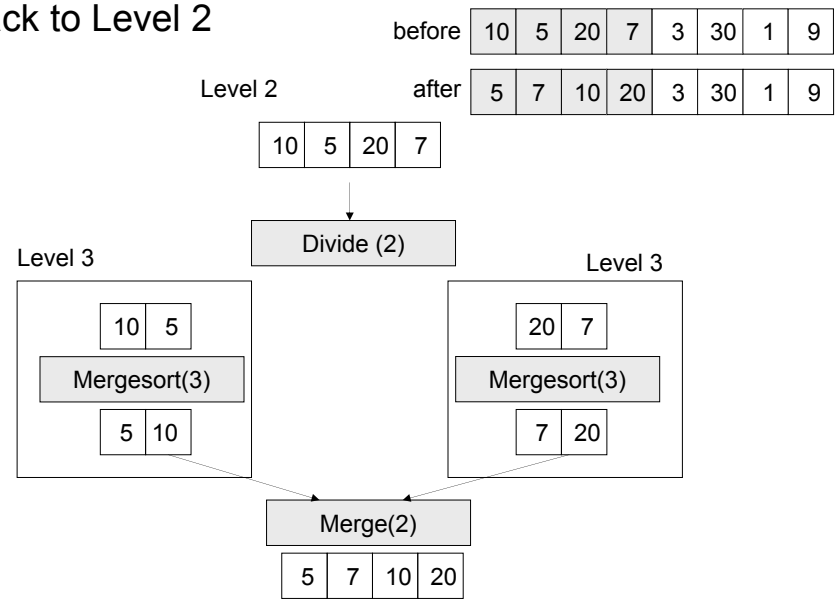
Level 3



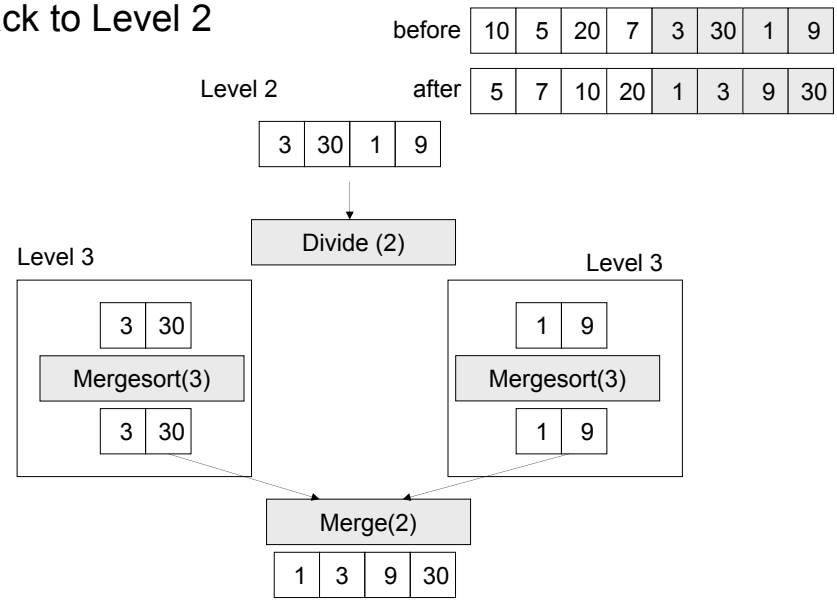
Level 3



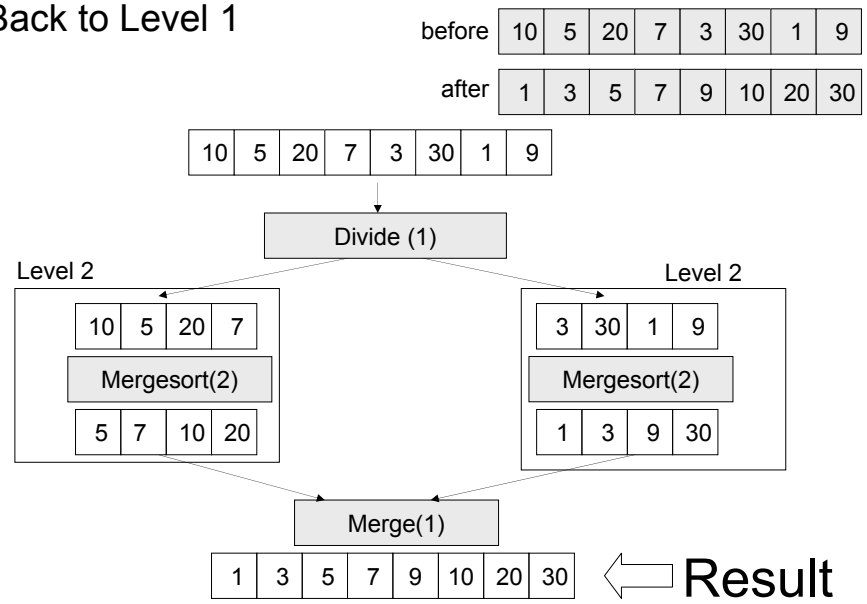
Back to Level 2



Back to Level 2



Back to Level 1



MergeSort

- The sorted array is
  - 1,3,5,7,9,10,20,30
- MergeSort is an example of Divide and Conquer Approach in Algorithm Design.