Data Structures 2018
Exercise 9, solutions (Week 45)

1. See file Sort.java in folder Problem 1.

2.-3. See file Sort.java in folder Problem 2-3.

4. Algorithm 1. Notice that indexing begins from 1, not from 0 as it usually does with arrays.

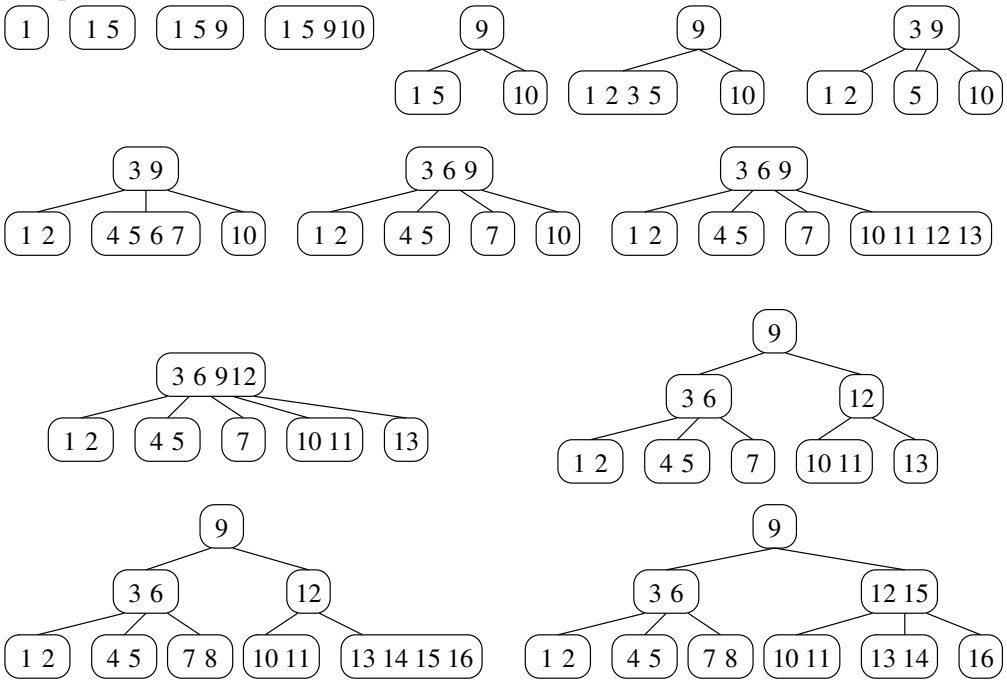
Algorithm 1 Sorting table A in ascending order using an in-place heap sort.

```
HeapSort(A)
  BuildMaxHeap(A)
  heapsize ← A.length
  for i ← A.length downto 2 do
    Swap(A[1],A[i])
    heapsize ← heapsize - 1
    PercolateDown(A,1)
  end for
```

Algorithm 2 Creating a maximum-heap from table A.

```
BuildMaxHeap(A)
  for i ← ⌊A.length/2⌋ downto 1 do
    PercolateDown(A,i)
  end for
```

5. See picture below.

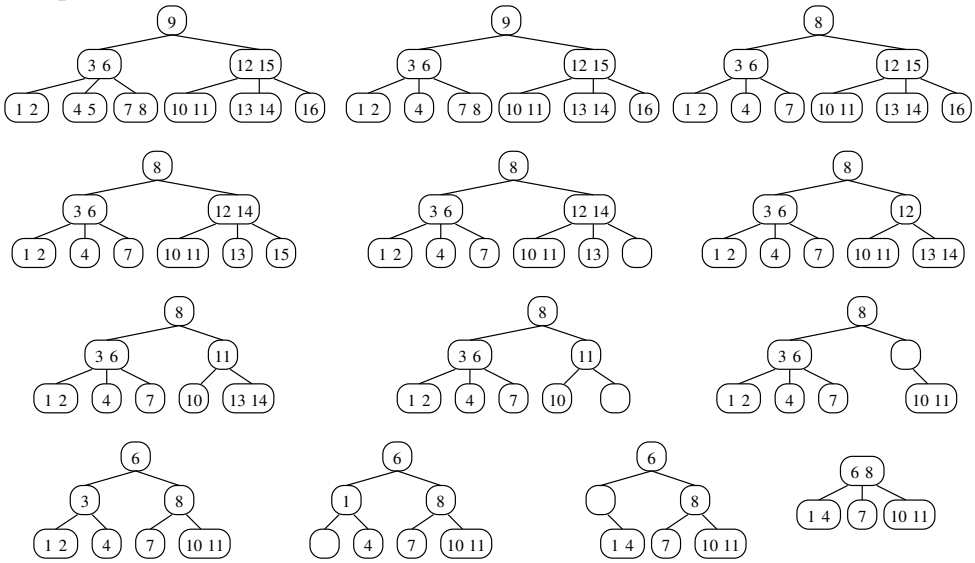


There can be 1, 2 or 3 keys in a node of (2,4) tree. If insertion increases the number of keys in a node to 4 (overflow), the node must be splitted in two parts. First, the key in the middle of the overflow node is lifted to the parent of this node. If there is no parent, a new root is created and the key lifted into it (look first row in the picture). The key selected from the middle is the 3rd key.

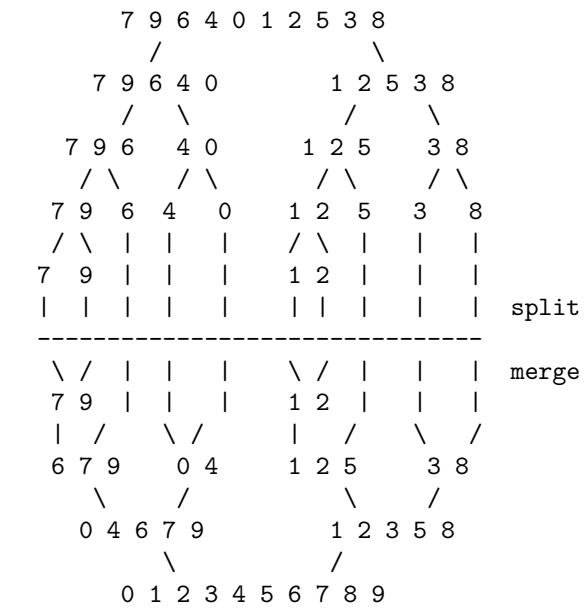
The keys from the left side of the 3rd key are put into a node and the key from the right side to another one. These nodes are connected to the parent including the lifted key as left and right side child nodes.

If the lift causes overflow to the parent node, the same process has to be done also in the parent node (look row 2 and 3 in a picture, addition of 13).

6. See picture below.



7. Merge sort



Quick sort

```
7 9 6 4 0 1 2 5 3 8
                8    pivot (E)
7 6 4 0 1 2 5 3   9  L, G
      3
0 1 2   7 6 4 5
    2     5
0 1     4   7 6
  1         6
0             7
------------------
0 1 2 3 4 5 6 7 8 9
```

The function calls of merge sort.

```
MergeSort((7 9 6 4 0 1 2 5 3 8))
        Split((7 9 6 4 0 1 2 5 3 8)) => (7 9 6 4 0), (1 2 5 3 8)
        MergeSort((7 9 6 4 0))
                Split((7 9 6 4 0)) => (7 9 6), (4 0)
                MergeSort((7 9 6))
                        Split((7 9 6)) => (7 9), (6)
                        MergeSort((7 9))
                                Split ((7 9)) => (7), (9)
                                MergeSort((7)) => (7)
                                MergeSort((9)) => (9)
                                Merge((7), (9))
                        => (7, 9)
                        MergeSort((6)) => (6)
                        Merge((7,9),(6))
                => (6,7,9)
                MergeSort((4,0))
                        Split((4, 0)) => (4), (0)
                        MergeSort((4)) => (4)
                        MergeSort((0)) => (0)
                        Merge((0), (4))
                => (0, 4)
                Merge((6,7,9), (0,4))
        => (0, 4, 6, 7, 9)
        MergeSort((1 2 5 3 8))
                Split((1 2 5 3 8)) => (1 2 5), (3 8))
                MergeSort((1 2 5))
                        Split((1 2 5)) => (1 2), (5)
                        MergeSort((1 2))
                                Split ((1 2)) => (1), (2)
                                MergeSort((1)) => (1)
                                MergeSort((2)) => (2)
                                Merge((1), (2))
                        => (1, 2)
                        MergeSort((5)) => (5)
                        Merge((1,2),(5))
                => (1,2,5)
                MergeSort((3,8))
                        Split((3, 8)) => (3), (8)
                        MergeSort((3)) => (3)
                        MergeSort((8)) => (8)
                        Merge((3), (8))
                => (3, 8)
                Merge((1,2,5), (3,8))
        => (1, 2, 3, 5, 8)
        Merge((0, 4, 6, 7, 9), (1, 2, 3, 5, 8))
=> (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

The function calls of quick sort.

```
QuickSort((7 9 6 4 0 1 2 5 3 8))
        Partition((7 9 6 4 0 1 2 5 3 8)) => (7 6 4 0 1 2 5 3), (8), (9)
        QuickSort((7 6 4 0 1 2 5 3))
                Partition((7 6 4 0 1 2 5 3)) => (0 1 2), (3), (7 6 4 5)
                QuickSort((0 1 2))
                        Partition((0 1 2)) => (0 1), (2), ()
                        QuickSort((0 1))
                                Partition((0 1)) => (0), (1), ()
                                QuickSort((0)) => (0)
                                QuickSort(()) => ()
                                Append((0), (1), ())
                                => (0 1)
                        => (0 1)
                        QuickSort(()) => ()
                        Append((0 1), (2), ())
                        => (0 1 2)
                => (0 1 2)
                QuickSort((7 6 4 5))
                        Partition((7 6 4 5)) => (4), (5), (7 6)
                        QuickSort((4)) => (4)
                        QuickSort((7 6))
                                Partition((7 6)) => (), (6), (7)
                                QuickSort(()) => ()
                                QuickSort((7)) => (7)
                                Append ((), (6), (7))
                                => (6 7)
                        => (6 7)
                        Append ((4), (5), (6 7))
                        => (4 5 6 7)
                => (4 5 6 7)
                Append((0 1 2), (3), (4 5 6 7))
                => (0 1 2 3 4 5 6 7)
        => (0 1 2 3 4 5 6 7)
        QuickSort((9)) => (9)
        Append((0 1 2 3 4 5 6 7), (8), (9))
        => (0 1 2 3 4 5 6 7 8 9)
=> (0 1 2 3 4 5 6 7 8 9)
```