1. (9 Points) Trace bottom-up merge sort (see class notes for implementation details) with the following input. Each merge result row should have the merging result from the previous step, and the last row should contain the final sorted result.

Index	0	1	2	3	4	5	6	7	8	9
Input	8	22	22	79	60	17	72	10	70	9
Merge Result	8	22	22	79	17	60	10	72	9	70
Merge Result	8	22	22	79	10	17	60	72	9	70
Merge Result	8	10	17	22	22	60	72	79	9	70
Merge Result	8	9	10	17	22	22	60	70	72	79

2. (6 Points) Show the partition result of this input array using the algorithm provided on class slides. Provide array contents after each swap (including the final swap). If there are multiple swaps, provide a row for each swap.

Index	0	1	2	3	4	5	6	7	8	9
Input	68	1	74	0	69	27	93	67	28	98
Swap result	68	1	28	0	69	27	93	67	74	98
Swap result	68	1	28	0	67	27	93	69	74	98
Swap result	27	1	28	0	67	68	93	69	74	98

3. (10 Points) Use heap sort to sort the following input array into **non-increasing** order. Ignore array element at index 0.

Index	0	1	2	3	4	5	6	7	8	9	10
Original Array	null	39	25	84	85	36	84	27	62	42	20
After heap construction	null	20	25	27	42	36	84	84	62	85	39
After Swap & Sink	null	25	36	27	42	39	84	84	62	85	20
After Swap & Sink	null	27	36	84	42	39	85	84	62	25	20
After Swap & Sink	null	36	39	84	42	62	85	84	27	25	20
After Swap & Sink	null	39	42	84	84	62	85	36	27	25	20
After Swap & Sink	null	42	62	84	84	85	39	36	27	25	20
After Swap & Sink	null	62	84	84	85	42	39	36	27	25	20
After Swap & Sink	null	84	85	84	62	42	39	36	27	25	20
After Swap & Sink	null	84	85	84	62	42	39	36	27	25	20
After Swap & Sink	null	85	84	84	62	42	39	36	27	25	20

4. (6 Points) For Quick3way implementation provided below~~, what is the running time and space complexity given an input array with all equal keys? Provide brief explanation~~.

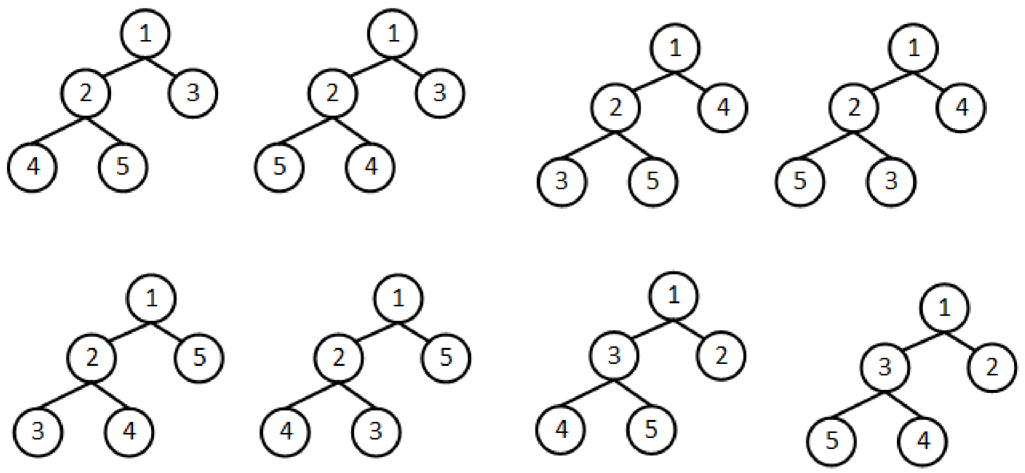
```
// quick sort with 3-way partitioning
public static <T extends Comparable<T>> void sort(T[] a, int lo, int hi) {
        if (hi <= lo) return;

        int[] results = partition(a, lo, hi);
        int lt = results[0];
        int gt = results[1];

        sort(a, lo, lt - 1);
        sort(a, gt + 1, hi);
}

private static  <T extends Comparable<T>> int[] partition(T[] a, int lo,
int hi) {
        int lt = lo, i = lo + 1, gt = hi;
        T v = a[lo];

        while (i <= gt) {
                int cmp = a[i].compareTo(v);
                if (cmp < 0) {
                        swap(a, lt++, i++);
                } else if (cmp > 0) {
                        swap(a, i, gt--);
                }else {
                        i++;
                }
        } // now a[lo..lt-1] < v; v == a[lt..gt]; v < a[gt+1..hi].

        int[] results = new int[2];
        results[0] = lt;
        results[1] = gt;

        return results;
}

private static <T extends Comparable<T>> void swap(T[] a, int i, int j) {
        T temp = a[i];
        a[i] = a[j];
        a[j] = temp;
}
```

a) Given a = {3, 1, 1, 1, 7, 3, 3, 5, 1, 5, 1, 7}, lo = 0, hi = 11; what are the contents of results array returned by **partition**(a, lo, hi) call?
Answer: [5, 7]

b) If an input array (a) has three distinct values, provide performance analysis for **sort** method given that input.

	Big-O Notation	Explanation
Best case Running Time	O(n)	In all cases, 3 partition calls (ignore base case when hi<=low) are needed to sort the array. Since partition method is O(n) in running time for all cases, thus total running time is O(n).
Worst case Running Time	O(n)	The same as above.
Best case Space Complexity	O(1)	The space complexity for partition method is O(1) for all cases. Sort method only uses three int variables, which is O(1) in space. Recursion depth with the given input is at most 3. Thus space complexity is O(1).
Worst case Space Complexity	O(1)	The same as above.

5. (8 Points) Given 5 keys, 1, 2, 3, 4, and 5. Draw all possible binary min heaps with those keys.
Answer:



6. (6 Points) Mark true or false for each of the following statements and provide brief explanation.
a) If the input array is randomly shuffled, the worst-case running time for quick sort would not be O(n^2).
b) Suppose we choose the element in the middle position of the array as the pivot. This makes it unlikely that quicksort will require quadratic time.
c) Merging two sorted arrays of N items requires at least 2N – 1 comparisons. (Note each call of SortUtils.isLessThan(…) is considered one comparison).
Answer:
a) False. Even after input array is randomly shuffled, there is still very small probability that the input array ends up as sorted or reversed sorted order, which can result in O(n^2) running time.

b) False. The element in the middle position can still be the smallest or the largest, thus after partition either left or right sub-array is empty. If the element is the middle is always the smallest or the largest, it can lead to quadratic time.

c) False. Suppose the first element in the right array is greater than all elements in the left array, then that element will be compared with every element in the left array, that requires N comparisons. After that, since left array is exhausted, elements in the right array are simply copied to the results without comparison.

7. (5 Points) Given three arrays of N names each, describe an O(nlogn) algorithm to determine if there is any name common to all three arrays, and if so, return the first such name?
Answer:
We can first use merge sort or heap sort to sort each array, the running time for sorting three arrays is O(nlogn) in worst case. With three sorted arrays, we can use three-way merge similar to two-way merge used in merge sort algorithm. With three-way merge, there is a cursor for each array, and the cursor under the smallest name moves one position to the right. During the process, if at one point name under every cursor is the same, then that name is common to all arrays and we can return that name. Otherwise, there is no common name to all arrays. This three-way merge is O(n) in running time, because it scans each array once. Thus, the entire algorithm is O(nlogn) in running time.

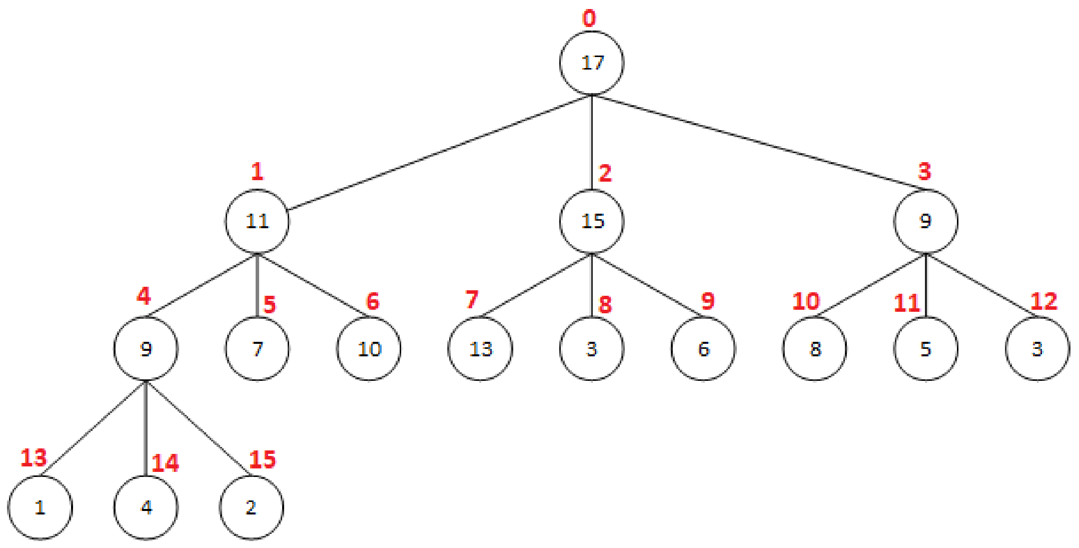
8. (6 Points) Given an input array with all equal values, compare the following sorting algorithm using big-O notation.

	Running Time	Space Complexity
Merge Sort	O(nlgn)	O(n)
Quick Sort	O(nlgn)	O(lgn)
Heap Sort	O(n)	O(1)

9. (4 Points) Answer the following questions.

a) What are the minimum and maximum numbers of nodes in a binary heap of height h?
Answer: The minimum number of nodes is 2^h, and the maximum number of nodes is $2^{(h+1)}$-1.
b) Is an array that is in non-decreasing sorted order a min-heap?
Answer: Yes. Here is an example: 2, 4, 6, 7, 7, 9.

10. (10 Points) D-heap is like a binary heap except that all nodes have d children (thus, a binary heap is a 2-heap). The following is an example max 3-heap, where keys are in black and array indexes are in red.



Implement the following methods in MaxDHeap class with root at index **0**.
public int getParentIndex(int k)
public int getLeftMostChildIndex(int k)
public int getRightMostChildIndex(int k)
private void swim(int k)
private void sink(int k)

Answer: An example solution is provided on the next page

```
public int getParentIndex(int k) {
        if (k==0) {
                return -1;
        } else {
                return (k-1)/d;
        }
}

public int getLeftMostChildIndex(int k) {
        return k*d + 1;
}

public int getRightMostChildIndex(int k) {
        return (k+1)*d;
}

private void swim(int k) {
        int d = this.d;
        while (k > 0 && isLessThan(pq[(k-1)/d], pq[k])) {
                swap((k-1)/d, k);
                k = (k-1)/d;
        }
}

private void sink(int k) {
        int d = this.d;
        while (d*k +1 < size) {
                int j = findLargestChildIndex(k);
                if (!isLessThan(pq[k], pq[j]))
                        break;
                swap(k, j);
                k = j;
        }
}

private int findLargestChildIndex(int k) {
        int j = -1;
        int index = d*k + 1;

        if (index<size) {
                Key maxKey = pq[index];
                j = index;

                for (int i=2; i<=d; i++) {
                        index = d*k + i;
                        if (index>=size) {
                                break;
                        } else {
                                if (isLessThan(maxKey, pq[index])) {
                                        maxKey = pq[index];
                                        j = index;
                                }
                        }
                }
        }

        return j;
}
```

Submission Note

1) For written part of the questions:
   a. Write your answers inside a text document (in plain text, MS Word, or PDF format)
   b. Name the file as firstname.lastname.assignment2.txt(doc, docx, or pdf) with proper file extension
2) For programming part of the questions
   a. Use JDK 1.8 and Junit5
   b. Put your full name at the beginning of every source file you created or modified. **2 points will be deducted if your names are not included in the source files.**
   c. Do not change the provided package, class, or method name. You can add extra classes or methods if they are needed.
   d. **If your code does not compile, you will get zero point**.
   e. Use the provided tests to verify your implementation. **Extra tests will be used for grading.**
   f. Zip all the source files into firstname.lastname.assignment2.zip
3) Submit both of your files (text document and zip file) via Canvas course web site.
4) Due Oct 8th, 11:59 PM