1. (10 Points) Trace bottom-up merge sort (see class notes for implementation details) with the following input, where merge result with sz=k should contain the result of merging adjacent sub-arrays each with size of k. The last row should contain the final sorted result.

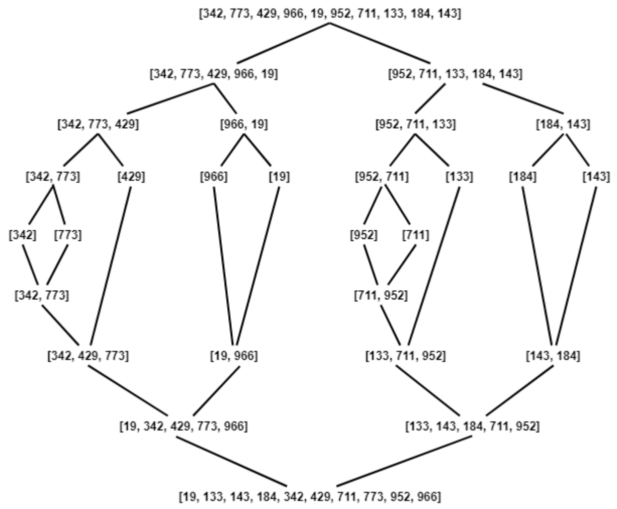
Index	0	1	2	3	4	5	6	7	8	9
Input	342	773	429	966	19	952	711	133	684	143
Merge Result sz = 1	342	773	429	966	19	952	133	711	143	684
Merge Result sz = 2	342	429	773	966	19	133	711	952	143	684
Merge Result sz = 4	19	133	342	429	711	733	952	966	143	684
Merge Result sz = 8	19	133	143	342	429	684	711	733	952	966

2. (5 Points) Does the merge method produce the proper output if two input subarrays are not in sorted order? Prove your answer or provide a counter example.

Answer:

No. For instance given these two arrays {143, 19} and {100, 80}, the results from the merge method is {100, 80, 143, 19}, which is not in sorted order.

3. (5 Points) Give the sequence of subarray sizes in top-down merge sort for N=10.
Answer: The subarray sizes used in top-down merge sort are: 5, 3, 2, 1, as shown in the following trace.



4. (5 Points) The following is another merge sort top down implementation, what is the running time and space complexity for this implementation in big-O? Briefly explain your answer.

```java
public static <T extends Comparable<T>> void sort2(T[] a) {
	sort2(a, 0, a.length - 1);
}

@SuppressWarnings("unchecked")
private static <T extends Comparable<T>>  void sort2(T[] a, int lo, int hi) {
	if (hi <= lo) return;

	T[] aux = (T[]) new Comparable[a.length];
	int mid = (lo + hi) / 2;
	sort2(a, lo, mid); // Sort left half
	sort2(a, mid + 1, hi); // Sort right half
	merge(a, lo, mid, hi, aux);
}
```

Answer:
Running time for this modified merge sort is still O(nlgn). After moving auxiliary array creation from the first method to the second method, we still have this running time functions, where C is a constant and Cn represents the running time for array creation and merge method. These running time functions will give us O(nlgn) running time.
$T(n) = 1$, when $n=1$
$T(n) = 2T(n/2) + Cn$, when $n>1$

Space complexity for this modified merge sort is O(nlgn). The recursive depth is lgn, and for each recursive call O(n) space is used due to new array creation. Thus, the space complexity is O(nlgn).

5. (5 Points) Show the partition result of this input array.
Answer:

Index	0	1	2	3	4	5	6	7	8	9
Input	342	773	429	966	19	952	711	133	684	143
a[1] and a[9] swapped	342	143	429	966	19	952	711	133	684	773
a[2] and a[7] swapped	342	143	133	966	19	952	711	429	684	773
a[3] and a[4] swapped	342	143	133	19	966	952	711	429	684	773
a[0] and a[3] swapped	19	143	133	342	966	952	711	429	684	773

6. (10 Points) Given 10 integers from 1 to 10, provide 5 permutations that each of them results in worst case running time for quick sort, which means each call to partition method can only reduce the array size by 1.
Answer:
There are more than 5 permutations can result in worst case running time, and here are 5 of them:

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 10, 2, 3, 4, 5, 6, 7, 8, 9},
{1, 10, 3, 4, 5, 6, 7, 8, 9, 2}
{10, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

7. (5 Points) Suppose that we scan over items with keys equal to the partitioning item's key instead of stopping the scans when we encounter them. With this change, what is the running time for this Quick Sort implementation given input with all equal keys?
Answer:
With this change and all keys equal input, for each partition run the left cursor will stop at where the pivot element is. Thus, the pivot element is swapped with itself and the partition method can only reduce the array size by 1. That gives us the worst-case running time, $O(n^2)$.

8. (5 Points) QuickSortRandomized uses two methods to do quick sort. The first method shuffles the input array before doing regular quick sort. The second method randomly picks a pivot element during each partition. Run QuickSortRandomizedTest five times and provide the results in the following table. From your experiments, which method runs faster?
Answer skipped.

N	Average Method1 Duration	Average Method 2 Duration
1000		
10000		
100000		
1000000		
10000000		

9. (10 Points) Implement a non-recursive version of quicksort using a stack (java.util.Stack, https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html). Provide implementation inside QuickSortNonRecursive.sort method.
Answer:

```
public static <T extends Comparable<T>> void sort(T[] a) {
	if (a!=null && a.length>1) {
		Stack<Integer> stack = new Stack<Integer>();
		stack.push(0);
		stack.push(a.length-1);
		while (!stack.isEmpty()) {
			int hi = stack.pop();
			int lo = stack.pop();
			if (hi<=lo) continue;

			int j = partition(a, lo, hi);
			stack.push(lo);
			stack.push(j-1);
			stack.push(j+1);
			stack.push(hi);
		}
	}
}
```

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 might 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.
2) Due Sep 30th, 11:59 PM