CME 2001 Assignment-1 (Comparison of Merge sort, Heapsort and Quicksort)

Due date: 02.11.2015 Monday 09:00 a.m. Late submissions aren't allowed.

Control date: Control date will be announced later. Research assistances will control your assignments in their offices. You will have 5 minutes to show your assignments.

Parallelism Control: The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in lectures.

Please do not forget to <u>bring your laptops</u> and <u>reports</u> while coming to the assignment control!

In this assignment, you are expected to implement three (Merge sort, Heapsort and Quicksort) of the well-known sorting algorithms and compare them according to the computing time and memory usage for different inputs.

Task – 1
Design and code the necessary classes for the sorting algorithms.

Task – 2
At the end of the document, there is "Test" class given for you to compute the time required for the sorting process. Since this class doesn't include any command to compute the memory usage, you should search and add it to your program. Additionally, please fill the missing parts of this class. For instance, you should create and fill the integer arrays.

Task – 3
Compare the running times (in nanoseconds) of the algorithms for different ordered arrays, with sizes of 1,000; 10,000; 100,000; 1,000,000 and fill the following table accordingly.

		Array Already Sorted (Best Case)				Random Array (Average Case)				Array Sorted in Decreasing order (Worst Case)			
		1,000	10,000	100,000	1,000,000	1,000	10,000	100,000	1,000,000	1,000	10,000	100,000	1,000,000
Running Time	**Merge sort**												
	Heapsort												
	Quicksort												
Memory	**Merge sort**												
	Heapsort												
	Quicksort												

Table 1. Comparison table

Task – 4
Prepare a scientific report in the light of the results obtained from your program and the information you learned in the class. Establish a connection between the asymptotic running time complexity (in Θ notation) and the results (in nanoseconds) of your practices. Your report should include the Comparison table and consists of the following sections:

- Abstract
- Applied Algorithms - best, avg, worst-case complexities
- Results Table (Table 1) and Assessments

- Algorithm Choice for Scenario 1 and 2

Also, you are expected to determine the appropriate sorting algorithms for the following scenarios. For each of the following scenarios, which algorithm would you choose and why? Explain your thoughts and reasons clearly and broadly.

Scenario 1: Assume that you are working in Google. You are asked to choose a sorting algorithm to sort millions of objects. An object is consisted of many other fields besides the field "Identification number", whose type is integer; and the objects will be sorted based on this field in increasing order. When you analyzed the inputs, you noticed that they are generally in random order.

Scenario 2: Assume that you are generating a program for an embedded device whose memory capacity is limited. You are asked to choose a sorting algorithm to sort an integer array of size 2000 in increasing order. When you analyzed the inputs, you noticed that they are generally in decreasing order.

Grading Policy

Item	Percentage
Merge sort	% 10
Heapsort	% 15
Quicksort	% 15
Running Time	% 20
Memory Usage	% 20
Report	% 20

Table 2. Grading Policy

Test Class

```java
import java.util.Random;
import java.lang.management.*;
public class Test {

	public static long getCPUTime() {
		ThreadMXBean bean = ManagementFactory.getThreadMXBean();
		return bean.isCurrentThreadCpuTimeSupported() ?
bean.getCurrentThreadCpuTime() : 0L;

	}
	public static void main(String[] args) {
		// Fill the necessary parts
		long startTime, endTime;
		Random rand = new Random();

		startTime = getCPUTime(); //gives CPU time in nanoseconds
	    // Sort
	    endTime = getCPUTime();

	    System.out.println("Random Ordered Sorting Time:"+ ((double)endTime-
(double)startTime) + " nanoseconds");
	}
}
```