SorterQuick Program

When the program is run you are given the option to generate an array from size 1 up to size 10,000,000. This array is generated and filled with random numbers. You are prompted with the decision to print this array or not. You are then given the choice to run one of four quicksort algorithms:

Quicksort with Median of Medians → then choose groups of 3, 5, or 7
Quicksort with Random Selection → choose 'i'
Quicksort with a Randomly chosen pivot for partition
Quicksort (simplified) with the prescribed heuristics.

Once the algorithm has finished running you have the option to print the sorted list. After this selection is made it will print the total time it took, in milliseconds, to sort the array. The user is then prompted to run the same array with the other algorithms, which involves storing a copy of the originally generated array. This allows the user to compare running times of different algorithms with the same data.

Median Finding Algorithms

Median of Medians

Median takes an array and passes groups length 3, 5, or 7 sub arrays, using indexes, to a helper function called subMedians. The subMedians method parses contents to an array corresponding to the group length. The sub array is then sorted with insertion sort and the median is derived and passed pack to the Median function where it is stored in an ArrayList until all medians in sub arrays have been found. ArrayList is used for its dynamic properties. When complete the median of the medians is found.
-Passed Parameters: array, groupSize

Randomized Median Finding Algorithm

Algorithm Randomized Select takes an array of numbers and element 'i' the i'th smallest element to search for, as arguments. This will allow for k = i, or k = i/2. It uses a recursive call that splits the array using a randomized partition, similar to quicksort. The recursive call only searches the half that should contain 'i'. The i-th smallest number is returned.
-Passed Parameters: array, index 0, index end, the i-th smallest integer to search for

Quicksort Algorithms

The first two versions of quicksort are traditional versions that use a recursive breakdown of the array by splitting the array at a pivot point given by a partition function. These partition functions call either the previously mentioned Median of Medians and Randomized Median Finding algorithms to set the pivot point. Each time partition is called a new pivot is set based on the sub array and one of these methods.

The simple quicksort method was implemented with the addition of a short method that sorts and finds the median of the three values for pivot, pivot1, pivot2, and pivot3.

Random quicksort generates a pivot point relative to the size of the array by using the nextInt() method from the Java Random class. Initially we were running this algorithm by generating all possible random numbers up to the last index of the current array which caused the algorithm to run extensive amounts of time. Adjusting the nextInt() method to start with searching at the beginning index of the are up to the length of the current array minus the index alleviated the problem.
Eg. int pivotIndex = rand.nextInt(right-left)+ left;

DATA

Intel® Core™Duo CPU T9600 @ 2.8GHz w/ 8.00 GB RAM																
	Time (t) in milliseconds (ms)															
QuickSort Algorithm	Trial 1	Trial 2	**AVG**	Trial 1	Trial 2	**AVG**	Trial 1	Trial 2	**AVG**	Trial 1	Trial 2	**AVG**	Trial 1	Trial 2	**AVG**	Trial MAX
ARRAY LENGTH	1,000	1,000	**1000.0**	10,000	10,000	**10000**	100,000	100,000	**100000**	500,000	500,000	**500000**	1,000,000	1,000,000	**1000000**	10,000,000
Median of Medians																
SIZE 3	9.0	9.0	**9.0**	19.0	43.0	**31.0**	148.0	419.0	**283.5**	1031.0	1094.0	**1062.5**	2138.0	2012.0	**2075.0**	24661.0
SIZE 5	3.0	3.0	**3.0**	11.0	9.0	**10.0**	75.0	221.0	**148.0**	561.0	531.0	**546.0**	1229.0	1271.0	**1250.0**	15211.0
SIZE 7	0.0	1.0	**0.5**	5.0	7.0	**6.0**	45.0	66.0	**55.5**	421.0	409.0	**415.0**	963.0	903.0	**933.0**	12078.0
Random Selection																
MIN	1.0	2.0	**1.5**	4.0	5.0	**4.5**	42.0	119.0	**80.5**	208.0	279.0	**243.5**	340.0	287.0	**313.5**	2534.0
MEDIAN (n/2)	5.0	1.0	**3.0**	2.0	2.0	**2.0**	21.0	21.0	**21.0**	121.0	137.0	**129.0**	323.0	279.0	**301.0**	2292.0
MAX	1.0	1.0	**1.0**	2.0	2.0	**2.0**	25.0	22.0	**23.5**	137.0	145.0	**141.0**	252.0	238.0	**245.0**	2335.0
Random Partition	2.0	4.0	**3.0**	15.0	15.0	**15.0**	116.0	116.0	**116.0**	530.0	673.0	**601.5**	1033.0	1067.0	**1050.0**	10638.0
Simplified w/ Heuristics	0.0	0.0	**0.0**	2.0	2.0	**2.0**	23.0	20.0	**21.5**	211.0	118.0	**164.5**	265.0	229.0	**247.0**	2950.0

When looking at the data we can see that in most cases the simplified quicksort with the assigned heuristics is faster. Although when we push the limits to larger sizes we can see that Random Selection used as the pivot shows faster results. In general, running Random Selection with the Median (n/2) produces results similar if not better than the Simplified quicksort which uses a similarly located pivot point. Overall median of medians does best with a larger group size. It seems that partitioning the array costs more than moving values to correct sides of the pivot.