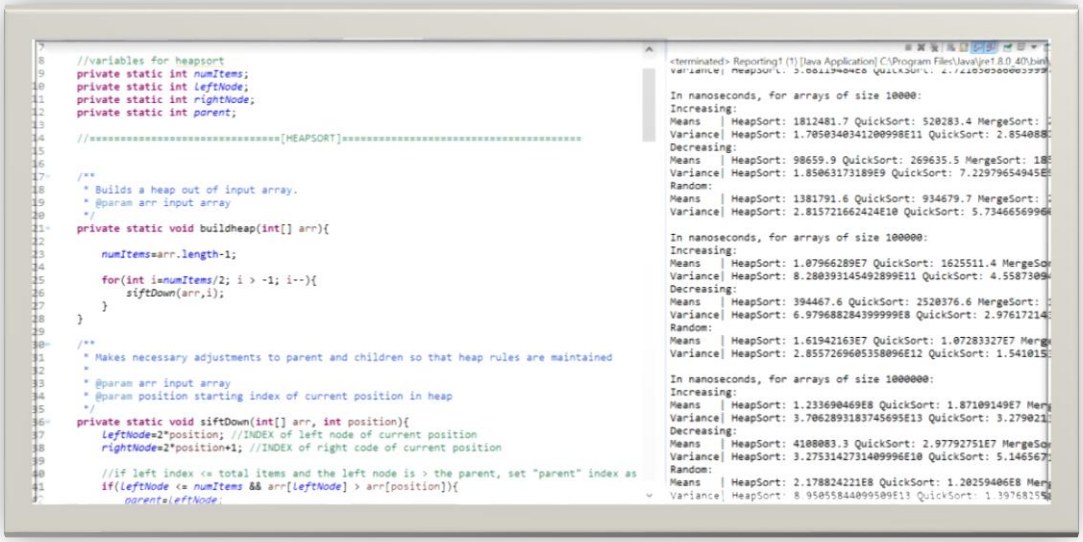
Analysis of Heapsort, Quicksort and Mergesort

Performance of three sorting algorithms on Arrays of varying size



```
//variables for heapsort
private static int numItems;
private static int leftNode;
private static int rightNode;
private static int parent;

//==============================[HEAPSORT]=====================================


/**
 * Builds a heap out of input array.
 * @param arr input array
 */
private static void buildheap(int[] arr){

    numItems=arr.length-1;

    for(int i=numItems/2; i > -1; i--){
        siftDown(arr,i);
    }
}

/**
 * Makes necessary adjustments to parent and children so that heap rules are maintained
 *
 * @param arr input array
 * @param position starting index of current position in heap
 */
private static void siftDown(int[] arr, int position){
    leftNode=2*position; //INDEX of left node of current position
    rightNode=2*position+1; //INDEX of right code of current position

    //if left index <= total items and the left node is > the parent, set "parent" index as
    if((leftNode <= numItems && arr[leftNode] > arr[position]){
        parent=(leftNode;
```

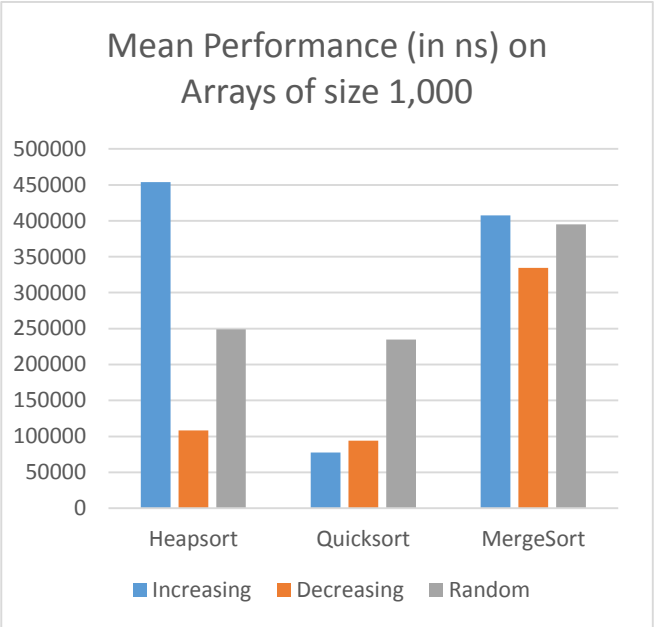
```
<terminated> Reporting1 (1) [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\

In nanoseconds, for arrays of size 10000:
Increasing:
Means    | HeapSort: 1812481.7 QuickSort: 520283.4 MergeSort:
Variance| HeapSort: 1.7050340341200998E11 QuickSort: 2.854088E
Decreasing:
Means    | HeapSort: 98659.9 QuickSort: 269635.5 MergeSort: 18
Variance| HeapSort: 1.850631731189E9 QuickSort: 7.229796549458
Random:
Means    | HeapSort: 1381791.6 QuickSort: 934679.7 MergeSort:
Variance| HeapSort: 2.815721662424E10 QuickSort: 5.7346656996

In nanoseconds, for arrays of size 100000:
Increasing:
Means    | HeapSort: 1.07966289E7 QuickSort: 1625511.4 MergeSor
Variance| HeapSort: 8.280393145492899E11 QuickSort: 4.5587309
Decreasing:
Means    | HeapSort: 394467.6 QuickSort: 2520376.6 MergeSort:
Variance| HeapSort: 6.979688284399999E8 QuickSort: 2.97617214
Random:
Means    | HeapSort: 1.61942163E7 QuickSort: 1.07283327E7 Merg
Variance| HeapSort: 2.855726960535809 6E12 QuickSort: 1.541015

In nanoseconds, for arrays of size 1000000:
Increasing:
Means    | HeapSort: 1.2336904 69E8 QuickSort: 1.87109149E7 Mer
Variance| HeapSort: 3.7062893183745695E13 QuickSort: 3.279021
Decreasing:
Means    | HeapSort: 4108083.3 QuickSort: 2.97792751E7 MergeSor
Variance| HeapSort: 3.2753142731409996E10 QuickSort: 5.146567
Random:
Means    | HeapSort: 2.178824221E8 QuickSort: 1.20259406E8 Mer
Variance| HeapSort: 8.95055844099509E13 QuickSort: 1.39768159
```

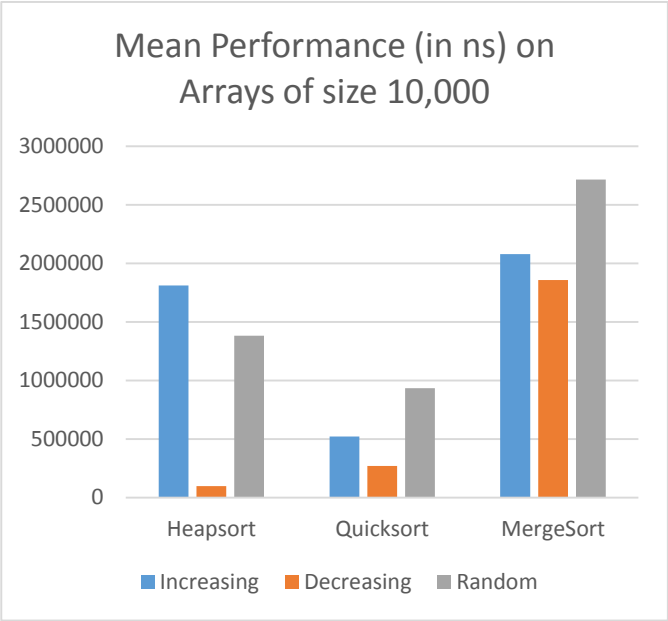

Katherine Cass

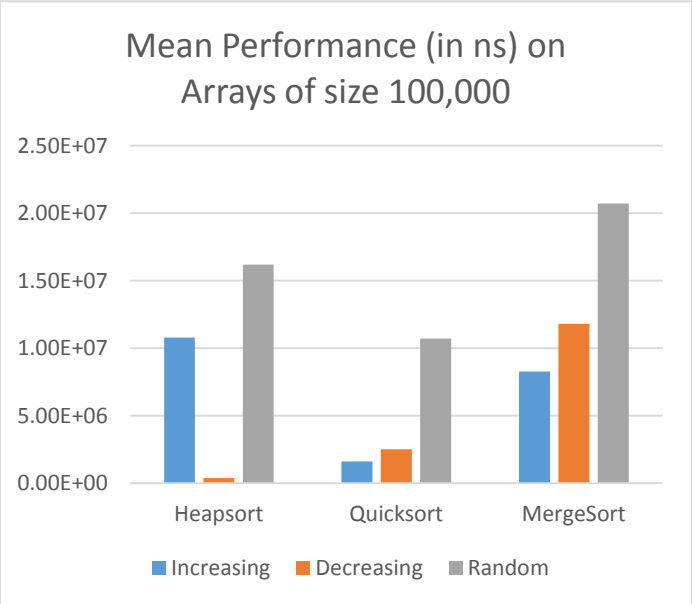
Student ID: *KRC53*

Class: *EECS 233, P4*

21 April 2015







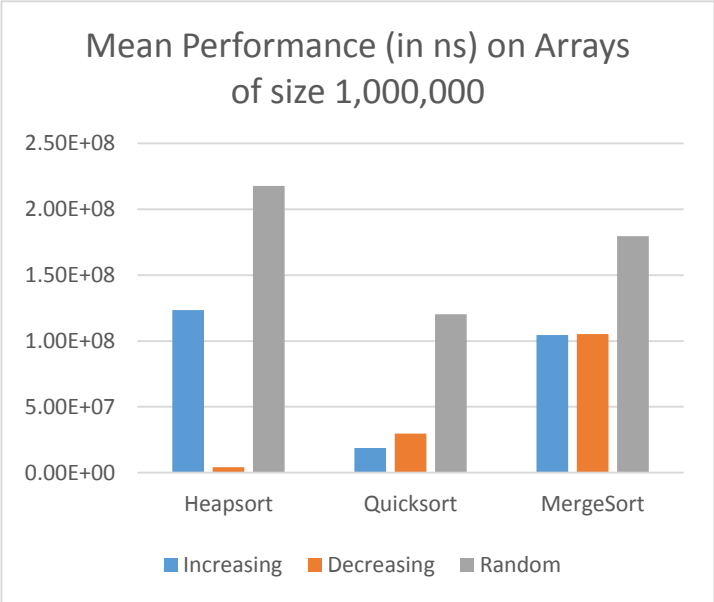


Fig. 1: performance of various sorting algorithms on arrays of size 1000, 10000, 100000, and 1000000, respectively.

Some observations based on data generated from my Reporting1 class, printed to the console and then manually entered into excel to generate these graphs (data table included below, and in "Reporting1results.txt"):

- **Heapsort** uses the most efficient number of comparisons, but requires data to be shifted around significantly, unless the data is already decreasing (then the siftdown method doesn't have to do much work). For this reason, it outperforms all other sorts on decreasing arrays of significant size. However, it preforms poorly on large random arrays and small sorted arrays.
- **Quicksort** tends to have the overall lowest runtime because it requires a low number of comparisons as well. This algorithm is generally efficient for the array datastructure, with the exception of decreasing arrays, which Heapsort outperforms all in.
- **Mergesort** is generally slow, only outperforming Heapsort when preformed on an increasing array and significantly large random arrays. This is understandable, because the data structure used for testing was an array. Mergesort works best on data structures that are not operated on in place, but instead moveable, such as linked list. For this reason, Mergesort would be a good option on very large amounts of data that require large amounts of memory.

Array Size	*Increasing Array*	*Decreasing Array*	*Random Array*
1000	**Means** HeapSort: 453826.1 QuickSort: 77533.7 MergeSort: 407596.8 **Variance** HeapSort: 1.1325483705749002E11 QuickSort: 4.177426266409999E9 MergeSort: 5.5712165204759995E10	**Means** HeapSort: 108110.9 QuickSort: 94169.3 MergeSort: 334553.5 **Variance** HeapSort: 3.8438126948999995E8 QuickSort: 1.52120861881E9 MergeSort: 4.15161135845E9	**Means** HeapSort: 249151.0 QuickSort: 234867.4 MergeSort: 395237.4 **Variance** HeapSort: 3.68119484E8 QuickSort: 2.7216305860039997E10 MergeSort: 6.546485387439999E9
10000	**Means** HeapSort: 1812481.7 QuickSort: 520283.4 MergeSort: 2078653.2 **Variance** HeapSort: 1.7050340341200998E11 QuickSort: 2.8540881568764E11 MergeSort: 3.9411250140095996E11	**Means** HeapSort: 98659.9 QuickSort: 269635.5 MergeSort: 1857641.9 **Variance** HeapSort: 1.85063173189E9 QuickSort: 7.22979654945E9 MergeSort: 2.114854258509E10	**Means** HeapSort: 1381791.6 QuickSort: 934679.7 MergeSort: 2717182.8 **Variance** HeapSort: 2.815721662424E10 QuickSort: 5.734665699609999E9 MergeSort: 4.382130224211599E11
100000	**Means** HeapSort: 1.07966289E7 QuickSort: 1625511.4 MergeSort: 8293059.1 **Variance** HeapSort: 8.280393145492899E11 QuickSort: 4.55873094124E9 MergeSort: 2.1933390109392898E12	**Means** HeapSort: 394467.6 QuickSort: 2520376.6 MergeSort: 1.18210734E7 **Variance** HeapSort: 6.979688284399999E8 QuickSort: 2.976172143904E10 MergeSort: 4.464553954496984E13	**Means** HeapSort: 1.61942163E7 QuickSort: 1.07283327E7 MergeSort: 2.07184495E7 **Variance** HeapSort: 2.8557269605358096E12 QuickSort: 1.5410153567681003E11 MergeSort: 3.777813188559425E13
1000	**Means** HeapSort: 1.233690469E8 QuickSort: 1.87109149E7 MergeSort: 1.045447179E8 **Variance** HeapSort: 3.7062893183745695E13 QuickSort: 3.2790213132889E11 MergeSort: 8.591474515040539E14	**Means** HeapSort: 4108083.3 QuickSort: 2.97792751E7 MergeSort: 1.051858562E8 **Variance** HeapSort: 3.2753142731409996E10 QuickSort: 5.14656717320089E12 MergeSort: 1.3596272642273755E15	**Means** HeapSort: 2.178824221E8 QuickSort: 1.20259406E8 MergeSort: 1.797325111E8 **Variance** HeapSort: 8.95055844099509E13 QuickSort: 1.3976825586954E12 MergeSort: 3.7404862719378484E13

Table 1. Output from Reporting1.java, in nanoseconds