Assignment 2 Sorting with Heap

Ertuğrul Yılmaz
Student ID: 041701030
COMP 201 Data Structures and Algorithms
Date: 08.12.2019

1. Algorithm Explanation

In this assignment, I had made an algorithm which displays the timings of different sorting algorithms for different cases. Main class has three different algorithms Heap Sort, Java Sort and Selection Sort. I wrote Heap Sort and Selection Sort as methods in main class. And no need to write anything for Java Sort. I measured the timing by using *.nanoTime()*. I declared and initialized three different arrays size of 10000,40000 and 80000.
I initialized those arrays with random numbers 0 to 10000000. To generate random number I used *util.Random* library. I obtained the averages by sorting same array size for same sorting algorithm 10 times. And I compared them with each other.

Most of the cases Java Sort is better than Selection Sort and Heap Sort only if array size is bigger than 40000. But most cases Heap Sort is better than Java Sort and Selection Sort only if array size is smaller than 10000.

Arrays.sort() method is using two different sorting algorithms. For primitive types it uses Dual-Pivot implementation of Quick Sort. For objects it uses iterative implementation of Merge sort. The algorithm of Merge Sort is simple algorithm divides the array in two recursively. After sorting merge them recursively.

The time complexity of Selection Sort is $O(n^2)$.
The time complexity of building a heap is O(n) and removing from root is O(logn) that because the time complexity of Heap Sort is O(nlogn).
The time complexity of Merge Sort is O(nlogn).

2. Sample Outputs

In this algorithm, I checked four different java codes
1. Heap Sort, Java Sort and Selection Sort averages for array size of 10000. (Figure 1).
2. Heap Sort, Java Sort and Selection Sort averages for array size of 40000. (Figure 2).
3. Heap Sort, Java Sort and Selection Sort averages for array size of 80000. (Figure 3).

Figure 1

```
Array Size: 10000

 1: 58,02 msec
 2: 5,84  msec
 3: 4,00  msec
 4: 4,13  msec
 5: 3,55  msec
 6: 2,24  msec
 7: 2,85  msec
 8: 3,01  msec
 9: 2,99  msec
10: 2,38  msec
Average Sort Time for Heap Sort              : 8,90  msec

 1: 41,57 msec
 2: 49,05 msec
 3: 47,59 msec
 4: 31,54 msec
 5: 26,85 msec
 6: 25,19 msec
 7: 21,50 msec
 8: 20,53 msec
 9: 22,53 msec
10: 23,27 msec
Average Sort Time for Java Array Sort        : 30,96 msec

 1: 101,02 msec
 2: 62,01  msec
 3: 250,73 msec
 4: 258,45 msec
 5: 251,86 msec
 6: 253,00 msec
 7: 253,06 msec
 8: 242,97 msec
 9: 246,28 msec
10: 250,03 msec
Average Sort Time for Selection Sort         : 216,94 msec
```

```
Array Size: 40000

 1: 13,97 msec
 2: 11,96 msec
 3: 9,79  msec
 4: 10,77 msec
 5: 10,89 msec
 6: 14,71 msec
 7: 13,31 msec
 8: 12,11 msec
 9: 11,48 msec
10: 9,98  msec
Average Sort Time for Heap Sort              : 11,90 msec

 1: 10,44 msec
 2: 7,33  msec
 3: 6,26  msec
 4: 7,67  msec
 5: 9,94  msec
 6: 7,57  msec
 7: 6,89  msec
 8: 6,95  msec
 9: 8,12  msec
10: 6,70  msec
Average Sort Time for Java Array Sort        : 7,78  msec

 1: 3855,35 msec
 2: 4052,19 msec
 3: 3907,36 msec
 4: 3874,38 msec
 5: 3872,67 msec
 6: 3868,14 msec
 7: 3873,97 msec
 8: 3873,83 msec
 9: 3856,03 msec
10: 3864,53 msec
Average Sort Time for Selection Sort         : 3889,84 msec
```

Figure 2

```
Array Size: 80000

 1: 22,05 msec
 2: 20,99 msec
 3: 27,40 msec
 4: 24,11 msec
 5: 23,66 msec
 6: 23,75 msec
 7: 25,10 msec
 8: 25,33 msec
 9: 22,40 msec
10: 23,45 msec
Average Sort Time for Heap Sort              : 23,82 msec

 1: 24,09 msec
 2: 17,29 msec
 3: 17,14 msec
 4: 16,89 msec
 5: 14,45 msec
 6: 15,27 msec
 7: 15,97 msec
 8: 16,57 msec
 9: 15,92 msec
10: 15,21 msec
Average Sort Time for Java Array Sort        : 16,88 msec

 1: 15311,25 msec
 2: 15688,00 msec
 3: 16907,28 msec
 4: 16621,51 msec
 5: 16347,03 msec
 6: 16380,47 msec
 7: 16586,47 msec
 8: 17318,58 msec
 9: 16765,19 msec
10: 17152,07 msec
Average Sort Time for Selection Sort         : 16507,78 msec
```

Figure 3