CME 2204 Assignment-1

Comparison of Heapsort, Shellsort and Introsort

Report

	EQUAL INTEGERS			RANDOM INTEGERS			INCREASING INTEGERS			DECREASING INTEGERS		
	1,000	10,000	100,000	1,000	10,000	100,000	1,000	10,000	100,000	1,000	10,000	100,000
heapSort	0	1	2	0	1	9	0	1	6	0	1	5
shellSort	0	2	2	0	2	11	0	1	2	0	0	3
introSort	1	4	5	1	4	19	0	1	9	0	1	8

HeapSort is an in-place, not divide-and-conquer sorting algorithm. HeapSort have O(nlog(n)) time complexity for all cases and it doesn't require many recursions or multiple arrays to work. Works with arrays and trees.

ShellSort is an in-place, divide-and-conquer sorting algorithm. ShellSort have O(nlog(n)) time complexity on average but the worst-case efficiency is O(n^2). The complexity depends on the interval chosen.

IntroSort is an in-place, divide-and-conquer, massively recursive sorting algorithm, similar with QuickSort. Difference between them is that IntroSort starting with QuickSort and if the recursion depth goes more than a particular limit it switches to Heapsort to avoid Quicksort's worse case O(n^2) time complexity. It also uses insertion sort when the number of elements to sort is quite less. The efficiency of the algorithm directly depends on which element is chosen as a pivot element.

Based on the measurements given in the table above we can make a comparison between these three sorting algorithms.

If we consider small data sets such 1000 elements in the array regardless of the order of elements (equal, random, increasing or decreasing) all 3 algorithms work equally fast. As the size of a data set starts to increase, the difference in speed between the algorithms becomes more obvious.

The table shows that the best algorithm to sorting big data sets with equal or random elements is HeapSort. Due to constant time complexity HeapSort perform well under all conditions. A little bit slower works ShellSort algorithm. IntroSort shows worst result.

When it comes to deal with already sorted data (in increasing or decreasing order) things are changes. In this case, the best performance shows ShellSort, since ordered input is a best case for ShellSort. In second place HeapSort and third place IntroSort.

IntroSort always takes longer time to execute. I think the reason is:
Inappropriate pivot selection. Algorithm takes last element in the array as a pivot point every time. This way in already ordered arrays pivot number will be always smallest or biggest number, which in turn causes massive recursion. And as we know recursion quite costly. Random selection of the pivot number can solve this problem.

Scenario: We aim to place students at universities according to their central exam grades and department preferences. If there are millions of students in the exam, which sorting algorithm would you use to do this placement task faster?

Answer: Based on the scenario we have unsorted array with millions random elements.
I would prefer to apply HeapSort algorithm. Though, this algorithm not the best and fastest algorithm, but I can rich constant time complexity which is O(nlogn). Also, it is in-place algorithm what means it doesn’t require additional space.
Theoretically IntroSort could be used as well, since it is in-place too and works with constant time complexity which is O(nlogn), but practically IntroSort algorithm performs slower than HeapSort.

Source code for sorting algorithms taken from:
HeapSort - https://www.geeksforgeeks.org/heap-sort/
IntroSort - https://www.geeksforgeeks.org/introsort-or-introspective-sort/
ShellSort - https://www.geeksforgeeks.org/shellsort/