DOKUZ EYLUL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CME 2204 Assignment-1

Comparison of Heapsort, Shellsort and Introsort Report

by
ELİF ÖZKER
2019510094

IZMIR
9.04.2022

1) Time Complexity Table:

	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	**0.401963 ms**	**1.6496737 ms**	**8.014955 ms**	**1.545798 ms**	**7.036543 ms**	**31.57786 ms**	**0.869018 ms**	**4.614251 ms**	**21.766574 ms**	**1.015300 ms**	**3.998263 ms**	**19.839837 ms**
shellSort	**0.35348555 ms**	**3.274874 ms**	**18.294284ms**	**0.95481 ms**	**9.472968 ms**	**35.1259577 ms**	**0.441817 ms**	**4.6598844 ms**	**18.312693 ms**	**0.58958 ms**	**5.302297 ms**	**18.4094878 ms**
introsort	**2.28458ms**	**14.4633 ms**	**28.06963 ms**	**1.379603 ms**	**5.1219584 ms**	**26.3427345 ms**	**1.07194425 ms**	**4.84459165ms**	**22.18662485ms**	**1.5996189ms**	**6.7032947ms**	**24.2497025ms**

2) ShellSort

Shell sort is an enhanced version of Insertion Sort.It sorts the elements that are far apart at first and decreases the spacing between the elements to be sorted.It continues these steps until the gap is 1.

For shell sort, the larger the size of the array, the larger the number of elements we compare, so our runtime will increase.So we can see the time values getting bigger as 1000 goes towards 10,000 and 100,000.

Best Case:O(nlgn): The best case is when the array is already sorted, ie increasing. We can see this from the table as well. The reason for this is that the total number of comparisons for each range is equal to the size of the array when it is sorted.And in addition, it won't go inside the third for loop at all.This makes it fast in terms of time.

Worst Case:O(n^2): In the worst case the Shell sort basically an insertion sort so it approaches O(n^2) time complexity. Worst case occurs when decreasing order. In this case, each loop will swap the data. This causes the inner for loop to run n times. For the outer for loop, we get O(n^2) because n is running.

Equal Integers

Again, it will be the same as in the increasing state. It will be the best case. It will not go into the innermost for loop and the swap operation will not take place. Therefore, it will cause the code to run faster.

Random Integers

This will be average case for Shell sort which is O(nlgn). The reason for being an average case is that some elements are already ordered and some are not.

3) HeapSort

Heapsort creates a max heap using the given array.Max-heap represents the ordering of array in which the root element represents the maximum element of the array.

Generally, it consists of two stages:

Creates a heap h using the elements of the array.
It deletes the root element of the heap formed in the 1st stage, one by one, writes the smallest element in its place and creates the max heap again.

Complexity of HeapSort

Generating a max-heap from an unordered list requires **O(n)** calls to the max heapify,each of which takes **O(lgn)** time so the running time of the HeapSort is **O(nlgn)**.
Since we are sending an array to HeapSort, the more the input size increases, the more root and parent we need to compare thus the running time will increase. So we can see the time values getting bigger as 1000 goes towards 10,000 and 100,000.

Decreasing Order , Increasing Order

If the input data is given ascending order there is no sorting required so array is already sorted but even though the heapsort is already sorted the algorithm swaps all of the elements to order the array. For the same input sizes, whether increasing or decreasing, they will give values close to each other, since they are O(nlgn).Therefore we do not see much time difference between increasing and decreasing order.

Random Integers
Average case will occur when all elements random.We will have to call max heapify every time and it will take log(n) time for swap operations and it will take O(nlogn) time again since there are n elements.

Equal Integers
The reason why it takes less time in the equal state is that it works faster in the heapify function because it does not go inside the if block (largest != i).It takes linear time which is O(n).

4)**IntroSort**

As a hybrid sorting algorithm, Introsort uses three sorting algorithms to reduce running time. These are **QuickSort, HeapSort and InsertionSort**. Introsort starts with Quick Sort, if the recursion depth exceeds the depth limit of the introsort, it switches to Heapsort, thus avoiding the worst case of QuickSort O(n^2). When the number of elements is few, insertion sort is activated.

For Intro Sort, the larger the size of the array, the larger the number of elements we compare, so our runtime will increase.So we can see the time values getting bigger as 1000 goes towards 10,000 and 100,000.

It first create partition and then:

1) If the partition size is greater than the depthlimit i.e 2*log(n),switching to heap sort.
2) If the partition size is too small,switching to insertion sort.The partition size is less than 16 we will do insertion sort.
3) If the partition size is greater than 16 and less than depthlimit (2*log(n)),Quicksort is activated.

Complexity of IntroSort
Time complexity of IntroSort is O(nlg(n)) for all cases.

Equal Integers
It is inefficient for array that has contains the same elements.It will be the worst case.

Increasing ,Decreasing Order
The best case is when it is already sorted.

Random Integers
It will be the average case for IntroSort.

5) Comparison Of ShellSort,HeapSort,IntroSort

When all elements are equal , heapSort enters linear time state.Among these three algorithms, it would be best to use heapsort. Also in the table, heapsort has the fastest runtime for the equal state.

It would be unreasonable to use shellsort for random values. Since it is O(n^2) in the worst case scenario, we get the longest run time in Shell sort, but intro sort works faster than shellsort for random situations because of worst case runtime is O(nlg(n)) which is faster than O(n^2).

Introsort,heapsort and Shell sort the all give values close to each other for increasing order because they all have base case time complexity which is O(nlg(n)).

Shellsort and heapsort give the best values, since the best case situation will occur in the increasing order or sorted, that is, it will be O(n(lgn)). There is not a big difference between the two, but according to the table, shellsort worked faster.

Shellsort is slower than heapsort beacuse of worst case scenerio.

IntroSort and heapsort output values pretty close to each other. This is because they both have O(nlgn) time complexity for increasing and decreasing order, so they show the same values for the same input numbers.

6)SCENERIO

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: Students' grades are unlikely to come in sequential order. Most likely, grades will be mostly different and random because there are millions of students. In case of random number, intro sort works faster than heapsort and shellsort. Therefore, it would be logical to choose the introsort for this scenario.

7) REFERENCES

[1] Reference for HeapSort: *https://www.geeksforgeeks.org/heap-sort/*
[Accessed 30.03.2022]
[2] Reference for ShellSort: *https://www.geeksforgeeks.org/shellsort/*
[Accessed 2.04.2022]
[3] Reference for IntroSort: *https://www.geeksforgeeks.org/introsort-or-introspective-sort/* *[Accessed 4.04.2022]*