IDAA432C

ASSIGNMENT-4

BY

- ADITYA VALLABH
- NEIL SYIEMLIEH
- KIRAN VELUMURI
- TAUHID ALAM

QUESTION

- **Given a set of numbers, write an efficient Heap Sort algorithm to report the required (ascending/descending) sorted sequence. Trace the movement of every element of the Heap during the execution of your algorithm.**

ALGORITHM DESIGN

BINARY HEAP

A Binary Heap is a Binary Tree with following properties.

- It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible).
- A Binary Heap is either Min Heap or Max Heap.

  In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree.
  Similarly, in a Max Binary Heap, the key at the root must be the maximum among all the keys present in the heap.

ARRAY TRAVERSAL

Since a Binary Heap is a Complete Binary Tree,
it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I (0-indexed)

- The left child will be at 2 * I + 1.
- The right child will be at 2 * I + 2

HEAPIFY

- Heapify will be used to maintain the heap, we can give any index i and from index i it will go forward to its children and balance the heap and so on.
- We can use this algorithm to build the heap too.
- We call Heapify from the last but one level of the tree till the root.
- This will build the heap which is required for sorting.

Steps:

1)IT FINDS IF ANY BIGGER VALUE IS AVAILABLE IN CHILDREN NODES

2) THEN IT SWAPS THE ROOT WITH THE CHILD

3) CALLS HEAPIFY AT THE CHILD TO MAINTAIN THE HEAP PROPERTY RECURSIVELY

Algorithm 1 Heapify Algorithm

Input: $arr, heads, n, idx$

$largest \leftarrow idx$

$left \leftarrow 2 * idx + 1$

$right \leftarrow 2 * idx + 2$

if $left < n$ *and* $arr[left] > arr[largest]$ **then**

    $largest \leftarrow left$

if $right < n$ *and* $arr[right] > arr[largest]$ **then**

    $largest \leftarrow right$

if $largest \neq idx$ **then**

    $swap(arr, heads, idx, largest)$

    $heapify(arr, heads, n, largest)$

SORTING

- Finally we are ready to sort the array, because our top element is the biggest in max-heap.
- We will start replacing these elements with bottom elements and will call Heapify in every iteration reducing the size of heap, to sort in increasing order, just like that.

Algorithm 2 Heap Sort Algorithm

Input: $arr, heads, n, idx$

for $i \leftarrow \frac{n}{2} - 1$ to 0 **do**

    $heapify(arr, heads, n, i)$

for $i \leftarrow n - 1$ to 0 **do**

    $swap(arr, heads, 0, i)$

    $heapify(arr, heads, i, 0)$

TRACKING MOVEMENT

- Associate a linked list with every array element.
- Head of linked list sits in another array parallel to the input array.
- When elements are swapped, swap their heads too.
- Update both elements' linked lists with:
- New location
- Iteration at which swap happened

Algorithm 3 Swap Algorithm

Input: $arr, heads, it, a, b$

$heads[a] \leftarrow insertNode(heads[a], b)$

$heads[b] \leftarrow insertNode(heads[a], a)$

$tmp \leftarrow arr[a]$

$arr[a] \leftarrow arr[b]$

$arr[b] \leftarrow tmp$

$temp \leftarrow heads[a]$

$heads[a] \leftarrow heads[b]$

$heads[b] \leftarrow temp$

COMPLEXITY ANALYSIS

COMPUTATION TIME(HEAPIFY):

The method Heapify this will run from root to children and from the children to their children and so on, so the complexity of this method will be O(log n) or O(h) where h is height of node in tree.

COMPUTATION TIME(BUILD HEAP):

$$
\begin{aligned}
T(n) &= \sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) \\
&= n \sum_{h=0}^{\log n} \left\lceil \frac{h}{2^{h+1}} \right\rceil = n \sum_{h=0}^{\infty} \left\lceil \frac{h}{2^{h+1}} \right\rceil \\
&= n * \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = n * 2 \\
&= O(n)
\end{aligned}
$$

COMPUTATION TIME(HEAP SORT):

The loop in heapSort is first building the heap and then just swapping elements and running from 1 to size with Heapify method inside which complexity is O(log n) so the total complexity will be

$$T(n) = O(n) + O(\log n) * O(n)$$
$$= O(n \log n)$$

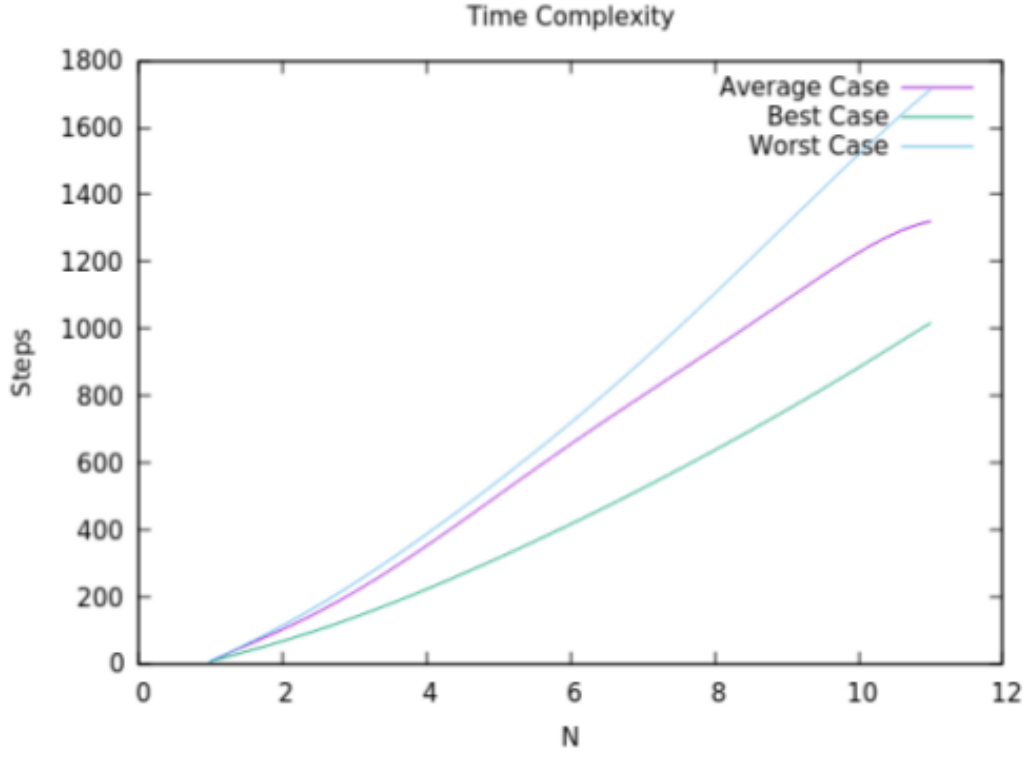
TIME COMPLEXITY

There is no early exit or latest exit in heap sort so the worst and the best case complexities would be equal.

$O(n \log n)$	$\Theta(n \log n)$	$\Omega(n \log n)$
Worst	Average	Best

EXPERIMENTAL STUDY

TIME COMPLEXITY



Best $\Omega(n \log n)$

Average $\Theta(n \log n)$

Worst $O(n \log n)$

CONCLUSION

- We have implemented the heap sort algorithm, which has Θ(n.logn) time complexity in all the cases, to sort the given set of unsorted numbers.
- For tracking the movement of the elements, linked lists have been used, which has a the space complexity of Θ(n) in all cases.


THANK
YOU