Trace the Movement of Elements & Find the Most Stable and Least Stable Elements During Heap Sort

Members:

IRM2014003
IHM2014502
IIT2013169

Problem

- Given an array representation of binary max-heap. Trace the movement of elements during heap sort and Find the Most Stable and Least Stable Elements During Heap Sort
- Our algorithm will trace movement of each element and prints movement path and finds the most stable and least stable element among all elements during heap sort.

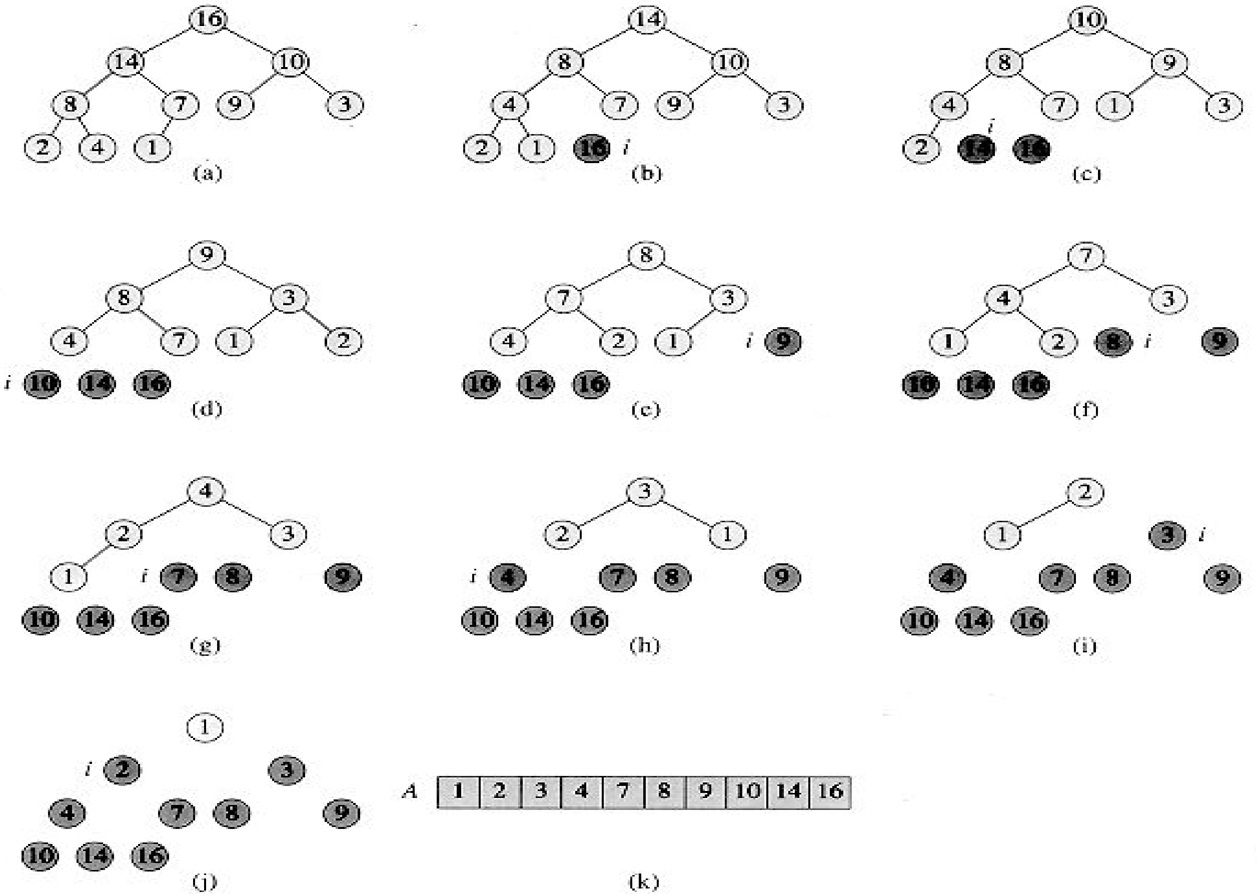
Example

Given heap

16,14,10,8,7,9,3,2,4,1

Least stable element-> 1

Most stable element-> 16



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k)

Algorithm 1 Heap Sort

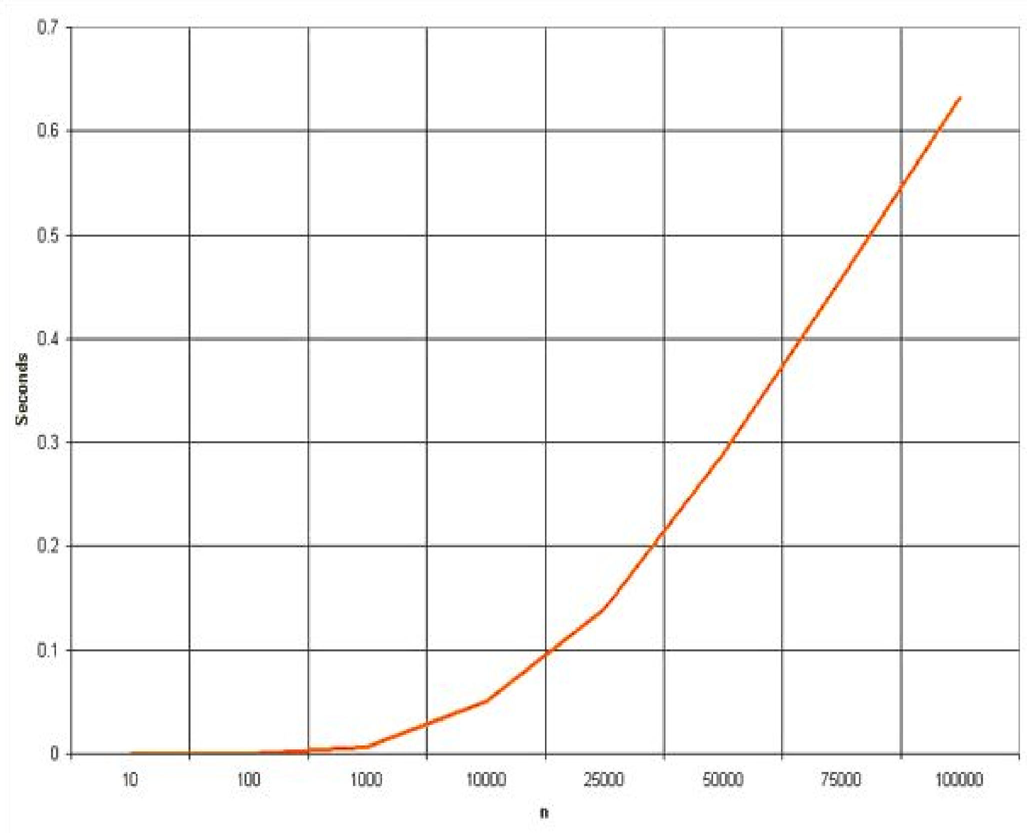
```
procedure MAIN
    input number of elements: $n$
    input array: $arr$
    initialize the list: $path$
    build_max_heap(arr, n)
    print sorted output
    find_stable(arr, n)

function BUILD_MAX_HEAP (arr, n)
    $i = n/2 - 1$
    while $i >= 0$ do
        max_heapify (arr, n, i)
        $i - -$
    $i = n - 1$
    while $i >= 0$ do
        path[i].add(0)
        path[0].add(i)
        swap(arr[i], arr[0])
        max_heapify (arr, i, 0)
        $i - -$

function MAX_HEAPIFY (arr, n, i)
    $max \leftarrow i$
    $l \leftarrow 2 * i + 1$
    $r \leftarrow 2 * i + 2$
    if $l < n$ and $arr[l] > arr[max]$ then
        $max \leftarrow l$
    if $r < n$ and $arr[r] > arr[max]$ then
        $max \leftarrow r$
    if $max \; != i$ then
        path[i].add(max)
        path[max].add(i)
        swap(arr[max],arr[i])
        max_heapify (arr, n, max)

function FIND_STABLE (arr, n)
    $max\_path \leftarrow INT\_MAX$
    $min\_path \leftarrow INT\_MIN$
    for each element ∈ path do
        Compare with path[element].size() and obtain most stable and least stable elements
    print all the most stable and least stable elements
```

Time Complexity

- The time complexity of the above algorithm is O(Nlog(N)) + O(N) for heap sort and finding the stable elements respectively, thus the overall time complexity is O(Nlog(N)). The space complexity is N for the array and the list containing the path which will be N*(length of each element path), hence giving a space complexity O(N).

Runtime Analysis

As the number of inputs increases the time complexity increases propotionaly as a funcion of N * log(N). Figure shows the plot of Time vs the Number of input elements



Thank you