elements & assu...

Complexity analysis :-

Recurrence Relation for Max heapify function

$$T(n) = T(2n/3) + O(1)$$

$T(2n/3)$: Time to run max heapify. The children's subtrees each have size of atmost $(2n/3)$

$O(1)$: Time to fix relationship b/w $A[i]$ & its children

The worst case occurs when the last row of tree is exactly half full.

Using Masters Theorem where $a = 1$, $b = 3/2$, $c = 0$ in $T(n) = aT(n/b) + O(n^c)$

$$\therefore T(n) = O(n^c \log_2 n) = O(\log_2 n)$$

The time complexity of the Build Heap function is:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left[\frac{n}{2^{h+1}}\right] O(h) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n)$$

∴ The Heapsort procedure takes $O(n \log_2 n)$ time

- as the call to build heap takes $O(n)$ time
- each of the n-1 calls to heapify takes $O(\log n)$

Experiment-6

AIM:- Write a program to analyse the time complexity of the heap sort algorithm.

Software used :- Codeblocks.

Theory:-
Algorithm

heapify (int arr[ ], int n, int i)

set largest ← i, l = 2*i+1, r = 2*i+2;
if (l < n and arr[l] > arr[largest])
set largest = l
end if
if (r < n and arr[r] > arr[largest])
set largest = r
endif.
if (largest != i)
swap (arr[i], arr[largest]);
heapify (arr, n, largest)
endif

end heapify

```
heapsort ( int arr [] , int n)
    loop i = n/2 - 1 → 0
        heapify (arr, n, i)
    end loop
        loop i = n-1 → 0
        swap (arr [0 ], arr[i]);
        heapify (arr, i, 0)
        end loop
    end heap sort
```