ANALYSIS OF HEAPSORT ALGORITHM

HEAP CREATION WITH INSERT: $T(n) = O(n \log n)$

We have first of all the heap creation process which is $N\log N$ time complexity using insert..

First we will consider the number of nodes in a binary tree at a level l---

<table>
<thead>
<tr>
<th>Level of the tree</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Let us consider the height $h$ of a binary tree of $n$ nodes:

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Height</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$\log_2(1+1)$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$\lceil \log_2(2+1) \rceil$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>$\log_2(3+1)$</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>$\lceil \log_2(4+1) \rceil$</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>$\lceil \log_2(4+1) \rceil$</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>$\lceil \log_2(4+1) \rceil$</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>$\lceil \log_2(4+1) \rceil$</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>$\lceil \log_2(5+1) \rceil$</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>$\lceil \log_2(5+1) \rceil$</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$\cdots$</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$N$</td>
<td>$\cdots$</td>
<td>$\lceil \log_2(n+1) \rceil$</td>
</tr>
</tbody>
</table>
A node inserted at level \( l \) has to move up \( l-1 \) levels at most.

\( 2^{l-1} \) nodes at level \( l \) have to move up \( l-1 \) levels at most.

For \( n \) nodes the number of levels is \( \text{Ceil}(\log_2(n+1)) \).

\[
T(n) = \text{for level from } 1 \text{ to } \text{Ceil}(\log_2(n+1)) \text{ do}
\]

\[
\text{Move } 2^{\text{levels}-1} \text{ nodes levels-1 distance}
\]

\[
= \sum_{1<=\text{levels}<\text{maxlevels}}(\text{levels}-1) 2^{\text{levels}-1} \text{, for maxlevels = Ceil}(\log_2(n+1))
\]

Now consider the geometric progression]

\[
1 + r + r^2 + r^3 + \cdots + r^n = (r^{n+1} - 1)/(r - 1)
\]

Differentiating both sides

\[
1 + 2r + 3r^2 + 4r^3 + \cdots + nr^{n-1} = d/dr[(r^{n+1} - 1)/(r - 1)]
\]

Multiplying both sides by \( r \)

\[
r + 2r^2 + 3r^3 + \cdots + nr^n = r d/dr[(r^{n+1} - 1)/(r - 1)]
\]

\[
= [r/(r-1)](n+1) r^n - [[(r^{n+1} - 1)/(r-1)^2]
\]
\[
T(N) = [r + 2r^2 + 3r^3 + \ldots + nr^n]_{r=2, n=\text{number of levels}}
\]

\[
= 2[n+1] 2^n - [(2^{n+1} - 1)]
\]

\[
= n2^{n+1} + 1
\]

\[
< 2n2^{n+1} + 1
\]

\[
= O(n2^n)
\]

\(n = \text{number of levels}\)

\[= \text{Ceil}(\log_2(n+1))\]

So \(T(N) = O(N \log N)\) is the time for heap creation.

**HEAP CREATION WITH HEAPIFY: \(T(n) = O(n)\)**

If there are \(n\) nodes in the heap then the height of the tree is levels =

\[\text{Ceil}(\log_2(n+1))\]

At a level there are \(2^{\text{level}-1}\) nodes, and a node at a level, has to migrate \(\text{level}\)-\(\text{level}\) times.
Hence the time complexity of heapify is

\[ T(n) = [1 \text{ node at level 1 has to migrate levels-1 times}] \]
\[ + [2 \text{ nodes at level 2 have to migrate levels-2 times}] \]
\[ + [4 \text{ nodes at level 3 have to migrate levels-3 times}] \]
\[ + \cdots \]
\[ + [2^{i-1} \text{ nodes at level I have to migrate levels-i times}] \]

\[ = \sum_{1 \leq i \leq \text{levels}} 2^{i-1}(\text{levels}-i) \]
(Let \( j = \text{levels}-i \) then \( i = j + \text{levels} \) and \( i-1 = \text{levels}-j-1 \))

\[ = \sum_{1 \leq j \leq \text{levels}-1} j2^{\text{levels}-j-1} \]

\[ = 2^{\text{levels}-1} \sum_{1 \leq j \leq \text{levels}-1} j2^{-j} \]
\[ = 2^{\text{levels}-1} \sum_{1 \leq j \leq \infty} j2^{-j} \]

[Consider the infinite geometric progression]

\[ \frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots \]

Differentiating both sides we have

\[ \frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + 5x^4 + \cdots \]

Multiplying both sides by
\[
\frac{x}{1-x}^2 = x + 2x^2 + 3x^3 + 4x^4 + \cdots
\]

Putting \( x = 1/2 \) we have

\[
2 = \sum_{1 \leq j \leq \infty} j^2
\]

So \( T(n) < 2n = O(n) \)

**HEAPSORT:** \( T(n) = O(n \log n) \)

Inserting an element in a heap takes \( O(\log N) \) time

Deleting an element from a heap takes \( O(\log N) \) time

In heapsort we delete \( N \) elements one after another so the time taken is \( O(N \log N) \).