Lower Bound Theory(Decision Tree)

Lower Bound Theory Concept is based upon the calculation of minimum time that is required to execute an algorithm is known as a lower bound theory or Base Bound Theory.

Lower Bound Theory uses a number of methods/techniques to find out the lower bound.

Aim: The main aim is to calculate a minimum number of comparisons required to execute an algorithm.

Techniques:

The techniques which are used by lower Bound Theory are:

1. Comparisons Trees.
2. Oracle and adversary argument
3. State Space Method

1. Comparison trees:

In a comparison sort, we use only comparisons between elements to gain order information about an input sequence (a1; a2......an).

Given a_i,a_j from (a_1, a_2.....a_n)We Perform One of the Comparisons

- $a_i < a_j$ less than
- $a_i \leq a_j$ less than or equal to
- $a_i > a_j$ greater than
- $a_i \geq a_j$ greater than or equal to
- $a_i = a_j$ equal to

To determine their relative order, if we assume all elements are distinct, then we just need to consider $a_i \leq a_j$ '=' is excluded &, ≥,≤,>,< are equivalent.

Consider sorting three numbers a1, a2, and a3. There are 3! = 6 possible combinations:

1. (a1, a2, a3), (a1, a3, a2),
2. (a2, a1, a3), (a2, a3, a1)
3. (a3, a1, a2), (a3, a2, a1)

The Comparison based algorithm defines a decision tree.

Decision Tree:

A decision tree is a full binary tree that shows the comparisons between elements that are executed by an appropriate sorting algorithm operating on an input of a given size. Control, data movement, and all other conditions of the algorithm are ignored.

In a decision tree, there will be an array of length n.

So, total leaves will be n! (I.e. total number of comparisons)

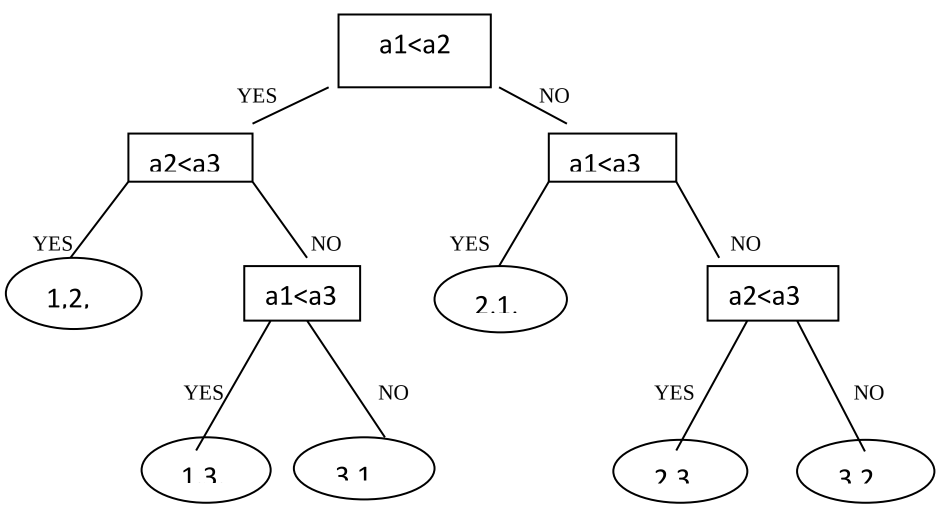
If tree height is h, then surely

    $n! \leq 2^n$ (tree will be binary)

Taking an Example of comparing a1, a2, and a3.

Left subtree will be true condition i.e. $a_i \leq a_j$

Right subtree will be false condition i.e. $a_i > a_j$



So from above, we got

$N! \leq 2^n$

Taking Log both sides

log n! <= h log2

h log2 >=log n!

h >= log2 [1,2,3….n]

h>=$\log_2 1$ +$\log_2 2$+$\log_2 3$+……..+$\log_2 n$

h>=$\sum_{i=1}^{n} \log_2 i$

h>=$\int_{i}^{n} \log_2 i - 1\, di$

h>=$\log_2 i\ .x^0 \int_1^n - \int_1^n \frac{1}{i}$xi di

h>=n $\log_2 n$ -$\int_1^n 1\, di$

h>=n $\log_2 n$ -i$\int_1^n$

h>= n$\log_2 n$ –n+1

ignoring the constant terms

h>= n$\log_2 n$

h=π n(log n)

Comparison tree for Binary Search:

Example**:** Suppose we have a list of items according to the following Position:

1. 1,2,3,4,5,6,7,8,9,10,11,12,13,14

   Mid=$(\frac{1+14}{2})$=$\frac{15}{2}$=7.5=7

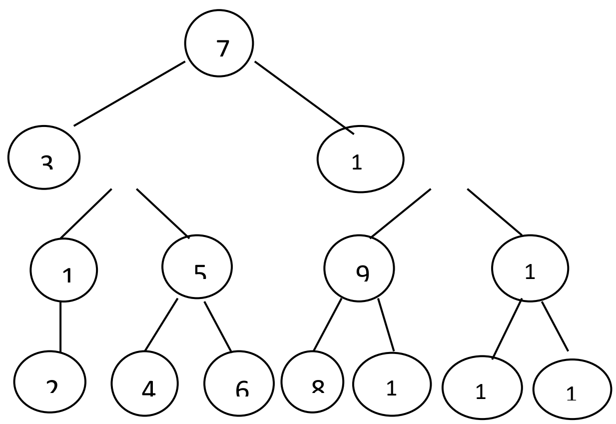
<table>
<tr><td colspan="2">1,2,3,4,5,6</td><td colspan="2">8,9,10,11,12,13,14</td></tr>
<tr><td colspan="2">Mid = $\left(\frac{1+6}{2}\right)$= $\frac{7}{2}$ = 3.5 =3</td><td colspan="2">Mid = $(\frac{8+14}{2})$ = $\frac{22}{2}$ = 11</td></tr>
<tr><td>1,2</td><td>4,5,6</td><td>8,9,10</td><td>12,13,14</td></tr>
<tr><td>Mid = $(\frac{1+2}{2})$ = $\frac{3}{2}$ = 1.5 =1</td><td>Mid = $(\frac{4+6}{2})$ = $\frac{10}{2}$ = 5</td><td>Mid = $(\frac{8+10}{2})$ = $\frac{18}{2}$ = 9</td><td>Mid = $(\frac{12+14}{2})$ =$\frac{26}{2}$ = 13</td></tr>
</table>

And the last midpoint is:

2, 4, 6, 8, 10, 12, 14

Thus, we will consider all the midpoints and we will make a tree of it by having stepwise midpoints.

According to Mid-Point, the tree will be:



Step1: Maximum number of nodes up to k level of the internal node is 2^k-1

For Example

2^k-1
$2^3-1= 8-1=7$
Where k = level=3

Step2: Maximum number of internal nodes in the comparisons tree is n!

(Here Internal Nodes are Leaves.)

Step3: From Condition1 & Condition 2 we get

$N! \leq 2^k-1$
$14 < 15$
Where N = Nodes

Step4: Now, $n+1 \leq 2^k$

Here, Internal Nodes will always be less than 2^k in the Binary Search.

Step5:

```
n+1<= 2^k
   Log (n+1) = k log 2

 k >=
  k >=log2(n+1)
```

Step6:

1. T (n) = k

Step7:

$T(n) \geq \log_2(n+1)$

Here, the minimum number of Comparisons to perform a task of the search of n terms using Binary Search