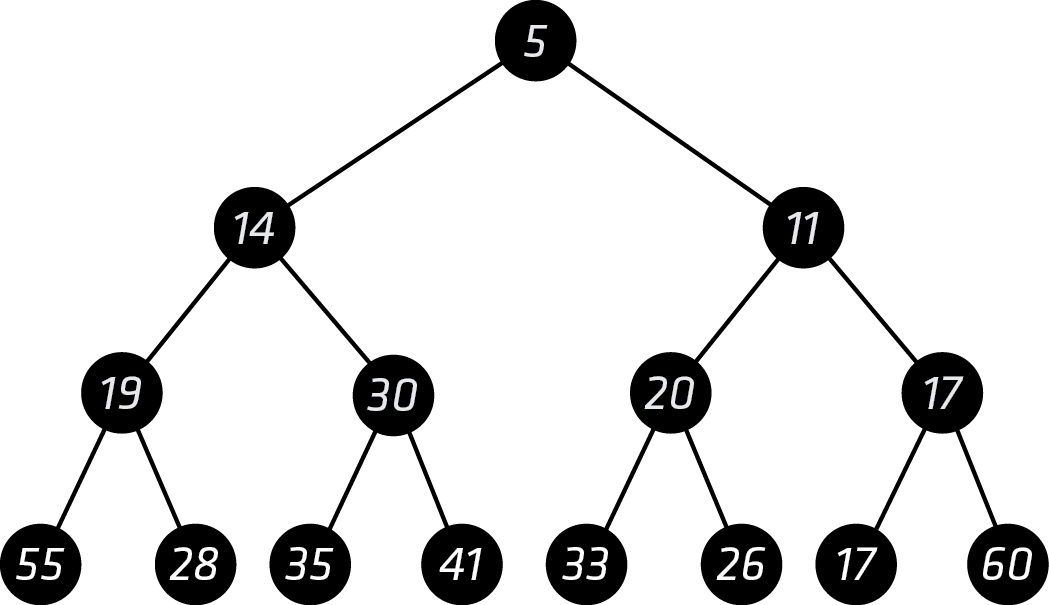
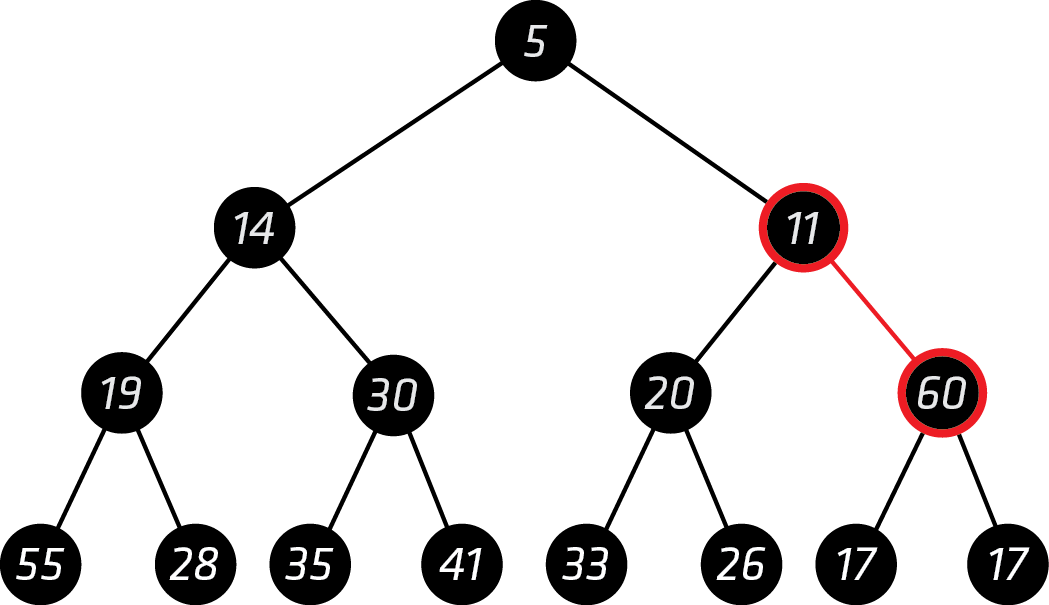
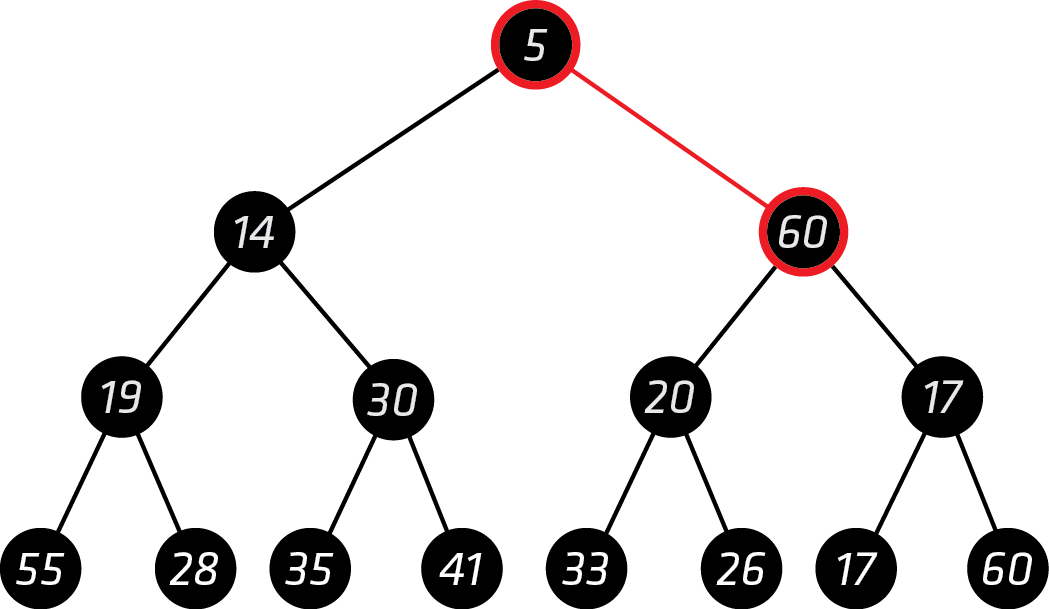
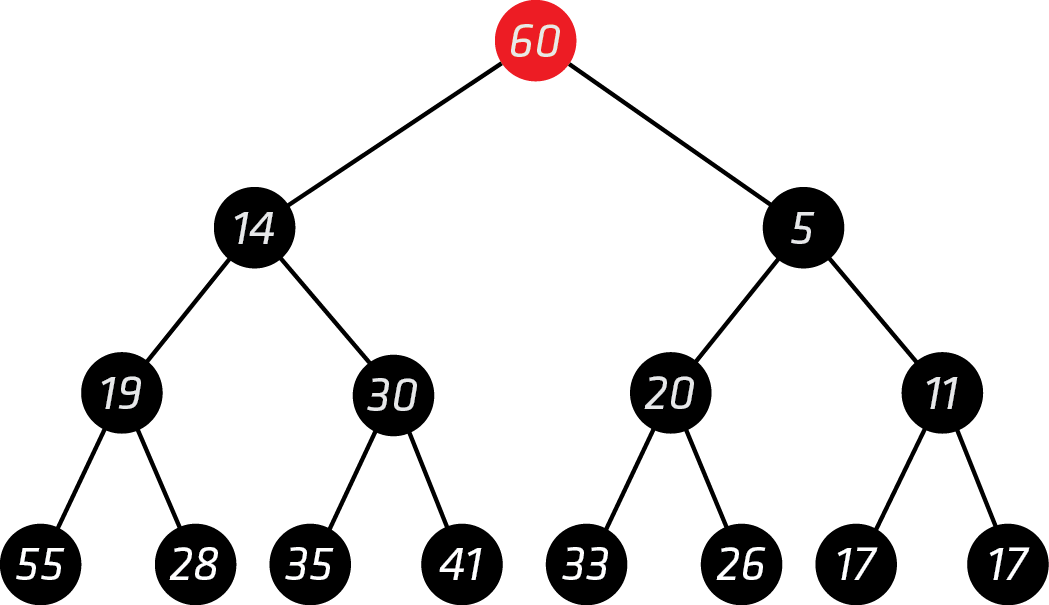
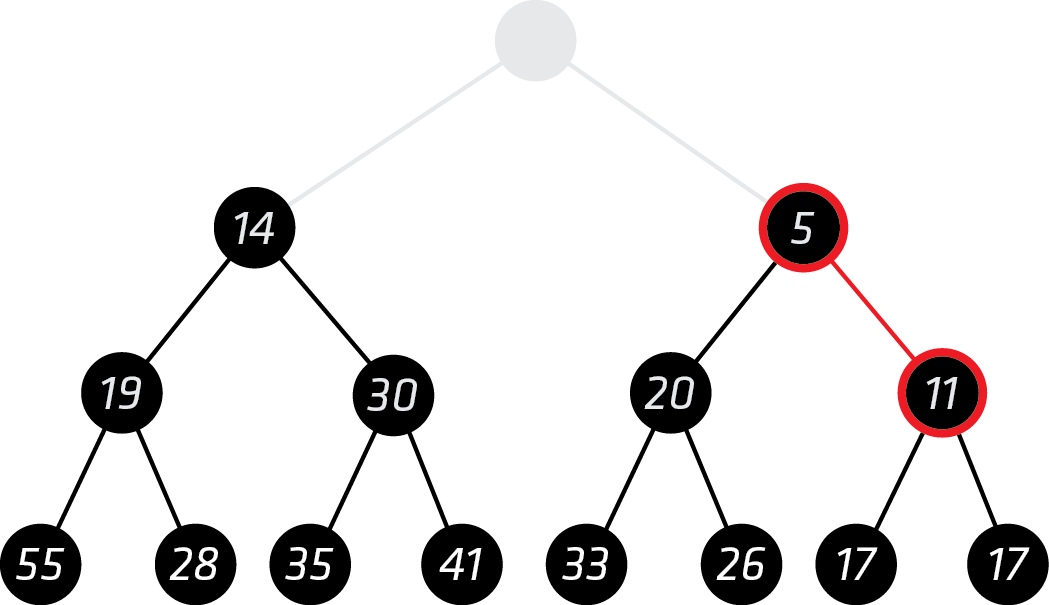
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

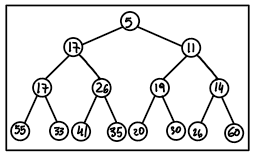
# q.1

## a



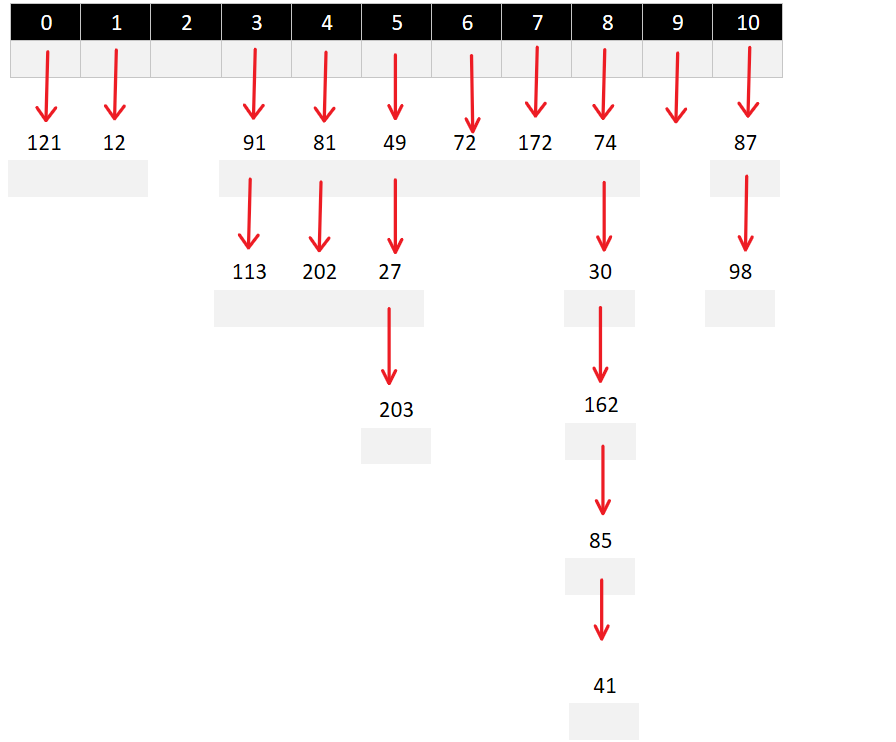
## b

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| C:\Users\Noah Francis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\B FINAL.PNG C:\Users\NOAHFR~1\AppData\Local\Temp\msohtmlclip1\02\clip_image001.pngc | |

Final tree:

# Q.2

## a

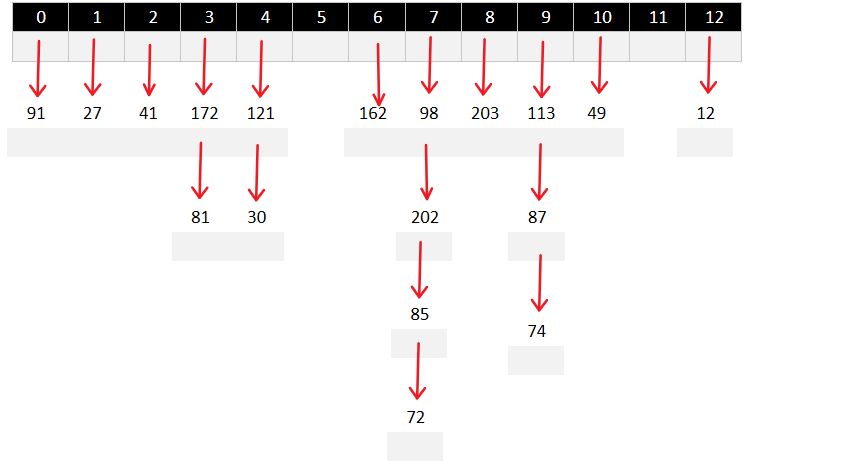


## b

the maximum number of collision is 10.

# Q3

The proposed solution with a larger array of 13 yields the following table:



This method reduces the load factor in this particular example with the given keys of the list since the maximum number of collision this time is 7 compared to 10 which was obtained previously with the array of 11 elements. However, it (the method) could potentially fail to reduce the number of collision if we would be given a different set of keys and therefore making this proposal senseless.

# Q.4

## a

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 24 |  |  | 95 | 45 |  |  | 25 | 27 | 16 | 31 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 89 |  |  |  |  |  |  | 38 | 14 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 12 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

## b

|  |  |
| --- | --- |
| 0 | 27 |
| 1 | 16 |
| 2 | 89 |
| 3 | 24 |
| 4 | 25 |
| 5 | 95 |
| 6 | 45 |
| 7 |  |
| 8 |  |
| 9 | 12 |
| 10 | 14 |
| 11 | 38 |
| 12 | 31 |

## c

|  |  |  |  |
| --- | --- | --- | --- |
| 0 |  |  |  |
| 1 | 27 |  |  |
| 2 | 89 |  |  |
| 3 | 25 |  |  |
| 4 | 24 | 38 | **COLLISION** |
| 5 | 95 |  |  |
| 6 | 45 |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 | 12 |  |  |
| 10 | 14 |  |  |
| 11 | 16 |  |  |
| 12 | 31 |  |  |

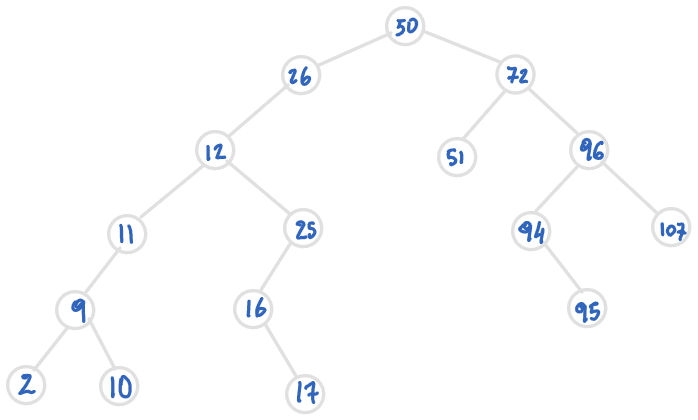
# Q5

## a

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | | 4 | |
|  |  |  |  | |  | |
|  |  |  | |  | |  |
| 13 |  | 20 | |  | | 17 |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  | 10 | |  | |  |
|  |  |  | |  | |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b | |  |  | | --- | --- | | 0 | 13 | | 1 |  | | 2 | 10 | | 3 | 20 | | 4 | 17 | | c | |  |  | | --- | --- | | 0 | 13 | | 1 | 20 | | 2 | 10 | | 3 |  | | 4 | 17 | |

# Q6

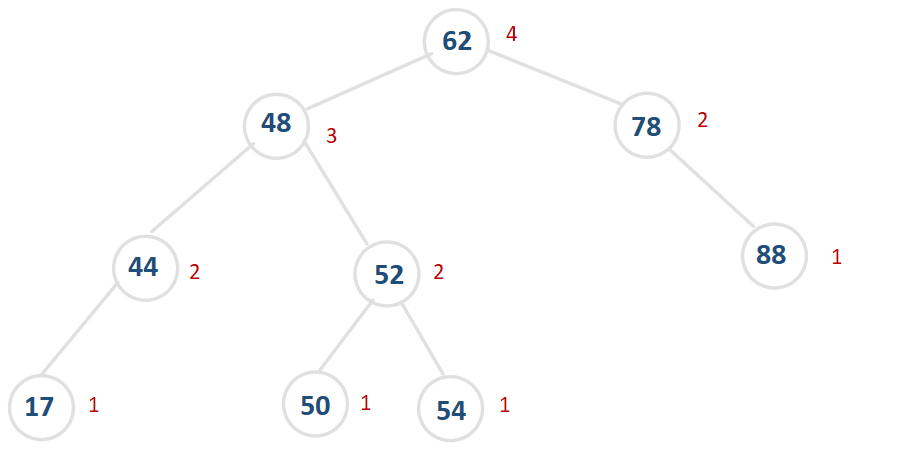


# Q7

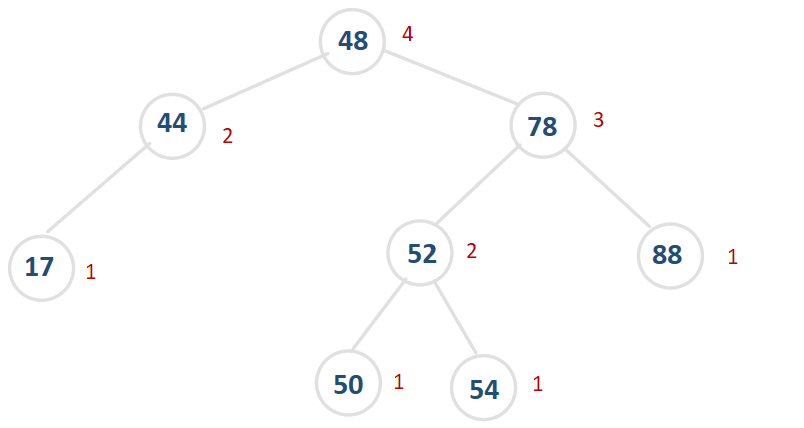
1. **Sorted linked list:** When searching for an element, in the worst case where the element is not found the time efficiency for searching is **O(n)** since the search was done on the size n of that list.
2. **Array based linked list:** The same operation will also result in **O(n)** and this also happens in the worst case where the element is not found
3. **Binary Search Tree:** As the name suggests a binary tree is used for searching because it is more
4. efficient then above abstract data structures, its efficiency is **O(log n)**.

# Q9

## a



## b



# Q9

Algorithm findHeight(h)

**Input**: h is the current height of the AVL tree, it is intialy equal to zero and only used reclusively.

**Output**: height of the AVL tree.

IF(T.parent = null)

return 0;

END IF

ELSE IF (T.balance < 1 and T.balance > -1)

return max(findHeight(T.left), findHeight(T.right)) + 1

// alternatively we can replace the return statement with the code bellow

left <-- findHeight(T.left)

right <-- findRight(T.right)

END ELSE IF

END

# Q10

count = 0

Algorithm equalElements(S[], i)

Input: S[] is an ordered array or sequence of n element, i is the current index of an array.

Output: returns true if there are two equal elements in the sequence or false otherwise.

IF (count = 0 AND S.length > 2 )

count ++

i = 1 { since this is the first time the method runs we want i to be after index 0}

END IF

IF(S.length =< i)

return false

ELSE IF (S.length = 2 OR (i+2) = (S.length -1))

IF(S[S.length -2] != S[S.length -1])

return false

ELSE

return true

END IF

ELSE

IF(S[i-1] = S[i] or S[i+1] = S[i])

return true

ELSE

equalElements(S, i+2)

END IF

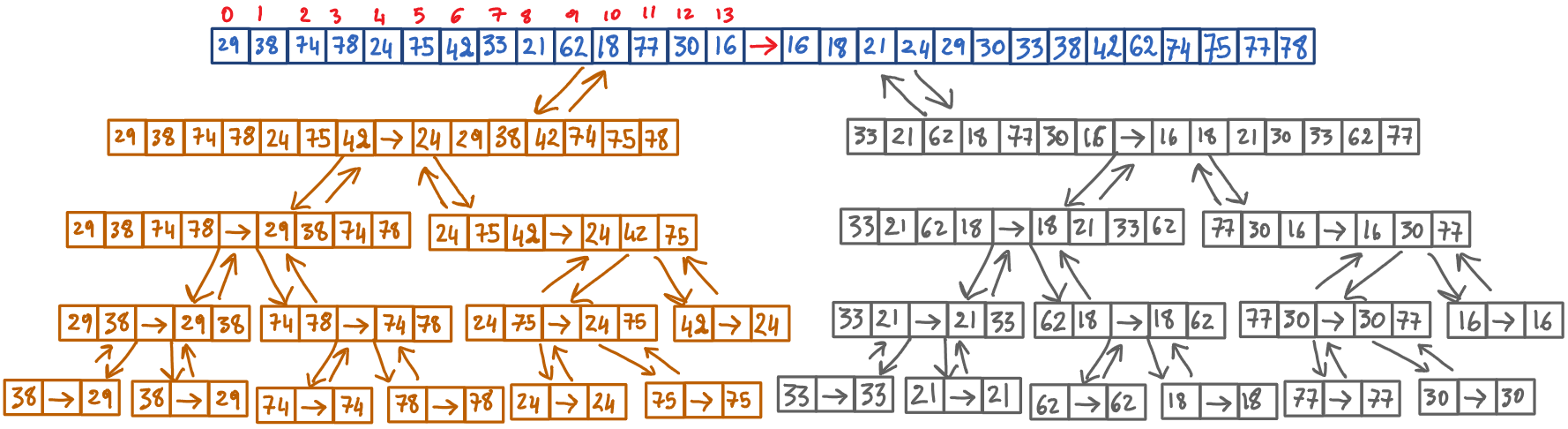
END IF

END

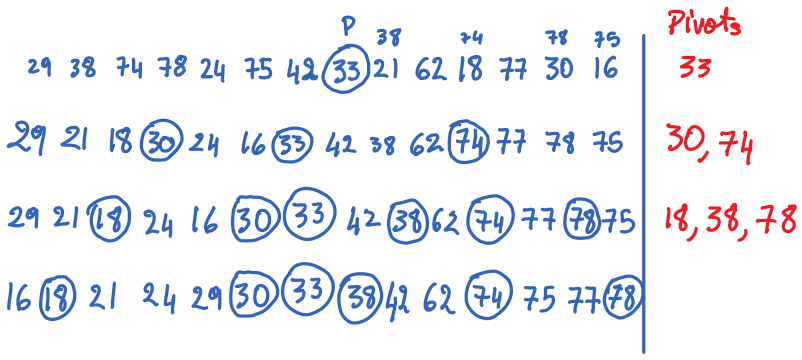
**The running time of this algorithm is O(n).**

# Q11

## a



## b



## C:\Users\NOAHFR~1\AppData\Local\Temp\msohtmlclip1\02\clip_image001.pngc

## d

