1) a)- The size n can decrease by any number between 1 and n

- The size of an instance will always decrease at least by a factor of 2, after two successive iterations of Euclid's algorithm.

- Two connective iteration of Euclid's algorithm we performed according the following formuls

$$gcd(m,n) = gcd(n,r) = gcd(r, n \bmod r) \text{ where } r = m \bmod n$$

b) • Need to show that $n \bmod r \leq n/2$
consider two cases $r \leq n/2$ and $n/2 < r < n$

if $r \leq n/2$ then
$n \bmod r < r < n$

if $n/2 < r < n$ then
$n \bmod r = n - r < n/2$

2) The decrease-by-one technique is used for generating $n!$ permutations of $\{1, \dots n\}$ Hence, smaller-by-one problem is to generate all $(n-1)!$ permutation. To do so;

• First solve the smaller problem.
• Then get the solution to larger problems
  - by inserting n in each of the n possible positions among elements of every permutation of $(n-1)$ elements
  - we can insert into the already existing list by moving right to left on left to right
• we will get the permutations $n.(n-1)! = n!$

For example: Let us obtain the permutations as

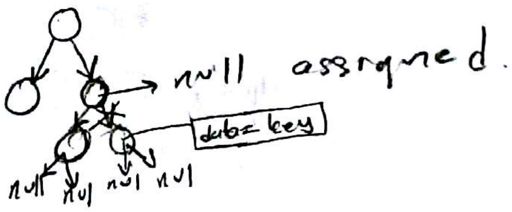
start A when: $n=1$ $n!=1$

insert B $\underbrace{AB \quad BA}_{\text{righ to left}}$ when: $n=2$ $n!=2$

insert C: $\underbrace{ABC \quad ACB \quad CAB \quad CBA}_{\text{Moving from right to left}}$ $\underbrace{CBA \quad BCA \quad BAC}_{\text{Moving from left to right}}$ when $n=3$ $3!=6$

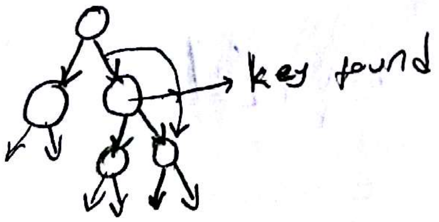
let us discuss more examples based on minimal change aproach

3) firstly, deleting key is searched on Binary three. If key is found leaf, righ or left pointer of parent of leaf assigned null.



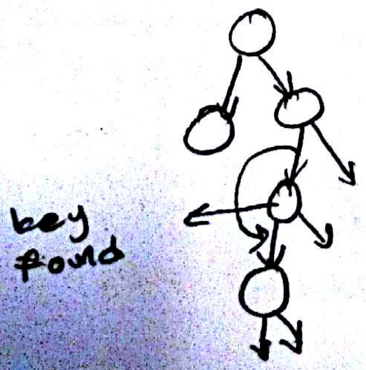
best case $\Theta(\log n)$
worst case $\Theta(\log n)$
average case $\Theta(\log n)$

Secondly, if parent has two child,



best case $\Theta(1)$
worst case $\Theta(\log n)$
average case $\Theta(\log n)$

Thirdly, if parent one child.



best case $\Theta(1)$
worst case $\Theta(\log n)$
average case $\Theta(\log n)$

Scanned by CamScanner

4) it is possible

Prove that

// Algorithm

```
begin sortedNumber (arr[1...n], size)

    indexOfNegOne = 0
    indexOfPosOne = size
    for i=0  i<size  ++i
        if arr[i] == -1
            swap(i, indexOfNegOne, arr)
            ++ indexOfNegOne
        end if
        if arr[i] == 1
            swap(i, indexOfPosOne, arr)
            -- indexOfPosOne

        endif

    end for

end
```

- This for loop goes n and all element is sorted O(n). It is sorted. Thus, This array is sorted

possible.

5) As the idea here is to examine the element at A[[n/2]] recursively

```
elementEqInd ( A[1..n] , offset)
    if A[[n/2]] equals offset + [n/2], then
        return true
    if |A| ≤ 1
        return false
    if A[[n/2]] < offset + [n/2]
        return elementEqInd (A[([n/2]+1)...n], offset]
    else return elementEqInd(A[1...([n/2]-1)] ,offset)
```

The running time is O(logn)