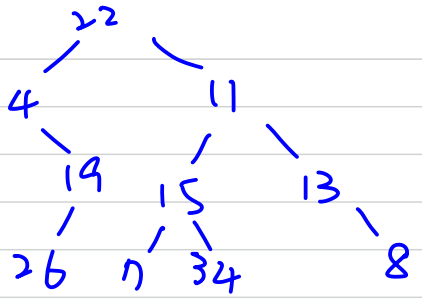
Q0.

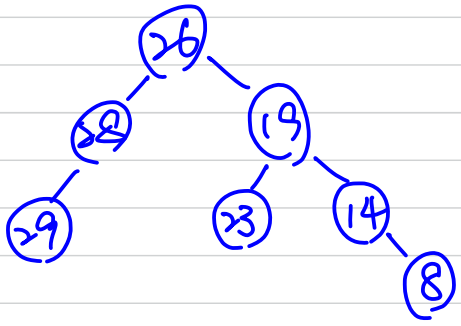
1. B9S0l0l0 jimmy
2. B9S0l0l0 jimmy
3. B9S0l0l0 jimmy
4. B9S0l0l0 jimmy , B07S0111 Wei
5. B9S0l0l0 jimmy , B07S0111 Wei , Discord

1-1.



找 root：
用 postorder 的尾找 root. —①
把 inorder 分成 left, root, right. —②
右半邊的 subtree：
找 subtree 的 root (用①, ②)
左半邊的 subtree：
找 subtree 的 root. (用①, ②)
直到沒有 node.

1-2.



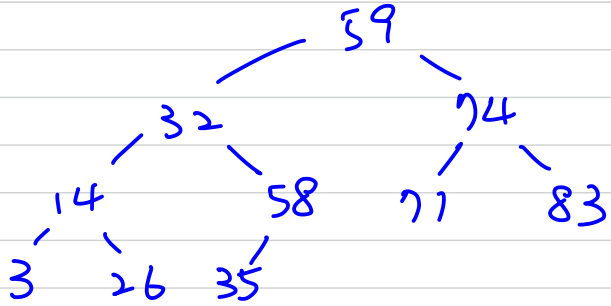
1-3. Convert (T)

```
if T = NULL
    return
Convert (T.right)
T t = T.right
Convert (T.left)
end
```

Since the algorithm only traverse every node once, the time complexity is O(n)

1-4. if x is right node of y, the node smaller than x are y, the node in left sub tree of y, and the left child of x, since x is leaf node, there is no child in x, and the node in left subtree of y are all smaller than y, Hence y is the largest node among all nodes smaller than x. vice versa. The statement is proved.

1-5.



1-6. Calculate the number of node and calculate the node index without building array.

```
total = 0 , num = 0
Waste time (T. total . num)
  if (T = NULL)
     return
  Waste time (T.left. total x2 +1 , num +1)
  Waste time ( T. right. total x2+1 . num+1)
```

Collect the maximum of total and the num, the waste space is total - num *

2-1. Get $a_m, a_M(i, j, k)$

$i=1,\ j=2,\ k=3$

for $l = 4$ to $n-2$

if (query(a_i, a_j, a_k) = False)

swap (a_j, a_k)

if (query (a_i, a_j, a_k) = False)

swap (a_i, a_j)

if (query (a_i, a_j, a_k) = True)

$a_j = a_l$

1次最多 query 3次, worse case = $3 + 3(n-3) = 3n-6$ 次 query.

query complexity = $O(n)$

2-2.

first, we choose a_1, a_2, a_3 and ask query until if is sorted. (worse case 3 times). then using algorithm in P2-3 to insert the remain a_i from a_4 to a_n. Since the insert algorithm has query complexity $O(\log n)$, and complexity of the loop is $O(n-4) = O(n)$.
Hence the total complexity is $O(n \log n)$.

2-3. $l = 1,\ r = n$

Search ({a}, l, r)

while $l < r$

mid = $(l+r)/2$

if (query (a_{mid}, a_{n+1}, a_n) = True)

l = mid +1

else

r = mid - 1

return l

The idea is similar to binary search, thus the query complexity is $O(\log n)$

2-4

2-5

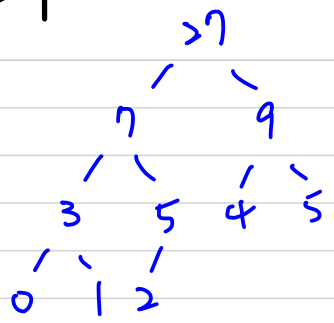
7 good triplets a_2, a_4, a_1 a_3, a_4, a_1 a_3, a_4, a_5

a_4, a_1, a_5 a_2, a_1, a_5

a_2, a_4, a_5 a_3, a_1, a_5

2-6

3-1

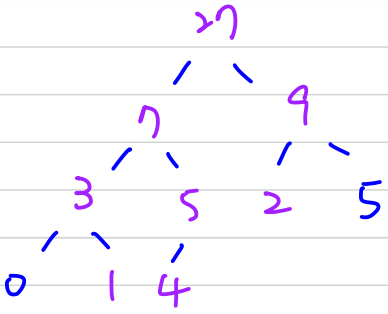


27	7	9	3	5	4	5	0	1	2

3-2

Build Max Heap

```
A.heapsize = A.length
for i = ⌊A.len/2⌋ to 1
    Max-Heapify(A, i)
```



27	7	9	3	5	2	5	0	1	4

$[2, 4, 9, 1, 5, 27, 5, 0, 3, 7]$.

3-3

```
Build-Cartesian(T, new)
  if (new > T.index)
  |  T.parent = new        let new node
  |  new.left = T          as new root.
  |  T = new
  else
  |  if (T.right = NULL)
  |  |  new.parent = T
  |  |  T.right = new
  |  else
  |  |  Build-Cartesian(T.right, new)
  |  end if
  end if
```

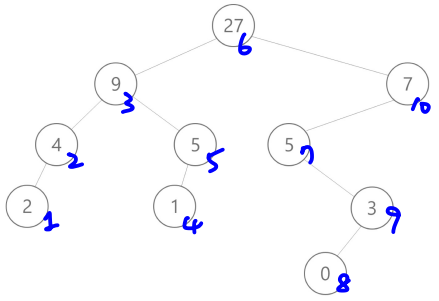
```
root = A[0]
for i = 1 to A.length
    Build-Cartesian(root, A[i])
```

說明: we have to traverse all the information in Array, thus the time is n, then in recursive, the worse case is find the leaf of tree, thus is $O(\log n)$,
Hence the time complexity is $O(n \log n)$

3-4 decorate the tree with their order index

```
get hight(index, T)
  while (T.index == index)
  |  if (T.index > index)
  |  |  T = T.left
  |  else
  |  |  T = T.right
  |  end if
  end while
  return T.hight
```

ex:

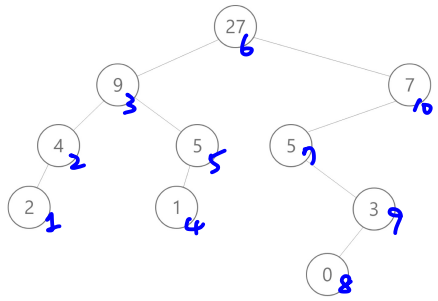


Since we only traverse a branch of tree (the worse case of branch height is $\log n$). Thus the time complexity is $O(\log n)$

3-5. decorate the tree with their order index

```
get max (l, r, T)
  while (l ≤ T.index ≤ r)
    | if (T.index > r)
    |  | T = T.left
    | if (T.index < l)
    |  | T = T.right
    |  end if
  end while
  return T.height
```

ex:



Since we only traverse a branch of tree (the worse case of branch height is $\log n$). Thus the time complexity is $O(\log n)$

3-6.

```
left-hend-side (T)
  if (T = NULL)
   | return
  end if
   left-hend-side (T.left)
   print (T.height)
```

Since we only can see the building that taller then its left hand side.

The feature of the Cartesian is. the right child of the node is neigher taller than parent nor to the left of parent.

Thus we only need to traverse the left child. And the branch of a tree height with worse case is $\log n$. Hence the time complexity is $O(\log n)$