HW 4 ⇒ ARSHITHA BASAVARAJ

2) Yes, the above loop will work if the last loop is modified to: j = 1 to A.length.

Explanation: (with an example)

i/p array ⇒ A = {1, 2, 3, 3, 5, 6, 2, 1, 1}

Using the algorithm, output array B will be as follows. [Original Algorithm]

B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
1	1	1	2	2	3	3	5	6

Modified algorithm output will be as follows

B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
1	1	1	2	2	3	3	5	6

However, the original algorithm is a stable counting sort, i.e., identical elements are placed in the same order as they occured in the input array, in the output array.

For instance, 1 appears thrice in the i/p array.

B[1] ⇒ the 1 that appears first in the i/p array
B[2] ⇒ the 1 " " 2nd in " " "
B[3] ⇒ the 1 that " 3rd " " " "

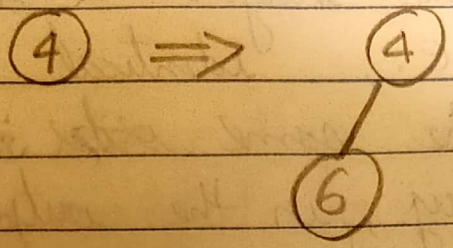
But modifying the original algorithm, we get

B[3] ⇒ the 1 that appears 3rd in i/p array
B[2] ⇒ the 1 " " 2nd in i/p array
B[1] ⇒ " 1 " " 1st in i/p array

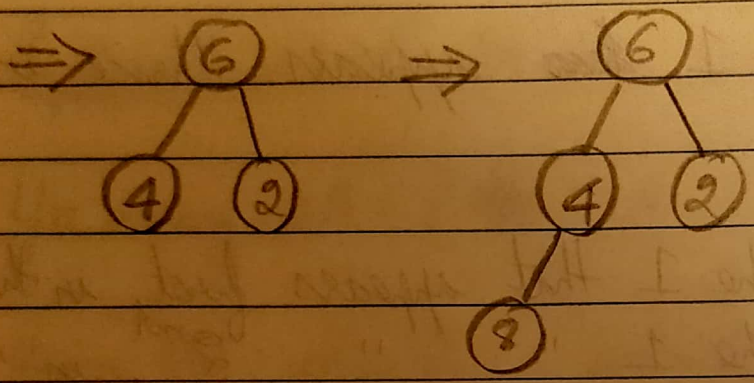
III^dly for other repetitive elements of the array

Hence, original is better than the modified algo. There's no advantage to using the modified algorithm.

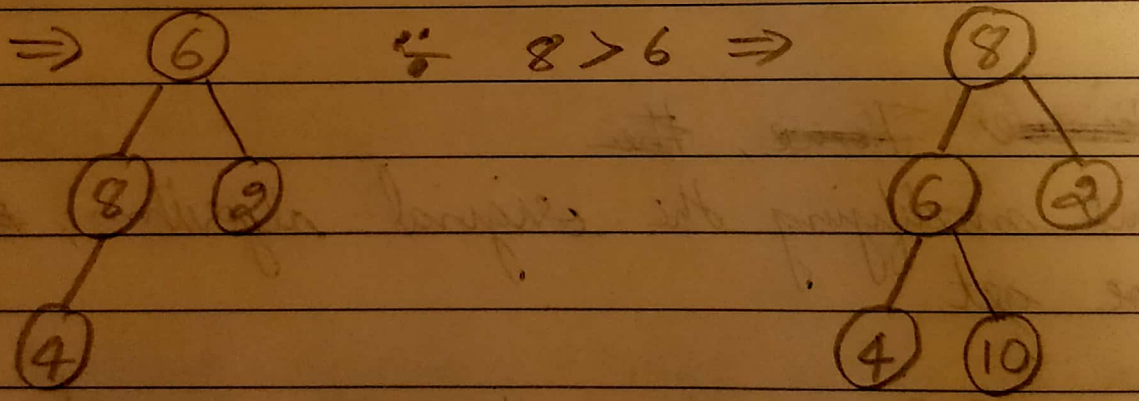
5)

a) 

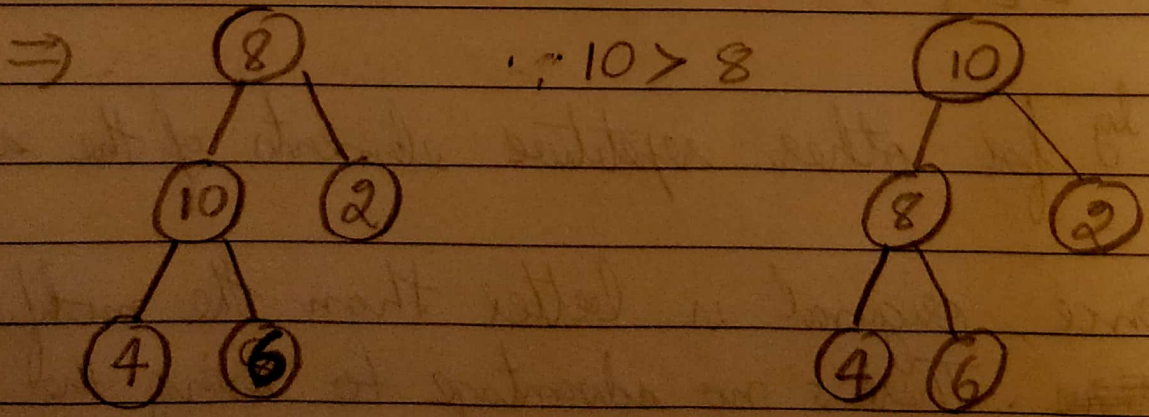
∵ 6 > 4



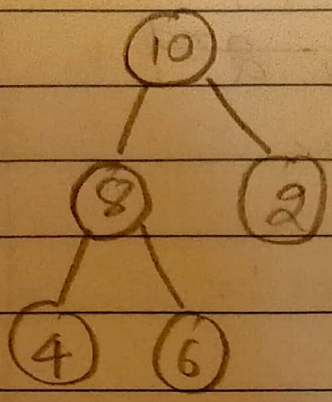
∵ 8 > 4



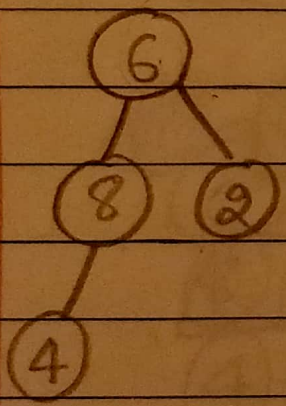
∵ 10 > 6



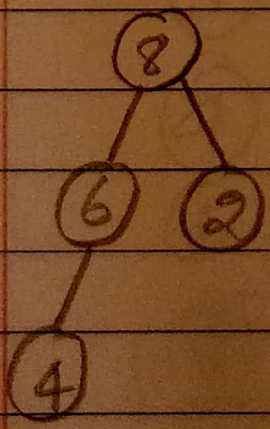
b> Heapify after extracting max & replacing with least element



⇒ Initial max heap.



⇒ 6 < max (8,2) ∴ swap 6 & 8



⇒ There are no other violating nodes

Content of the heap after extract-max operation

3) In the worst case, Quick sort has $O(n^2)$ as it's runtime.

But if we choose a pivot that divides the unsorted assay equally, then the time complexity can be reduced to $O(n \log n)$.

This right pivot selection can be achieved by finding the median of medians of the unsorted assay.

Median of medians can be found by using the Selection algorithm.

[Algorithm description]

Select (k, n, A) // selects the k^{th} smallest
// of n elements in A
// A is the input array with
// n elements.

In our case, we need $\lceil n/2 \rceil^{th}$ element of the unsorted array.

Procedure

① Divide i/p into $n/5$ groups of 5 elements each.

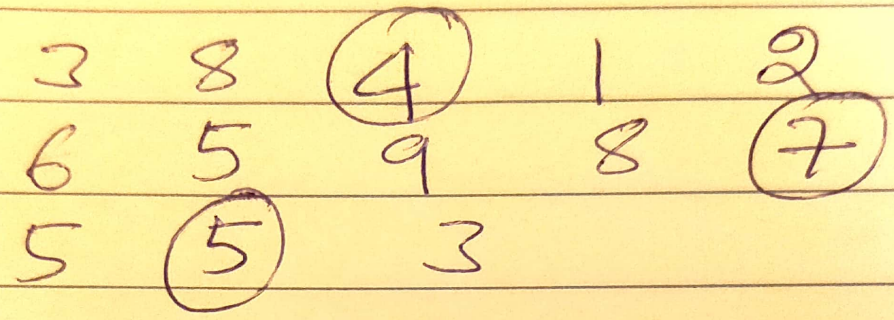
② Find the median of each group

③ Recursively find the median of medians

Select $\left(n = n/5,\ k = n/2 \cdot 1/2\right)$

④ Use this value as the pivot for quick sort.

E.g.: 3, 8, 4, 1, 2, 6,
5, 9, 8, 7, 5, 5, 3



median of medians } = 5 = pivot

If we use this pivot, after 1st partition of unsorted array we get the following.

3 4 1 2 5 5 3 | 5 | 8 6 9 8 7

∴ Quick sort with median of medians as the pivot gives $O(n \log n)$ in worst case.

(S1)

1. Let stack 1 be stored with elements of i/p array.

(i) Declare an empty stack 2 (S2)

(ii) Run a loop for 'n' times (n is size of array)

a. Keep pushing elements in S2 till S2.top() < S1.top()

b. If S1.top() < S2.top()

a. Keep pushing elements in s2 till s2.top() < s1.top()

b. If s1.top() < s2.top()
then swap[s1.top(), s2.top()]
then push s2.top() back on top.

After each iteration, one of the stack is empty so the process (ii) is repeated on the other stack.

(iii) At the end of each iteration, the sorted element is placed in its correct position in the original array.

(iv) The array is then printed out.

[Pseudocode]

```
Stack s1; // contains elements of the array
Stack s2; // empty initially.

int i;
int temp;
int t;          } // 3 variables

for (i = 0 to n)
{
    // After each iteration one stack is empty
    if (i % 2 == 0) // & the other is full
    {
        while (s1 != empty)
        {
```

```
while (s1 != empty)
{
    t = s1.pop();

    if (s2 == empty)
        s2.push(t);
    else
    {
        if (s2.top() > t){
        //swap
        temp = s2.pop();
        s2.push(t);
        s2.push(temp);
    }
```

```
        else
        {
            s2.push(t);
        }
    }
}

// store back the sorted element in the
   original array.
arr[n-1-i] = s2.pop();

}
```

else

The same operation as above is done. Only this time S1 is empty & S2 is full.

~~Reference~~ Reference : Internet

4a. Result:

0 1 2 3 4 5 6 7 8 9

b. Yes, it sorts all arrays of n integers.

The given algorithm is just another implementation of Insertion sort. And therefore, it works.