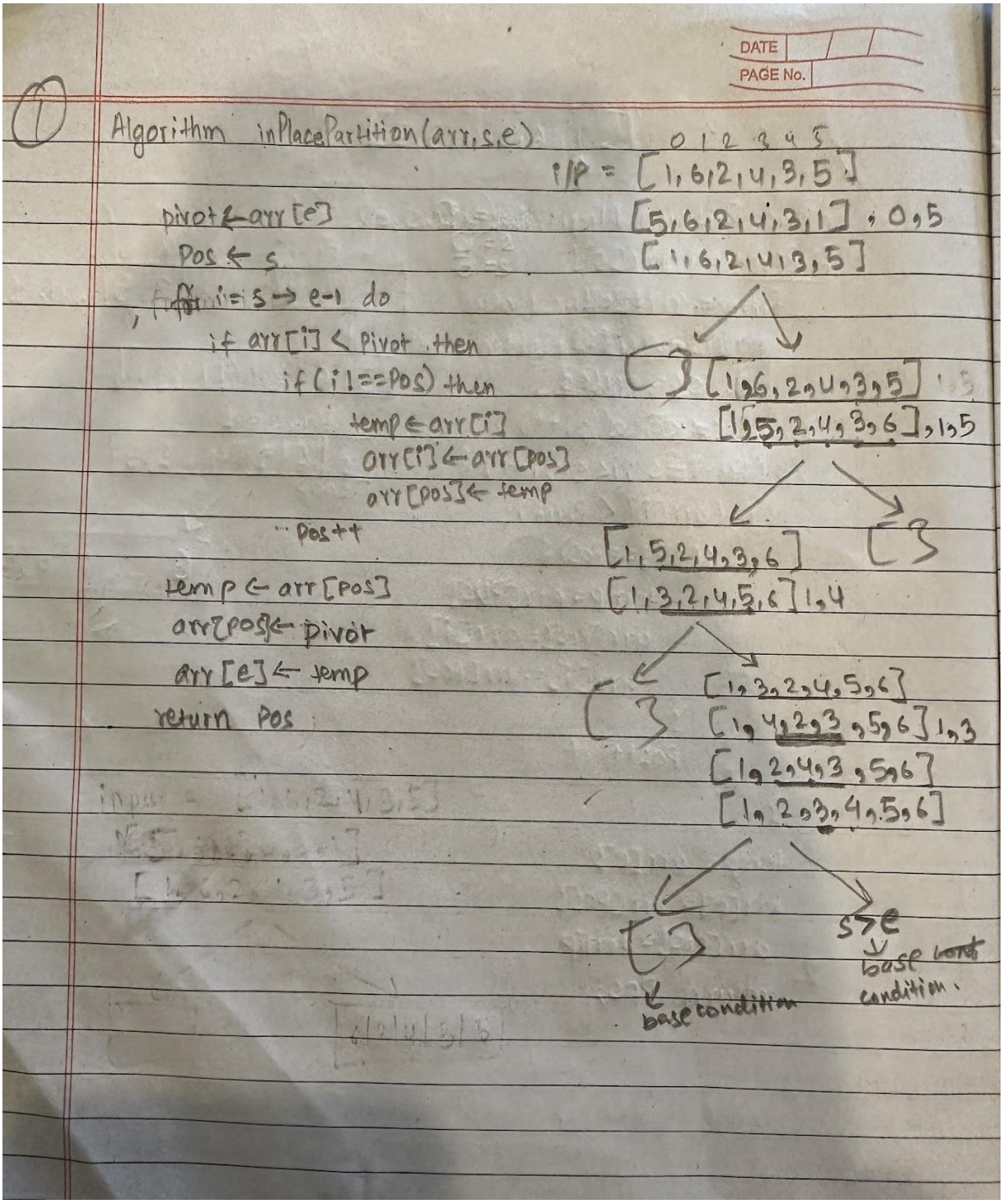
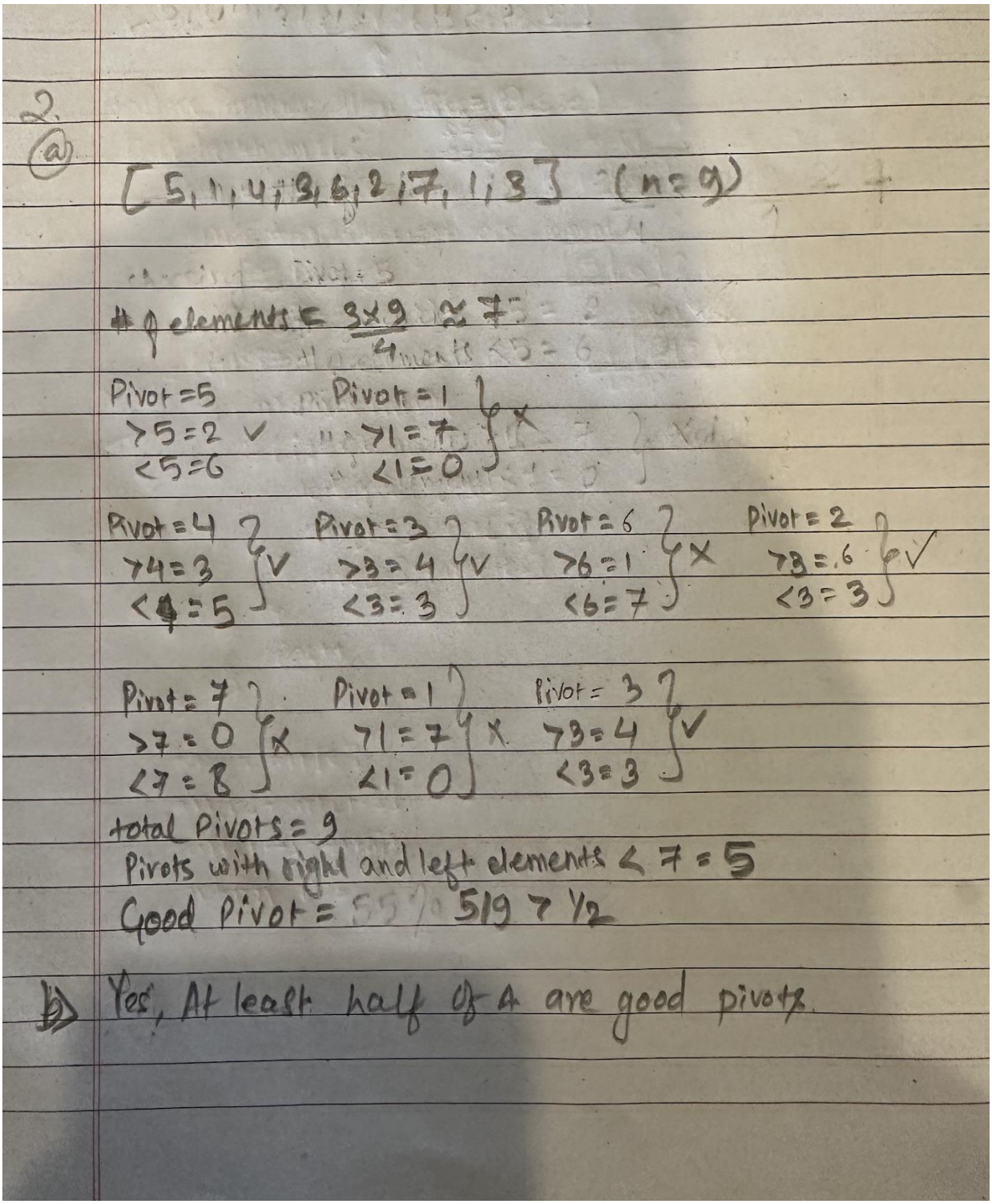
1.


DATE
PAGE No.
Algorithm inPlacePartition(arr, s, e)
pivot ← arr[e]
Pos ← s
for i = s → e-1 do
if arr[i] < pivot then
if (i != Pos) then
temp ← arr[i]
arr[i] ← arr[pos]
arr[pos] ← temp
Pos++
temp ← arr[pos]
arr[pos] ← pivot
arr[e] ← temp
return pos
0 1 2 3 4 5
I/P = [1,6,2,4,3,5]
[5,6,2,4,3,1] ; 0,5
[1,6,2,4,3,5]
[1,6,2,4,3,5]
[1,5,2,4,3,6], 1,5
[1,5,2,4,3,6]
[1,3,2,4,5,6] 1,4
[1,3,2,4,5,6]
[1,4,2,3,5,6] 1,3
[1,2,4,3,5,6]
[1,2,3,4,5,6]
s>e
base condition
base condition

2.


2
(a)
[5, 1, 4, 3, 6, 2, 7, 1, 3] (n=9)
# of elements ≤ 3×9/4 ≈ 7
Pivot = 5
>5 = 2 ✓
<5 = 6
Pivot = 1
>1 = 7
<1 = 0
} X
Pivot = 4
>4 = 3
<4 = 5
} ✓
Pivot = 3
>3 = 4
<3 = 3
} ✓
Pivot = 6
>6 = 1
<6 = 7
} X
Pivot = 2
>2 = 6
<2 = 3
} ✓
Pivot = 7
>7 = 0
<7 = 8
} X
Pivot = 1
>1 = 7
<1 = 0
} X
Pivot = 3
>3 = 4
<3 = 3
} ✓
total Pivots = 9
Pivots with right and left elements < 7 = 5
Good Pivot = 5/9 > 1/2
(b) Yes, At least half of A are good pivots.

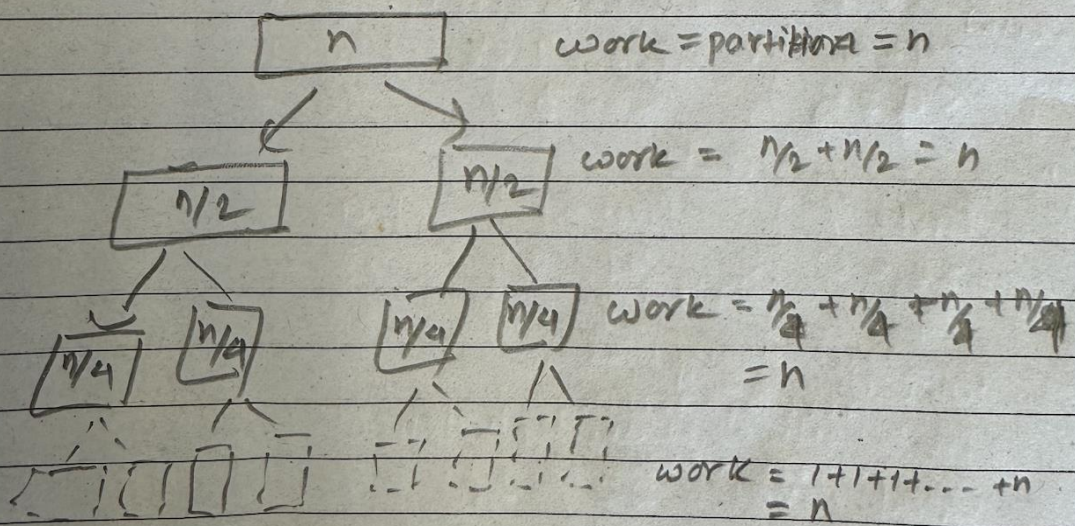
3.

(3)

Good case is when pivot divides the array into reasonably balanced ~~Arrays~~ halves.

Best case is when pivot divides the array into two equal halves.
Good case is not the base case in Quicksort but they do have similar Time Complexity.

lets take an Array with n elements and Suppose ~~the~~ the pivot divides the Array into equal halves in every levels.



Time Complexity = # of levels * work
= O(log₂n) * O(n)
= O(n log₂n)

4.

```
class Solution {
   public static int quickSortedIndex(int[] arr, int s, int e){
      int pivot = arr[e];
      int pos = s;
      for(int i=s;i<e;i++){
         if(arr[i] < pivot){
            if(pos != i){
               int temp = arr[pos];
               arr[pos] = arr[i];
               arr[i] = temp;
            }
            pos++;
         }
      }
      arr[e] = arr[pos];
      arr[pos] = pivot;
      return pos;
   }

   public int findKthLargest(int[] nums, int k) {
      int position = nums.length - k;
      int start = 0;
      int end = nums.length - 1;
      while(start < end){ // element in position or index is the of the correct order
         int cursor = quickSortedIndex(nums,start,end);
         if(cursor==position){
            break;
         }
         if(position < cursor){
            end = cursor - 1;
         }else{
            start = cursor + 1;
         }
      }
      return nums[position];
   }
}
```