2. QuickSort

Consider the following variation of the Quicksort algorithm:

Quicksort(A, p, r)

```
If p>r then error "empty array"
Else if r–p==1 then
        If A[p]>A[r] then swap A[p] and A[r]
Else if r–p≥2 then
        Swap A[p+2] and A[r]
        q=Partition(A,p,r)
        Quicksort(A,p,q–1)
        Quicksort(A,q+1,r)
```

What are the base case input sizes? _____0,1___

What is the pivot picking strategy? The strategy is to use the third element in the array to partition the array into two parts. The partition will use the third element in the array to swap the values or not.

Suppose it is given an array A of size 5 as input, and that this array contains the integers 6,7,8,9 and 10 in some order. Show the array with a distribution of these numbers that will make this algorithm exhibit the worst-case performance in every recursive execution. The first and second cell contain 8 & 10. Fill in the rest of the numbers:

8	10			

Show your work below by explaining why the algorithm will perform at its worst case for your input. This explanation should have the following components: (1) The recursion tree of the above algorithm operating on your input array, with each node of the recursion tree showing the input to that execution, the number picked as the pivot, and the array after partition completes, and (2) an explanation in plain English as to why the algorithm will perform at its worst complexity in each recursive execution.

Partition(A, p, r)

```
1  x = A[r]
2  i  = p – 1
3  for j = p to r -1
4       if A[j] ≤ x
5            then i = i + 1
6            swap A[i] and A[j]
7  swap A[i +1] and A[r]
8  return  i +1
```

Complexity:
Partition on A[p…r] is Θ(n)
where n = r –p +1