Data Structure and Algorithm Training Program

Week 1 : Pseudo Code Solution Part 1

Problem 2 : Write a Program to reverse the order of the elements of a given array.

Iterative Pseudo Code : Two pointer Approach of Iteration

- Time Complexity : O(n)
- Space Complexity : O(1) [In-Place]

```
reverse(A[] , l , r)
 {
    i=l
    j=r
    While(i<j)
     {
       temp=A[i]
       A[i]=A[j]
       A[j]=temp
        i=i+1
        j=j-1
     }
 }
```

Recursive Pseudo Code : Decrease & Conquer approach

- Time Complexity : O(n)
- Space Complexity : O(n) Recursion Call Stack
- Decreasing the input size by 2 at each step of recursion
- Recurrence Relation : T(n) = T(n-2) + c

```
recursiveReverse(A[] , l , r)
 {
    if(l>=r)
      Return

    temp=A[l]
    A[l]=A[r]
    A[r]=temp
    recursiveReverse(A[] , l+1 , r-1)
 }
```

Problem 6 : Find a minimum Value in Sorted and rotated array

Pseudo Code : Divide and conquer approach of binary search.

- Time Complexity : O(logn)
- Space Complexity : O(logn) [Recursive Call Stack]

```
rotatedMinimum(A[] , l , r)
  {
     if(l>r)
          return A[0]
     if(l==r)
          return A[l]

     mid = (l+r)/2
     if(mid<r && A[mid]>A[mid+1])
          return A[mid+1]

     if(mid>l && A[mid-1]>A[mid])
          return A[mid]

     if(A[mid]<A[r])
          return  rotatedMinimum(A, l, mid-1)
     else
          return  rotatedMinimum(A, mid+1, r)
  }
```

Base Case

- When array is not rotated at all
- If there is only one element

Divide

- Calculate the mid
- Check If the mid element is minimum
- Check If (mid+1) element is minimum

Conquer

- Compare with A[r]
- Search in the left half or right half

Problem 7 : Find maximum and minimum Value in an array

Pseudo Code : Brute Force Approach

- Time Complexity : O(n)
- 2n-2 comparisons in worst case
- Space Complexity : O(1)

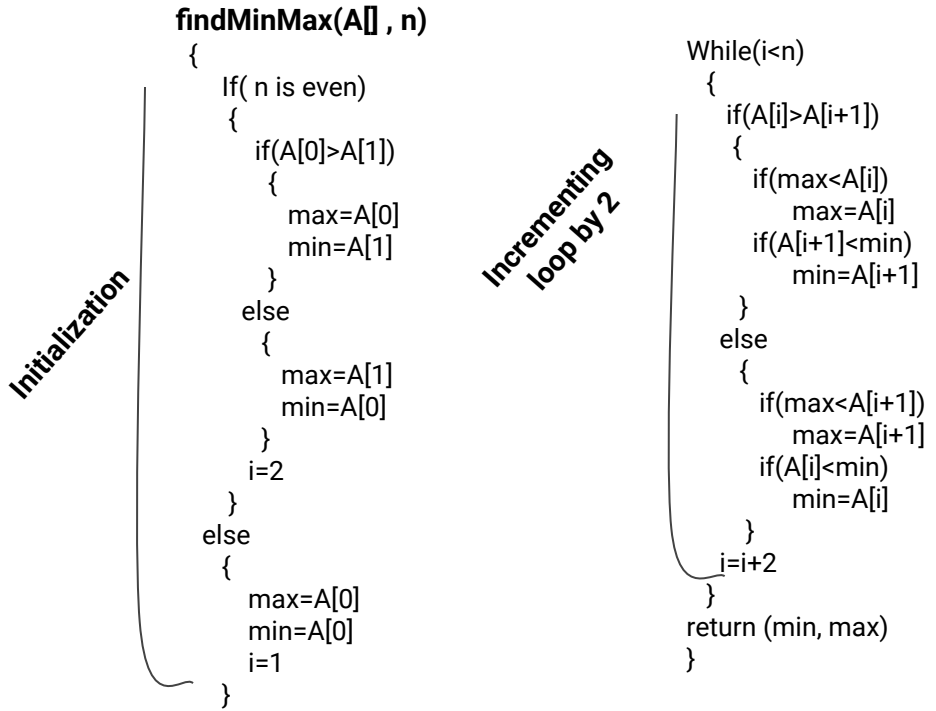
```
findMinMax(A[] , n)
{
  max=A[0]
  min= A[0]
  for(i=1 to n-1)
   {
     if(A[i]>max)
        max=A[i]

     else if(A[i]<min)
       min= A[i]
   }

return (max, min)
}
```

Pseudo Code : Efficient Approach

- Time Complexity : O(n)
- Worst case : 3n/2-1 Comparisons in worst case
- Space Complexity : O(1)
- Comparing 2 value in each stage of iteration [Incrementing the the loop size by 2 at each step.]



Continued….

Pseudo Code : Recursive Approach [Divide & Conquer]

- Recurrence Relation : T(n) = 2T(n/2)+ 2
- Time Complexity T(n) = 3n/2 - 2 = O(n)
- 2 Base case : (l==r) and (r==l+1)

- Space Complexity : O(logn) Recursion Call Stack [Because height of Recursion tree is O(logn)]

```
findMinMax(A[] , l , r )
{
    if(l==r)
     {
          max=A[l]
          min=A[l]
          return (min, max)
     }

    If(r==l+1)
     {
          if(A[r]>A[l])
           {
               max=A[r]
               min=A[l]
           }
          else
           {
               max=A[l]
               min=A[r]
           }
        return (min, max)
     }
```

Base Case

```
mid = (l+r)/2
```

Divide

```
(max1, min1) = findMinMax(A[] , l, mid)
(max2, min2) = findMinMax(A[] , mid , r)
```

Conquer

```
if(max1>max2)
     max= max1
 else
     max= max2

if(min1<min2)
     min= min1
else
     min= min2

return (min, max)
}
```

Combine

Enjoy Algorithms!

Thank You.