

1) Alg 1 =  $T(n) = 5T(\frac{n}{2}) + O(n)$

$b=5 \quad s=2 \quad d=1 \quad \begin{matrix} 5 > 2^1 \\ 5 > 2 \end{matrix} \quad \boxed{\begin{matrix} \therefore O(n^{\log_2 5}) \\ O(n^{2.322}) \end{matrix}} \quad \checkmark$

Alg 2 =  $T(n) = 2T(n-1) + O(1)$

$T(1) = 2 \cdot T(0)$

$T(2) = 2 \cdot T(1) + O(1) = 2T(1)$

$T(3) = 2 \cdot T(2) + O(1) = 2 \cdot 2 \cdot T(1)$

$T(4) = 2 \cdot T(3) + O(1) = 2 \cdot 2 \cdot 2 \cdot T(1)$

$\boxed{\therefore T(n) = O(2^n)} \quad \checkmark$

Alg 3 =  $T(n) = 7T(\frac{n}{3}) + O(n^2)$

$b=7 \quad s=3 \quad d=2 \quad \begin{matrix} 7 < 3^2 \\ 7 < 9 \end{matrix} \quad \boxed{\therefore O(n^2)} \quad \checkmark$

$O(n^2) < O(n^{2.322}) < O(2^n) \quad \checkmark$

After comparing the running times, I would choose Alg 3:  $O(n^2)$  ✓

2) RmDup() Input: inArr (array of  $n$  elements),  $n$   
Output: outArr (sorted array w/o duplicates)

```
{  
    inArr = MergeSort(inArr, n); // sorts inArr in  $O(n \lg n)$  time ✓  
  
    currVal = inArr[0]; // current value/duplicate to look for  
    outArr[0] = inArr[0];  
    int out = 1; // index for outArr  
    for (int i = 1; i < n; i++) //  $O(n)$   
    {  
        if (currVal != inArr[i]) // checks for next non-duplicate in the array  
        {  
            currVal = inArr[i];  
            outArr[out] = inArr[i];  
            out++;  
        }  
    }  
    return outArr;  
}
```

$$O(n \lg n) + O(n) = \boxed{O(n \lg n)} \quad \checkmark$$

Without preprocessing work, each element would have to be checked against each other element, doing this could produce a running time of  $O(n^2)$  ✓



3)

a)

Input: inArr, left (0), right (size of array)  
Output: Either index i for inArr[i] = i or false

FindIndex (inArr, left, right)

{

if (right == 1)

return right;

mid =  $\frac{\text{left} + \text{right}}{2}$ ;

if (inArr[mid] == mid)

return mid;

if (inArr[mid] < mid)

return FindIndex (inArr, mid + 1, right);

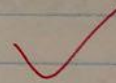
if (inArr[mid] > mid)

return FindIndex (inArr, left, mid - 1);

else

return false;

}



3b) input: sorted array  $B$ ,  $p, q$   
output: number of elements between  $p$  and  $q$

Num Element ( $B, p, q$ )

```
{  
    if ( $p == q$ )  
        return 0;  
    else {  
         $ip = \text{GetElementIndex}(B, p, 0, B.length);$   
         $iq = \text{GetElementIndex}(B, q, 0, B.length);$   
        return  $ip - iq$ ;  
    }  
}
```

input: sorted array  $B$ , element  $a$ , left, right  
out: index of  $a$

GetElementIndex( $B, a, left, right$ )

```
{  
    if ( $right == 1$ )  
        return right;  
     $mid = \frac{left + right}{2}$ ;  
    if ( $B[mid] == a$ )  
        return mid;  
    if ( $B[mid] < a$ )  
        return GetElementIndex( $B, a, mid+1, right$ );  
    if ( $B[mid] > a$ )  
        return GetElementIndex( $B, a, left, mid-1$ );  
    else  
        return false;  
}
```

$\therefore$  the running time of this is  $O(\log n)$