Solution 1:-

Recurrence: $T(n) <= 4T(n/2) + 5n^2$ for $n > 1$
$T(1) <= 5$ for $n = 1$

Claim: There exist a constant d>0 such that:
$T(n) <= dn^2 \log n + dn^2$ for every $n >= 1$

Proof by Strong Mathematical Induction:

Base Case:

n=1: we want: $T(n) <= dn^2 \log n + dn^2$

we know: $T(1) <= 5 <= d(1)^2 \log (1) + d(1)^2$

Thus, $5 <= d$

Induction Case:

n>1: we know:

$T(n)$
$<= 4T(n/2) + 5n^2$
$<= 4*[(dn^2/4)\log (n/4) + (dn^2/4)] + 5n^2$……. using induction hypothesis
$<= dn^2\log n – dn^2 + 5n^2$
$<= dn^2\log n + (5 – d) n^2 <= dn^2 \log n + dn^2$

$(5 – d)n^2 <= dn^2$
$5 – d <= d$
$5 <= 2d$

Thus, $d >= 5/2$.

Thus in order to satisfy both the base and induction case **d >= 5**

Solution 2:-

Pseudo Code (code snippet):

```
for(int i=0; i<len; i++)
        {
            if(arr[i] < min)
            {
                def = def + (min - arr[i]);
            }

            // If there is a deficiency at the left of the current stack...
            if (arr[i] > min && def != 0)
            {
                if (arr[i] >= (min + def - exc))
                {
                    trans = trans + "\n"+(i+1);

                    arr[i] = arr[i] - (def - exc);

                    i--;        // To go back to previous stack...

                    currDef = min - arr[i];
```

```
                    if(currDef < 0)
                        currDef = 0;

                    currExc = arr[i] - min;
                    if(currExc < 0)
                        currExc = 0;

                    trans = trans + " " + (i+1) + " " + (def - exc);
                    arr[i] = arr[i] + (def - exc);

                    def = def - currDef;
                    exc = exc - currExc;

                    steps++;
                    i--;        // To compensate for the for loop i++;
                }

                else
                {
                    exc = exc + arr[i] - min;
                }
            }

            // If there is deficiency to the right of the current stack...
            if (i >= 0 && arr[i] > min && def == 0)
            {
                trans = trans + "\n"+(i+1);
                currExc = arr[i] - min;
                arr[i] = arr[i] - currExc;
                arr[i+1] = arr[i+1] + currExc;

                trans = trans + " " + (i+2) + " " + currExc;

                steps++;
            }

            }
```

Explanation:

1. Calculate and store deficiency & excess of DvDs in each stack.
2. If current element can satisfy the deficiency keeping itself satisfied then take the DvDs = deficiency from this to previous stack.
3. Repeat step 2, till all the stacks till 1st stack is satisfied.
4. Move all the excess plates (if all the stacks before this stack is satisfied) to the stack to the right.
5. Repeat step 4, till all the stacks till the last one is satisfied.

The Algorithm is correct as it does satisfy all the conditions of the problem set. And solves the problem in O(n) time.

Estimated Running Time:

There is only one loop running from 0 to n, with few trips back to certain i-th value.
In worst case the Algo's running time would be O(2*n) = O(n).

Problem 3(a):

Pseudo Code:

a) SELECT-RAND (A, N/2) //N is length of array A

1. x = a[i] where i = a random number from {1,…,n}
2. rearrange A so that all elements smaller than x come before x, all elements larger than x come after x, and elements equal to x are next to x
3. j1,j2 = the leftmost and the rightmost position of x in rearranged A
4. if (N/2<j1) return SELECT-RAND(A[1…(j1-1)],N/2)
5. if (j1<=N/2<=j2) return x
6. if (N/2>j2) return SELECT-RAND(A[(j2+1)…n],N/2-j)

b)

```
for   i=1 to N
   if (x == A[i])
  count ++;
  If (count > N/2)
   Print( "Yes")
 Else
 Print ("No")
```

Explanation:

In a list of N numbers, if there exists a number which occurs more than N/2 times then indeed it's the median.

This can be proved because if we have a number having occurrence which is more than N/2 times, the number will cover either more than the first half or more than the second half or will lie somewhere in the middle (if the array is sorted), but surely will be the median as it will always cross the middle.

However, we are not sure that the median of any list will have a occurrence of more than N/2 or not. We implemented the Select Algorithm covered in the class to first find the median(putting K=N/2) and once the median has been found, a check is done for the occurrence of the median to be greater than N/2 or not.

Running Time:

The running time will be O(n) as the SELECT-RAND algorithm has an expected of O(n) and the second pat of the algorithm will also take O(n) time .

Problem 3(b):

Pseudo Code:

For k = N/3 and 2N/3

SELECT-RAND (A, k) //N is length of array A

1. x = a[i] where i = a random number from {1,…,n}
2. rearrange A so that all elements smaller than x come before x, all elements larger than x come after x, and elements equal to x are next to x
3. j1,j2 = the leftmost and the rightmost position of x in rearranged A
4. if (k<j1) return SELECT-RAND(A[1…(j1-1)],k)
5. if (j1<=k<=j2) return x
6. if (k>j2) return SELECT-RAND(A[(j2+1)…n],k-j)

Lets say x(N/3) and x(2N/3) are the values obtained from the above algorithm.

```
b)   for   i=1 to N
     for p = x(N/3) and x(2N/3)
        if (p == A[i])
       count ++;
       If (count > N/3)
        Print( "Yes") ;
       exit();
      Else
       Print("No");
```

Explanation:

In an list of N numbers ,if a number occurs more than N/3 times then that number will surely be the $N/3^{rd}$ or $2N/3^{rd}$ smallest number . But $N/3^{rd}$ and $2N/3^{rd}$ smallest number need not have the occurrence of more than N/3 .

This can be proved on the similar lines if we sort elements and if an element(say x)occurs more than N/3 times, it will surely cover the $N/3^{rd}$ position or the $2N/3^{rd}$ position. We can consider just three extreme cases.

First, all x's lie to the leftmost (or x is the smallest number) then as they are more than N/3 in number they will surely cross the $N/3^{rd}$ position.
Second, if they lie to the rightmost part(or x is the biggest) they will cross the $2N/3^{rd}$ position.
Third, if they lie exactly in the middle you will have less than N/3 numbers on the left of x's or less than N/3 on the right of x's so they surely will cover $N/3^{rd}$ or $2N/3^{rd}$ position.
Any other case will surely contain x at $N/3^{rd}$ or $2N/3^{rd}$ position

Running time:

Running time will again be O(n) as the same algorithm is now run twice which would not make any difference in the running time.