1. First, find the leftmost (x smallest) white point p1 and the rightmost (x largest) white point p2 in the picture, then p1 must be the leftmost point of region1, p2 is the rightmost point of region2, from these two Starting from the point, region1 to the right, region2 to the left, find the white points adjacent to them to change the color and recurse.

```
int x, y, x1, x2, y1, y2;
for(x=0; x<N; x++){
        for(y=0; y<N; y++){
                if(color[x,y]==white){
                        x1=x; y1=y;
                        return;
                }
        }
}

for(x=N; x>0; x--){
        for(y=N; y>0; y--){
                if(color[x,y]==white){
                        x2=x; y2=y;
                        return;
                }
        }
}

fillByRed(int x, int y){
        color[x,y]=red;
        if(color[x,y+1]==white){
                fillByRed(x,y+1);
        }
       if(color[x+1,y+1]==white){
                fillByRed(x+1,y+1);
        }
       if(color[x+1,y]==white){
                fillByRed(x+1,y);
        }
       if(color[x+1,y-1]==white){
                fillByRed(x+1,y-1);
        }
       if(color[x,y-1]==white){
                fillByRed(x,y-1);
        }
}
```

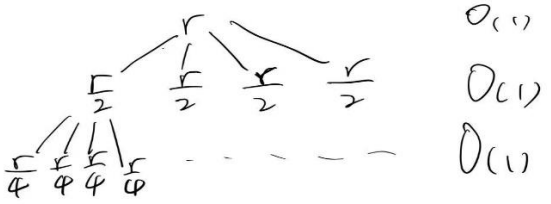
```
fillByBlue(int x, int y){
        color[x,y]=blue;
        if(color[x,y+1]==white){
                fillByBlue(x,y+1);
        }
        if(color[x-1,y+1]==white){
                fillByBlue(x-1,y+1);
        }
        if(color[x-1,y]==white){
                fillByBlue(x-1,y);
        }
        if(color[x-1,y-1]==white){
                fillByBlue(x-1,y-1);
        }
        if(color[x,y-1]==white){
                fillByBlue(x,y-1);
        }
}

fillByRed(x1,y1);
fillByBlue(x2,y2);
```

2. if (A[mid] > mid), then call binarySearch(A, 1, mid-1, mid)
   if (A[mid] < mid), then call binarySearch(A, mid+1, r, mid)
   time complexity: O(lgn)

3. (1)a=r;

```
DrawSquare(x, y, r)
        r1 = r/2;
        if r1==a/8
          return;
        DrawSquare(x-r, y+r, r1);
        DrawSquare(x+r, y+r, r1);
        DrawSquare(x-r, y-r, r1);
        DrawSquare(x+r, y-r, r1);
```

$$T(r) = \begin{cases} O(1), & \text{for } n=1 \\ 4T(r/2) + O(1), & \text{for } n \geq 2. \end{cases}$$

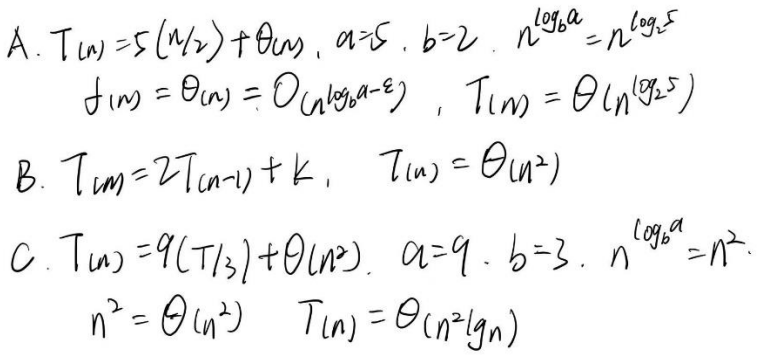


(2)(3)

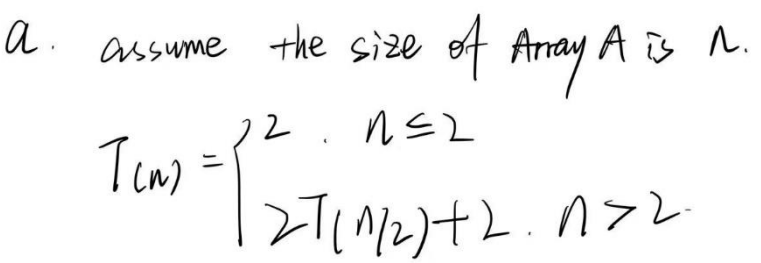
4. First, we split array A into two subarrays A1 and A2 of half size, then we find the majority elements lme and rme of these two subarrays, if they are same, this element is the majority element of A. If they are different, the next step is to calculate the frequency of these two majority elements. If frequency greater than floor(n/2)+1, the corresponding element is the majority element of A. If all the conditions are not met, no majority element.

```
k=floor(n/2)
lme = getMajorityElement(a[1...k])
rme = getMajorityElement(a[k+1...n]
if lme = rme:
        return elemlsub
lcount = getFrequency(a[1...n],lme)
rcount = getFrequency(a[1...n],rme)
if lcount > k+1:
        return lme
else if rcount > k+1:
        return rme
else
        return no majority element.
```

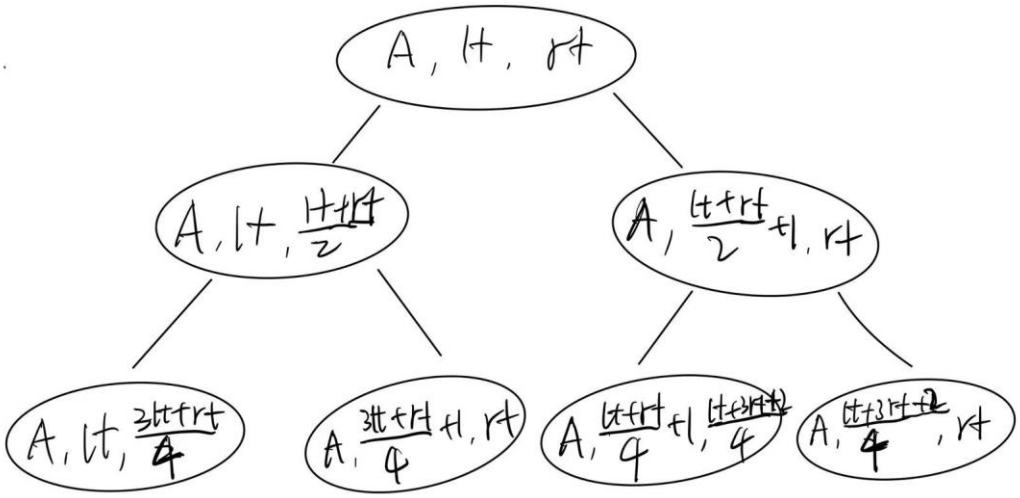
5. Assume T(n) is the total running time of algorithm A, so T(n)=5T(n/2)+O(n).
   Assume T(n) is the total running time of algorithm B, so T(n)=2T(n-1)+K.
   Assume T(n) is the total running time of algorithm C, so T(n)=9T(n/3)+O(n^2).
   Based on The Master Method:



6.



b.



b. $T(n) = 2T(n/2) + 2$

$= 2((2T(n/4)) + 2) + 2$

$= 4T(n/4) + 2^2 + 2$

$= 4(2T(n/8) + 2) + 2^2 + 2$

$= 8T(n/8) + 2^3 + 2^2 + 2.$

$= 2^k T(n/2^k) + 2^k + 2^{k-1} + 2^{k-2} + \cdots + 2$

$= \frac{n}{2} T(2) + \frac{n}{2} + \frac{n}{2^2} + \cdots + 2$

$= \frac{n}{2} T(2) + n - 2 = 2n - 2.$

C. base case: $n = 2$. $2n - 2 = 2$. (true)

when $n = 2^k$. $2n - 2 = 2^{k+1} - 2$

when $n = 2^{k+1}$ $2n - 2 = 2^{k+2} - 2$

$T(n) = 2T(n/2) + 2 = 2^k T(n/2^k) + 2^k + \cdots + 2$

when $n = 2^{k+1}$ $k = \lg n - 1$

$T(n) = 2^{\lg n - 1} T(n/2^{\lg n - 1}) + 2^{\lg n - 1} + \cdots + 2$

$= \frac{n}{2} T(2) + 2^{\lg n} - 2 = n + 2^{\lg n} - 2$

$= 2^{k+1} + 2^{k+1} - 2$

$= 2^{k+2} - 2$

7.

7. $T(n) = 16T(n/4) + n^4 + 3$. $a=16$, $b=4$. $n^{\log_b a} = n^2$

$f(n) = n^4 + 3 = \Omega(n^2)$ $T(n) = \Theta(n^4+3)$.

8.

8. $T(3) = 5T(2) - 8T(1) + 4T(0) = 2$

$n^3 = 5n^2 - 8n + 4 \Rightarrow n=1, n=2$

$\Rightarrow T(n) = C_1 1^n + C_2 2^n + C_3 n 2^n$

$T(0) = 0 \Rightarrow C_1 + C_2$

$T(1) = 1 \Rightarrow C_1 + 2C_2 + 2C_3$

$T(2) = 2 \Rightarrow C_1 + 4C_2 + 8C_3$

$\rightarrow C_1 = -2, C_2 = 2, C_3 = -\frac{1}{2}$

$T(n) = 2^{n+1} - \frac{1}{2} n \cdot 2^n - 2$

9.

9. (1) $O(n^3)$

(2)
```
sum=0
for i=1,2,...n
    sum=A[i]
    for j=i+1, j+2, i+3, ..., n.
        sum += A[j]
        store the result in B[i,j]
    End for
End for
```

$O(n^2)$

(3) outer loop will operate n times.

inner loop:

$(n-1)+(n-2)+\cdots+1$

$= \frac{n \times (n-1)}{2}$

So running time will be $O(n^2)$.

10. Step1: Find the two lines La and Lb closest to the vertical from each side of the vertical line (the absolute value of ai is largest), and these two lines must be visible.
Step2: find the first intersection points of these two lines from top to bottom, the lines of

these intersection points must be visible.
Step3: repeat step2.

O(n^2)