(Question 2)

شرح الگوریتم: ابتدا مربع $n \times n$ را به ۴ مربع مساوی $\frac{n}{2} \times \frac{n}{2}$ تقسیم می‌کنیم.

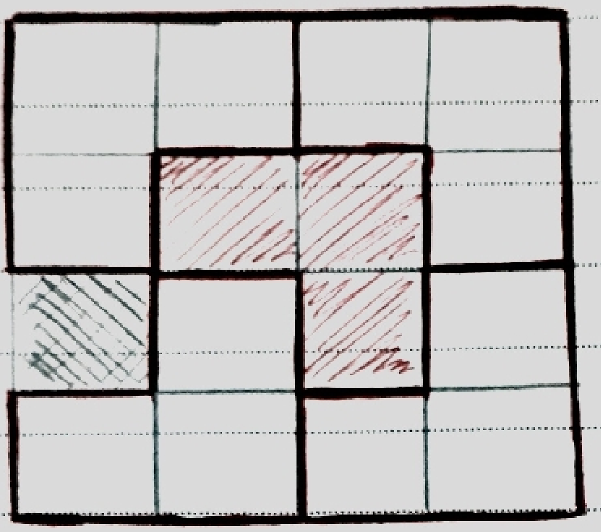
این یعنی مسئله‌ی بزرگ را به زیرمسئله‌های کوچکتر تقسیم کرده‌ایم ولی از اینجایی به بعد باز خانه‌های یکی از مربعها پر شده است لذا جنس زیرمسئله‌ها دقیقاً مشابه هم نیست برای حل این مورد خاص ۷ شکل را به گونه‌ای در وسط مربع $n \times n$ قرار می‌دهیم به طوری که از ۳ مربع این ۷ دیگر از خانه‌های ۳ تسمتی قرار گیرند و مربع پر شده‌ای ندارد. حال ۴ زیرمسئله با سایز $\frac{n}{2}$ داریم که از جنس مسئله‌ی اصلی است (هر زیرمسئله دارای یک خانه‌ی پر شده است)

و این روند را تا جایی ادامه می‌دهیم که به ۴ مربع 2×2 هرکدام با یک خانه‌ی پر شده برسیم حال می‌توان کاشی ۷ شکل را به گونه‌ای کنار داخل هرکدام از این مربع‌ها قرار داد. و با ادامه‌ی این روند کل سطح $n \times n$ با کاشی‌های ۷ شکل پر خواهد شد.

و پیچیدگی زمانی: $T(n) = 4T(\frac{n}{2}) + 1 \Rightarrow T(n) \in \theta(n^2)$

master theorem

Asman



مطابق شکل در این مثال خاص طی ۶ مرحله اسکلش مسئله به ۴ زیرمسئله ی کوچکتر و قرار دادن کاشی L شکل در وسط مربع بزرگ به گونه ای که هر ۴ مربع کوچکتر تشکیل شده یک خانه ی پر شده داشته باشند جواب مسئله و محل قرارگیری کاشی ها تا پر شدن مربع بزرگ مشخص می گردد.

Question 2)

می‌دانیم $x^{n} = x^{\lceil \frac{n}{2} \rceil} \times x^{\lfloor \frac{n}{2} \rfloor}$ می‌باشد لذا اگر مقادیر $x^{\lceil \frac{n}{2} \rceil}$, $x^{\lfloor \frac{n}{2} \rfloor}$ و را

محاسبه نماییم با ضرب این دو مقدار x^{n} حاصل می‌شود.

```
float power(int n, float x)
{   if(n==1)
      return x;
    else
      return power(⌈n/2⌉, x) * power(⌊n/2⌋, x);
}
```

برای تحلیل پیچیدگی این الگوریتم خواهیم داشت،

عمل ضرب بین $power(\lfloor \frac{n}{2} \rfloor, x)$ ← Basic operation

و $power(\lceil \frac{n}{2} \rceil, x)$

$T(n) = 2T(\frac{n}{2}) + 1 \Rightarrow$ طبق Master theorem چون (۱)

$\emptyset < \log_2^2 = 1$ لذا پیچیدگی این الگوریتم از $\Theta(n)$ می‌باشد.

این مسئله را بصورت $\log^n$ نیز میتوان حل کرد

```
float power(float x, int n)
{   if (n == 1)
        return x
    else
        float m = power(x, n/2)
        if(n % 2 == 0)
            return m * m
        else
            return m * m * x
}
```

$$T(n) = T\left(\frac{n}{2}\right) + \theta(1) \xrightarrow[\text{master}]{} T(n) \in \theta(\log^n)$$

Question 3)

الف) کافی است id هر نسخه کفش را با id نسخه کفش مجاورش مقایسه کنیم

در صورتی که هیچ یک از نسخه های مجاور با نسخه i d یکسانی با آن نداشتند نسخه تک همان

نسخه کفش است این الگوریتم برای n نسخه n جستجو انجام می دهد لذا از $\theta(n)$ می باشد.

```
keytype Shoe( keytype arr[ ], int n)
{
    for index i = 0 to n-1
        if(arr[i] != arr[i+1] and
           arr[i] != arr[i-1])
            return arr[i];
}
```

keytype = id type

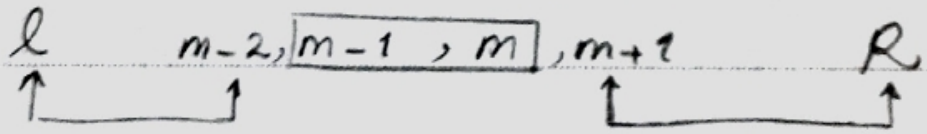
$2k+1 = n$ = تعداد نسخه کفش ها

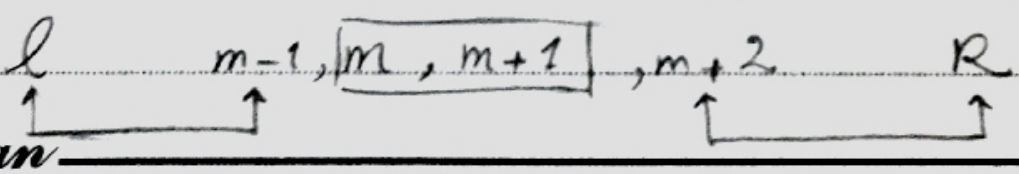
arr = آرایه ای که id نسخه کفش ها به شکل صعودی در آن چیده شده است.

و با اندیس 0 تا $n-1$ شماره گذاری شده.

ب) با استفاده از روش تقسیم و غلبه می‌توان الگوریتمی با پیچیدگی $O(\log n)$ ارائه داد.

```
keytype Shoe(keytype arr[], index l, index R)
{  if (R == l) return arr[l] (or arr[R])
   index m = l + (R-l)/2
   if( arr[m] != arr[m+1] and arr[m] != arr[m-1])
       return arr[m]
   else if (arr[m] == arr[m-1])
       if ((R - m) % 2 == 0)
           return Shoe(arr, l, m-2)
       else
           return Shoe(arr, m+1, R)
   else if (arr[m] == arr[m+1])
       if ((m-l) % 2 == 0)
           return Shoe(arr, m+2, R)
       else
           return Shoe(arr, l, m-1)
}
```





الگوریتم فوق ابتدا بررسی می کند عضوی که آیا تک عضوی است id همان تک عضو را

به عنوان کلید کفش یکتا برمی گرداند و در غیر اینصورت :

عنصر وسط آرایه را بررسی می کند در صورتی که با هیچ یک از 2 عنصر مجاورش id یکسانی

نداشت یعنی همان کلید کفش تک است پس id آن را برمی گرداند.

و در غیر اینصورت ابتدا مشخص می کنیم جفت نظیر آن عنصر چپ یا راستش است

پس بدون در نظر گرفتن این جفت بررسی می کنیم کدام یک از زیر آرایه های سمت

راست و سمت چپ آن عناصر تعداد فرد عنصر است (می دانیم کلید کفش

یکتا در بخشی است که دارای تعداد فرد عنصر باشد)

لذا جستجو را در هر مرحله به زیر آرایه ی حاوی عناصر سمت چپ یا راست جفت

متذکور محدود می کنیم پس می توان گفت در هر مرحله آرایه به حالت نصف تبدیل است

لذا خواهیم داشت ،

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1) \xrightarrow{\text{master theorem}}$$

$$T(n) \in \Theta(\log n)$$

Question 4)

روش حل: آرایه را ۲ قسمت کرده فرض می‌کنیم طولانی‌ترین پیوند مشترک هر قسمت بدست آمده است حال برای بدست آوردن طولانی‌ترین پیوند مشترک کل آرایه کافی است این ۲ پیوند مشترک را با کاراکتر به کاراکتر مقایسه کرده تا جایی که یکسان هستند پیش برویم بدین شکل طولانی‌ترین پیوند مشترک اعضای آرایه را بدست می‌آوریم.

```
String Max_Commen_String(string arr[], index l, index R)
{
    if (l == R) return arr[l] (or arr[R])
    else
        index mid = l + (R-l)/2
        String maxR = Max_Commen_String(arr, l, mid)
        String maxl = Max_Common_String(arr, mid+1, R)

        String res = " "
        for i = 0 to min(maxR.Lenght, maxl.lenght)
            if (maxR[i] == maxl[i])
                res += maxR[i]
            else break
    return res
}
```

تحلیل الگوریتم ۲: درصورتی که طول بزرگترین عضو آرایه x باشد خواهیم داشت

$T(n) = 2T(\frac{n}{2}) + O(x) \quad T(1) = 0$

درصورتی که x یک const باشد و ابستگی به طول آرایه نداشته باشد خواهیم داشت

(x = const) (درحالت کلی)

$T(n) = nx - x \Rightarrow T(n) \in O(nx) \Rightarrow T(n) \in O(n)$

از لحاظ پیچیدگی حافظه این الگوریتم در هر مرحله ترکیب ۲ زیرآرایه نیاز به یک متغیر

String با طول حداکثر x می باشد چنانچه n تعداد اعضای آرایه باشد پیچیدگی حافظه

در بدترین حالت $O(x \log^n)$ خواهد بود

است پس $f(n) \in O(x \log^n)$ = پیچیدگی حافظه

و چنانچه x یک const باشد و به n بستگی نداشته باشد $f(n) \in O(\log^n)$