بسمه تعالی

گزارش تمرین ۷ سوالات تحلیلی

ساختمان داده

محسن کربلائی امینی، ۹۸۲۴۲۱۲۸

دی ۱۴۰۲

سوال ۱:

آرایه اصلی:

[۱, ۴, ۷, ۸, ۹, ۱۲, ۱۵, ۱۷, ۱۸, ۲۰, ۲۲, ۲۵, ۲۷, ۳۰, ۳۲, ۳۴, ۳۶, ۳۹, ۴۱, ۴۳, ۴۵, ۴۸, ۵۰, ۵۳, ۵۵, ۵۸, ۶۰, ۶۳, ۶۶, ۶۹]

بعد از ۱۰ جابه‌جایی:

[۸, ۱۵, ۲۵, ۵۰, ۷, ۱۲, ۴, ۱۷, ۱۸, ۲۰, ۲۲, ۹, ۲۷, ۳۰, ۳۲, ۳۴, ۳۶, ۳۹, ۴۱, ۴۳, ۴۵, ۴۸, ۱, ۵۳, ۵۵, ۶۰, ۶۹, ۶۳, ۶۶, ۵۸]

مانند مرتب‌سازی رشته‌ای عمل می‌کنیم و دسته‌بندی‌های مرتب از داده‌ها را بدست می‌آوریم. O(n)

[۸, ۱۵, ۲۵, ۵۰]

[۷, ۱۲]

[۴, ۱۷, ۱۸, ۲۰, ۲۲]

[۹, ۲۷, ۳۰, ۳۲, ۳۴, ۳۶, ۳۹, ۴۱, ۴۳, ۴۵, ۴۸]

[۱, ۵۳, ۵۵, ۶۰, ۶۹]

[۶۳, ۶۶]

[۵۸]

سپس این دسته‌ها را دو به دو ادغام می‌کنیم. از آنجایی که تنها ده swap هم اتفاق افتاده است، تعداد این دسته‌ها نمی‌تواند نزدیک یک تابعی از n شود، بنابراین این مورد هم از O(n) تبعیت می‌کند.

```
array merge(array۱,array۲) {
    array result;
    for (i=۰,j=۰; i<array۱.size() || j < array۲.size() ; )
    {
        if (array۱[i] < array۲[j])
        {
            result.add(array۱[i]);
            i++;
        }
        else{
            result.add(array۲[j]);
            j++;
        }
    }
    return result;
}
```

```
for (i=۰,j=۰ ; i < arr.size() ; i++)
{
    group[j].add(arr[i]);
    if (i+۱ == arr.size()+۱ || arr[i+۱] < arr[i])
        j++;
}
j--;
result = merge(group[۰],group[۱])
for (i=۲ ; i < j ; i++)
{
    result=merge(result,group[i]);
}
```

سوال ۲:

- Separate Chaining: در این روش، برای حل مسئله تصادم، برای هر خانه، یک linkedList در نظر گرفته می‌شود، که در صورت رخداد تصادم، عناصر درون آن قرار گیرند.

در این روش فقط یک حالت وجود دارد که هیچ خانه‌ای در ۵ خانه اول خالی نماند: تمام ۵ درج، در خانه‌های ۱ تا ۵ اتفاق بیوفتند. یعنی اگر هر ۵ درج در خانه ۱ هم اتفاق بیوفتد، باز ۴ خانه خالی خواهند بود. چرا که هر ۵ درج در لینکدلیست خانه یک درج شده است. با اعمال ترتیب، می‌توان گفت در ۵! حالت میتواند این اتفاق رخ دهد.

تعداد کل حالات: ۵^۱۰۰ چرا که هر درج ممکن است بر روی هر کدام از ۱۰۰ خانه اتفاق بیوفتد.

بنابراین داریم:

$$P(a) = \frac{۵ * ۴ * ۳ * ۲ * ۱}{۱۰۰^{۵}}$$

- Open Addressing: در این روش وقتی یک تصادم رخ می‌دهد، عنصر مورد نظر درون خود hashtable درج می‌شود، به این شکل که به خانه‌های بعدی که خالی باشند، ارسال می‌شود. یعنی در صورت درج دو عنصر در خانه ۱۰۰، خانه ۱ پر خواهد شد.

در این حالت، علاوه بر ۵! حالت بدست آمده در روش قبلی، باید حالت‌های درج دیگر مانند ۱،۱،۱،۱،۱ و یا ۱،۲،۲،۲،۲ را هم در نظر داشته باشیم. که باعث پرشدن تمام ۵ خانه اول می‌شوند.

چالش محاسبه این مورد در این می‌باشد که در صورت داشتن دو ۵، خانه ۶ پر می‌شود و یک خانه خالی می‌ماند. به دلیل ذیق وقت ادامه حل این سوال به درس آمار احتمال ارجاع داده می‌شود. D :

سوال ۳:

$$x^{٢} mod\ ٧ = ٤, ٠ < x \leq ١٠٠$$

$$x\ mod\ ٧ * x\ mod\ ٧ = ٤ => x\ mod٧ = ٢,٥ => x = ٧k + ٢\ or\ x = ٧k + ٥$$

برای بدست آوردن این تعداد می‌توان از کد جاوای زیر کمک گرفت.

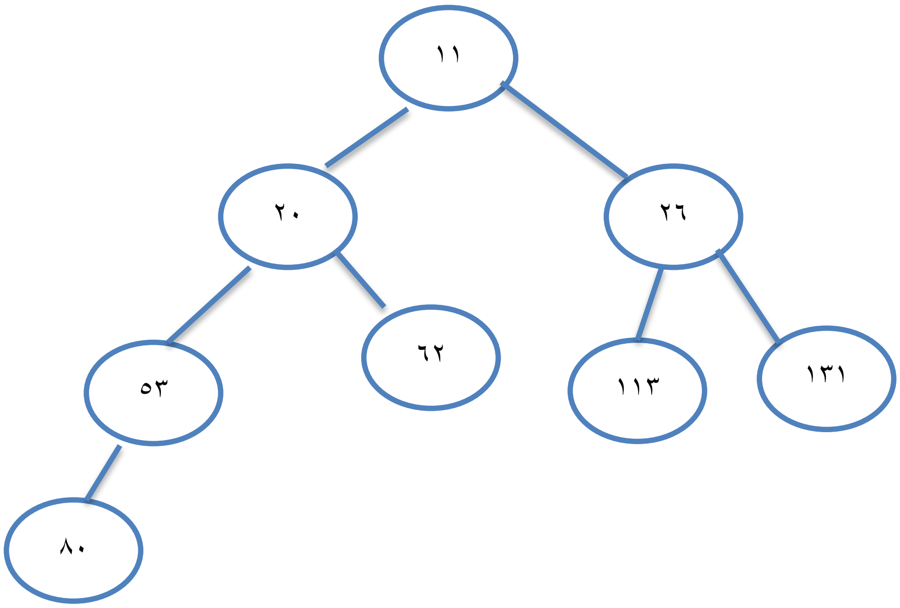
```
public class Three {
    public static void main(String[] args) {
        int counter=٠;
        for (int x = ٠; x <= ١٠٠; x++) {
            if (x % ٧ == ٢ || x % ٧ == ٥) {
                counter++;
            }
        }
        System.out.println(counter);
    }
}
```

در نتیجه ۲۹ عدد در معادله بالا صدق می‌کنند.

سوال ۴:

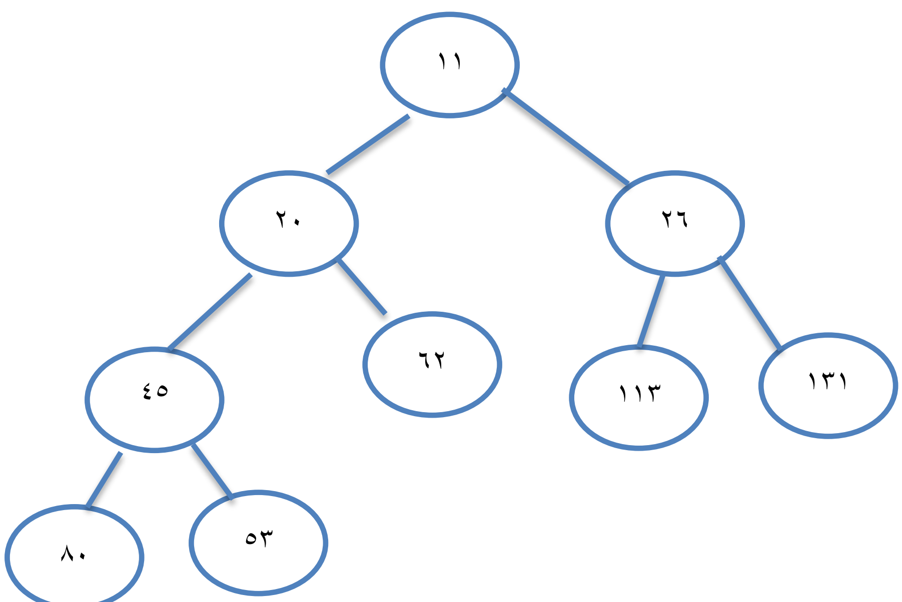
سوال ۵:

- Pop:



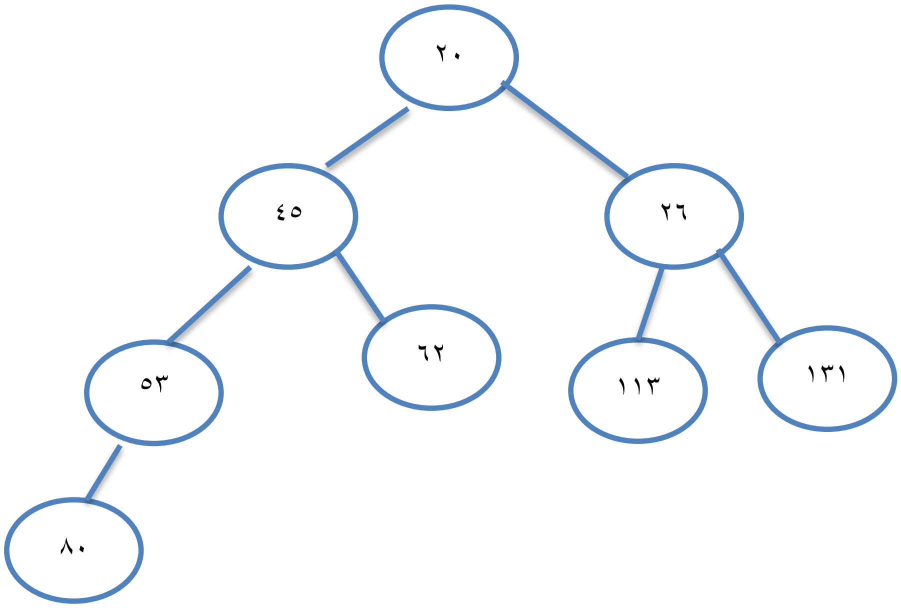
۱۱	۲۰	۲۶	۵۳	۶۲	۱۱۳	۱۳۱	۸۰
۱	۲	۳	۴	۵	۶	۷	۸

- Push ۴۵:



۱۱	۲۰	۲۶	۴۵	۶۲	۱۱۳	۱۳۱	۸۰	۵۳
۱	۲	۳	۴	۵	۶	۷	۸	۹

- :Pop



۲۰	۴۵	۲۶	۵۳	۶۲	۱۱۳	۱۳۱	۸۰
۱	۲	۳	۴	۵	۶	۷	۸

سوال ۶:

بله می‌توان. با یک روش ساده می‌توانیم هنگام درج در `min heap` مقدار قرینه عدد مورد نظر را ذخیره کنیم و هنگامی که یک عنصر را فرامی‌خوانیم، مقدار قرینه عدد ذخیره شده را برگردانیم که همان مقدار اصلی می‌باشد. این امکان به این دلیل موجود است که برای اعداد منفی با قدر مطلق بزرگ‌تر مقادیر کوچک‌تری را خواهیم داشت. به این ترتیب مقدار اصلی بزرگ‌تر به بالای `min heap` منتقل می‌شود و در نهایت مقدار ریشه `heap`، بزرگ‌ترین مقدار خواهد بود.

سوال ۷:

از عنصر اول آرایه شروع می‌کنیم. در این عنصر باید عنصر کمینه آرایه قرار بگیرد. عنصر کمینه آرایه حداکثر در خانه `k` ام آرایه قرار دارد. با یک جست‌وجویی `binary` می‌توان این عنصر را پیدا کرد و در سر جایش قرار داد(به صورت `swap` با عنصر حال حاضر – که این موضوع همچنان تغییری در شرط فاصله `k` از جایگاه ایجاد نخواهد کرد چرا که داریم از ابتدا حرکت می‌کنیم و عنصر جابه‌جا شده یا در سر جای مناسبش قرار گرفته و یا در ادامه الگوریتم خواهیم دید که باز به عقب‌تر باز خواهد گشت.)

سپس برای عنصر دوم و بعدی نیز این کار را تکرار می‌کنیم، هر بار کمینه تا k عنصر جلوتر پیدا و swap می‌شود. پیچیدگی زمانی این الگوریتم از O(nlogk) خواهد بود. چرا که تقریبا به ازای هر عنصر، یک جست‌وجوی binary بروی یک بازه به اندازه k خواهیم داشت.

سوال ۸:

بزرگ‌ترین عدد در یک min heap قطعا در برگ‌ها حضور دارد. برای پیدا کردن بزرگ‌ترین عدد کافیست بتوانیم تعداد برگ‌ها را بیابیم و از آخر آرایه تا تعداد مورد نظر مقدار حداکثر را پیدا کنیم.

در صورتیکه تعداد کل عناصر n باشد، تعداد برگ‌ها در یک درخت دودویی کامل برابر است با:

$$l = ۲^h$$

در این صورت با بررسی مقدار l عنصر آخر آرایه می‌توان مقدار حداکثر را پیدا کرد. بنابراین پیچیدگی زمانی این کار از o(l) خواهد بود.