هر سوال را در محل در نظر گرفته شده پاسخ دهید. پاسخ های خارج از محل تصحیح نمی‌شوند.

۱. [۶] درستی یا نادرستی عبارات زیر را با علامت (✓) یا (✗) مشخص کنید. دلیل خود را در نقطه چین زیر هر عبارت توضیح دهید. نمره کامل فقط به جواب صحیح با توضیح صحیح تعلق میگیرد.

برای توضیح جواب‌ها به اسلایدهای مربوطه مراجعه کنید.

(آ) $n = \mathcal{O}(n \log_2 n)$ ✓
(ب) $n \log_2 n = \mathcal{O}(n)$ ✗
(ج) $n^2 = \mathcal{O}(2^n)$ ✓
(د) $2^n = \mathcal{O}(n^2)$ ✗
(ه) $n = \theta(2^{50} n)$ ✓
(و) $n = \Omega(\log n)$ ✓

2. [21] Give the order of growth (as a function of N) of the running times of each of the following code fragments. Justify your answer in the space provided.

(a) $O(\underline{\ \mathbf{N}\ })$, sum=$\underline{N + \frac{N}{2} + \frac{N}{4} + \ldots}$

```
int sum = 0;
for (int n = N; n > 0; n /= 2)
    for (int i = 0; i < n; i++)
        sum++;
```

See answer to exercise 6 here

(b) $O(\underline{\ \mathbf{N}\ })$, sum=$\underline{1 + 2 + 4 + 8 \ldots}$

```
int sum = 0;
for (int i = 1; i < N; i *= 2)
    for(int j = 0; j < i; j++)
        sum++;
```

See answer to exercise 6 here

(c) $O(\underline{N \log N})$, sum=$\underline{N \log N}$

```
int sum = 0;
for (int i = 1; i < N; i *= 2)
    for (int j = 0; j < N; j++)
        sum++;
```

See answer to exercise 6 here

(d) $O(\underline{N \log_4 N})$, sum=$\underline{N \log_4 N}$

```
int sum = 0;
for (int i = 1; i < N; i <<= 2)
    for (int j = 0; j < N; j++)
        sum++;
```

Shift left by 2 bits is equal to multiple by 4. So, similar to previous part except it is log base 4.

(e) $O(\underline{\ y\ })$, fn1(x,y)=$\underline{\quad x \times y \quad}$

```
int fn1(int x, int y)
{
    if (y == 0) return 0;
    return (x + fn1(x,y-1));
}
```

fn1(x,y) = x + fn1(x, y-1) = x + x + fn1(x, y-2) = x + x + x + fn1(x, y-3) ...= x * y + fn1(x, 0) = x * y

(f) $O(\underline{\ \mathbf{b}\ })$, fn2(a,b)=$\underline{\quad a^b \quad}$

```
int fn2(int a, int b)
{
    if (b == 0)
        return 1;

    return fn1(a, fn2(a, b-1));
}
```

fn2(a, b) = a * fn2(a, b-1) = a * a * fn2(a, b-2) ...= a^b * fn2(a, 0) = a^b

(g) $O(\underline{\quad N \log_3 N \quad})$

```
void fn(int n) {
    if (n <= 1) return;
    fn(n/3);
    for (int i = 0; i < n; i++)
        op(); // O(1) operation
    fn(n/3);
    fn(n/3);
}
```

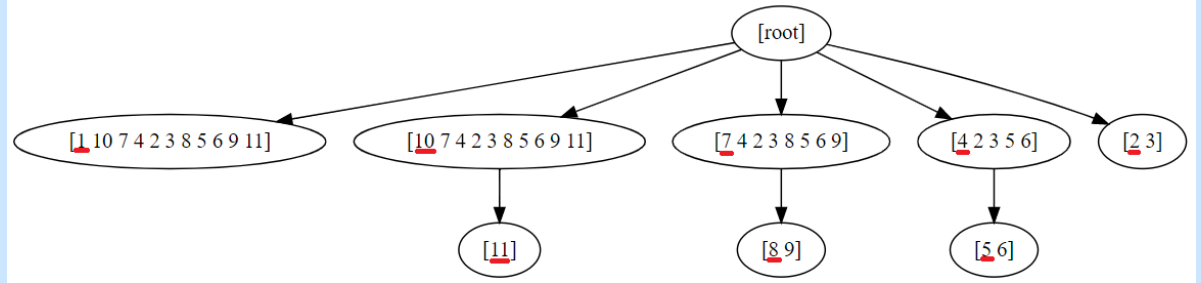
$$T(n) = 3T(n/3) + O(n)$$

Use master theorem. Question from Princeton university Spring 2019 mid term exam. link.

۳. [۱۵] درخت بازگشتی و تکنیک Tail Recursion

(آ) [۱۰] درخت بازگشتی الگوریتم QuickSort را برای آرایه زیر با استفاده از تکنیک Tail Recursion متوازن و انتخاب اولین عنصر به عنوان pivot رسم کنید.

$$A = [1, 10, 7, 4, 2, 3, 8, 5, 6, 9, 11]$$

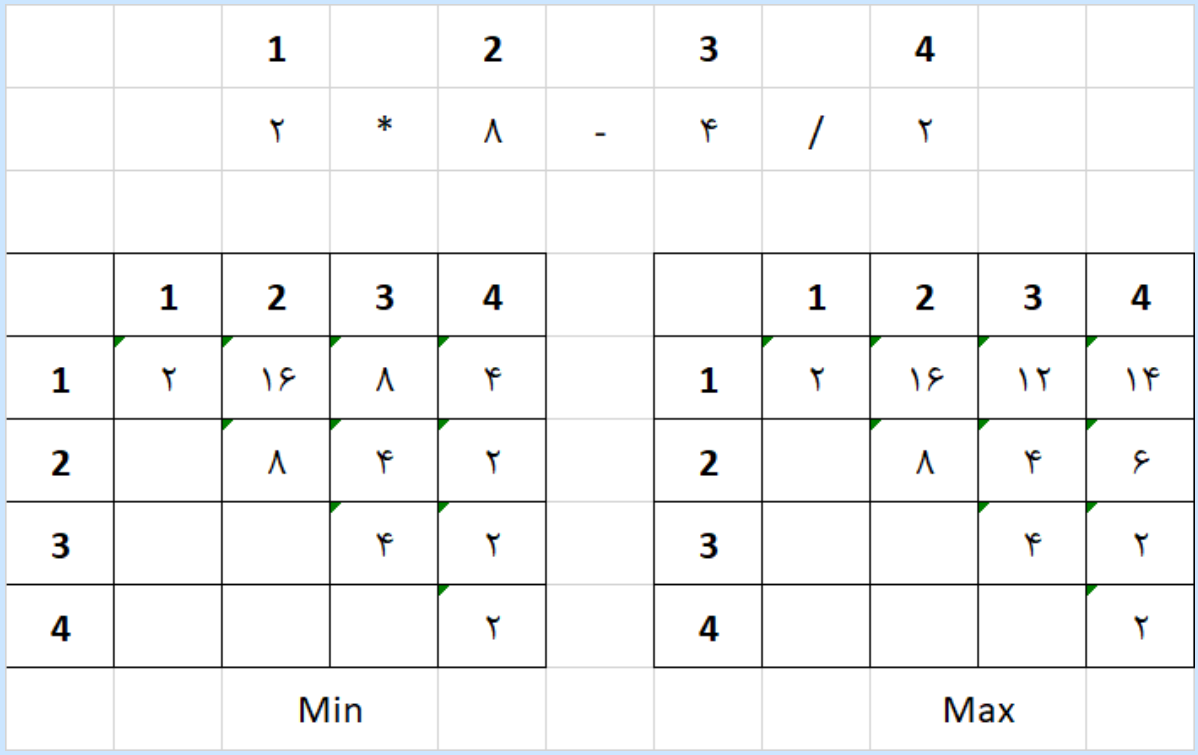


(ب) [۵] فایده استفاده از این تکنیک چیست؟

کاهش عمق Stack: در این روش نیاز به استفاده از حافظه کمتری

۴. [۱۲] دنباله اعداد و عملگرهای زیر را در نظر بگیرید.می‌دانیم از دنباله مقابل با انواع پرانتزگذاری (که ترتیب انجام عملیات ها را مشخص می‌کند) می‌توان مقادیر مختلفی را به دست آورد. با استفاده از (Dynamic programing) ماکسیمم این مقادیر را بدست بیاورید و روند فرایند را مرحله به مرحله بنویسید. عملگرهای مجاز جمع، ضرب، تقسیم و بعلاوه هستند و کلیه اعداد صحیح، مثبت و بزرگتر از یک می‌باشند.

$$2 \times 8 - 4 \,/\, 2$$



۵. [۱۸] طراحی ساختمان‌داده‌ای برای ذخیره و بازیابی مختصات دکارتی (شامل x و y): هدف طراحی ساختمان داده‌ای است با امکان انجام عملیات زیر:

۱. اضافه کردن (مختصات) یک نقطه
۲. پیدا کردن نقطه با کمترین x
۳. پیدا کردن نقطه با کمترین y
۴. حذف نقطه با کمترین x
۵. حذف نقطه با کمترین y

چنانچه بیش از یک نقطه با x یکسان، یا y یکسان موجود بودند، می‌توانید بدلخواه از بین آنها انتخاب کنید. برای گرفتن نمره کامل لازم است هریک از موارد بالا در بدترین حالت در زمان $O(\log n)$ پایان یابند. در صورت استفاده از ساختمان‌داده‌های شناخته شده، گفتن روش استفاده یا تغییر لازم است ولی گفتن جزییات پیاده‌سازی آن ساختمان داده لازم نیست. هر فرضی که برای حل مساله انجام می‌دهید را بنویسید.

(آ) ساختمان داده‌ای مورد استفاده خود را با اجزاء آن توضیح دهید. چنانچه ساختمان داده‌ای جدیدی است تعریف کلاس را ارائه دهید. چنانچه از ساختمان‌داده شناخته شده‌ای استفاده می‌کنید، چگونگی تغییر یا استفاده از آن را توضیح دهید.

جواب سوال ۷ پاسخنامه. امتحان میان‌ترم دانشگاه Princeton را ببینید

(ب) الگوریتم/روش اضافه کردن نقطه را توضیح دهید.

(ج) روش پیدا کردن/باز گرداندن نقطه با کمترین x یا کمترین y را توضیح دهید. درصورت نیاز می‌توانید از شبه‌کد استفاده کنید.

(د) روش حذف کردن نقطه با کمترین x یا کمترین y را توضیح دهید.

(ه) پیچیدگی محاسباتی بدترین حالت برای n عملیات به هر ترتیبی با شروع از مجموعه نقاط تهی چیست:

$\theta($_____$)$

(و) درستی جواب قسمت قبل را توضیح دهید.

۶. [۱۶] درخت جستجوی دو-دویی (Binary Search Tree)

(آ) تفاوت درخت AVL و درخت قرمز-سیاه را توضیح دهید. هر کدام در چه موردی بر دیگری مزیت دارد؟

حداکثر تفاوت فاصله برگ از ریشه در درخت AVL یک است. در صورتیکه در درخت قرمز سیاه حداکثر تفاوت ضریب ۲ می‌باشد. با وجود اینکه پیچیدگی محاسباتی هر دو در بدترین حالت $\log n$ است، اما ضریب ثابت در دو درخت متفاوت می‌باشد. درخت AVL در هنگام Insert هزینه بیشتری نسبت به درخت قرمز-سیاه دارد و در ازای این هزینه ارتفاع کمتری را تضمین می‌کند. در نتیجه اگر تعداد جستجوها از تعداد اضافه‌کردن‌ها خیلی بیشتر باشد، درخت AVL بهتر عملی می‌کند (و بالعکس).

(ب) درخت قرمز-سیاه چپ گرا (Left Leaning Red-Black Tree) بر درخت قرمز-سیاه معمولی/ساده چه مزیتی دارد؟

مزیت درخت چپ‌گرا سادگی پیاده‌سازی است بطوریکه همیشه سه عمل را در صورت وجود شرایط آن بترتیب اجرا کرده و خواص درخت حفظ می‌شوند. در درخت چپ‌گرا، تنها کدی که لازم است برای بالانس نگه داشتن درخت اضافه بشود، سه خط زیر است:

```
if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
if (isRed(h.left) && isRed(h.right)) flipColors(h);
```

(ج) ساختمان داده Splay Tree:

i. پیچیدگی محاسباتی در بدترین حالت: $O(\_N\_)$

در صورتیکه اعداد به ترتیب اضافه شوند درخت می به حالت لیست پیوندی درآمده و پیچیدگی محاسباتی $O(N)$ می‌شود.

ii. پیچیدگی محاسباتی تحلیل سرشکن[1] $O(\underline{\log N})$

برای توضیح اسلایدهای مربوطه را ببینید.

iii. در چه حالتی بر درخت AVL مزیت دارد؟

بستگی به ترتیب اضافه/جستجو کردن، درخت Splay می‌تواند پیچیدگی محاسباتی بسیار بهتری داشته باشد. بطور کلی چنانچه برخی کلیدها بیش از بقیه جستجو شوند، پیچیدگی محاسباتی برای آنها بهتر خواهد بود. جزئیات دقیقتر در اسلاید موجود است.

iv. مزیت اثبات نشده‌اش[2] بر هر درخت جستجوی دو-دویی؟

حدس ریاضی بر این است که به ازای هر نوع/ترتیب از ورودی چنانچه یه درخت جستجوی دو-دوئی بهینه طراحی و پیاده‌سازی شود، پیچیدگی محاسباتی درخت Splay برای آن ورودی حداکثر به اندازه ضریب ثابتی بیشتر خواهد بود. این حدس ریاضی به Dynamic Optimality Conjecture معروف است.

۷. [۱۶] وب‌سایت haveibeenpwned.com قابلیت دسترسی به $9,319,487,388$ پسورد لو رفته را فراهم می‌کند. شما می‌توانید در این وب‌سایت پسورد خود را وارد کرده و ببینید آیا اطلاعات مربوط به اکانتی با چنین پسوردی قبلا لو رفته یا نه. برای حفظ اطلاعات خصوصی کاربران در این وب‌سایت دانلود اصل پسوردها امکان پذیر نیست. بجای آن فایلی که هر خط آن هش[3] آن پسورد با الگوریتم SHA-1 موجود است در قالب فایلی زیپ شده‌ای با اندازه ۱۱ گیگابایت قابل دانلود است. اندازه هر هش SHA-1 برابر ۲۰ بایت می‌باشد. ساختمان داده‌ای برای امکان فراهم کردن جستجو در این لیست را فراهم کنید. فرض کنید تابعی با نام SHA1 موجود است که یک string به عنوان ورودی گرفته و هش متناظر با آن را برمی‌گرداند.

(آ) ساختمان داده‌ای مورد استفاده خود را توضیح دهید.

از یک HashTable استفاده می‌کنیم. اضافه کردن به این ساختمان داده‌ای و جستجو در آن برای تابع هش و اندازه مناسب $O(1)$ می‌باشد. کلید در اینجا مقدار هش SHA1 برای هر پسورد می‌باشد. نیاز به منتسب کردن یک مقدار خاص به هر کلید نیست . از این جهت از HashSet هم می‌توان استفاده کرد یا مقدار منتسب به کلیدهای موجود را برابر true در نظر گرفت.

(ب) شبه‌کد تابع Load را بنویسید.

```
void Load(file)
{
    HashSet hs = new HashSet();
    foreach(line in file)
        hs.Add(line)
}
```

[1] amortized
[2] Conjecture
[3] Hash

(ج) شبه کد تابع Find را بنویسید.

```
bool Find(string pwd)
{
    var hash = SHA1(pwd);
    return hs.Contains(hash);
}
```

(د) اندازه دقیق مقدار حافظه مورد استفاده را بنویسید. اندازه هر اشاره‌گر یا reference را ۴ بایت در نظر بگیرید. هر فرض لازم برای جزئیات پیاده‌سازی، رند کردن اعداد و جلوگیری از نیاز به ماشین‌حساب را انجام داده و بنویسید. مثلا می‌توانید برای راحتی تعداد پسوردها را ۹ میلیارد در نظر بگیرید. در قدم‌های میانی هم به همین ترتیب. ولی لازم است فرض خود را بنویسید.

فرض می‌کنیم جدول هش بصورت دینامیک بزرگ شده و در انتها ۷۰ دصد آن پر باشد. همچینین فرض می‌کنیم ۹ میلیارد هش پسورد داشته باشیم. در نتیجه اندازه جدول هش ما برابر $\frac{1}{0.7} \times 9 \times 10^9$ خواهد بود. فرض می‌کنیم که در هر خانه از جدول هش یک لیست پیوندی موجود است با توجه به اینکه اندازه اشاره‌گر ۴ بایت است در نتیجه مقدار فضای مورد استفاده آرایه مربوط به جدول هش برابر $\frac{1}{0.7} \times 9 \times 10^9 \times 4$ خواهد بود. برای لیست‌های پیوندی به ازای هر عنصر مقدار هش و یک اشاره‌گر باید نگهداری کنیم. لذا حجم کل حافظه مورد استفاده برای لیست‌های پیوندی برابر $9 \times 10^9 \times (20 + 4)$ می‌باشد. در نتیجه مقدار کل حافظه برابر عبارت زیر می‌باشد.

$$(\frac{1}{0.7} \times 9 \times 10^9 \times 4) + (9 \times 10^9 \times (20 + 4))$$

(ه) [۹] (امتیازی) چگونه می‌توان اپ موبایل با محدودیت حافظه برای منظور بالا طراحی کرد؟

i. از چه ساختمان داده‌ای یا الگوریتمی می‌توان استفاده کرد؟

از ساختمان داده BloomFilter استفاده می‌کنیم.

ii. نسبت به پیاده‌سازی بدون محدودیت حافظه چه اشکالی از نظر پیچیدگی محاسباتی، دقت یا موارد دیگر دارد؟ تغییر محدودیت حافظه چه تاثیری روی این اشکال(ها) می‌گذارد؟

پیچیدگی محاسباتی اضافه کردن و جستجو در این ساختمان داده برابر تعداد توابع هشی است که برای آن در نظر می‌گیریم. در این ساختمان داده‌ای عدم وجود پسورد با دقت ۱۰۰ درصد می‌باشد. ولی درصورتیکه ساختمان داده‌ای وجود آن را برگرداند، با احتمالی بین صفر تا یک، این پسورد در لیست موجود می‌باشد. مقدار این احتمال بستگی به تعداد توابع هش و تعداد بیت‌های در نظر گرفته شده می‌باشد. هر چه تعداد بیت‌ها بیشتر باشد، احتمال به یک نزدیکتر می‌شود.

iii. تابع Load این اپ موبایل را توضیح دهید.

هنگام ساختن BloomFilter توابع هش استفاده شده و آرایه بیت‌های نهایی باید در یک فایل ذخیره شوند. در هنگام Load شدن اپ، آرایه بیت و توابع هش باید از فایل خوانده شده و در حافظه جایگزاری می‌شود.

۸. [۴] (امتیازی) حمله Hash Flooding و راه حل مقابله با آن را توضیح دهید.

این حمله به این ترتیب است که با استفاده از دانش تابع هش مورد استفاده در یک سرویس/سرور مجموعه‌ای از کلیدها که مقدار هش یکسانی دارند را به سرور داده. به این ترتیب سرور همه کلیدها را در یک خانه از جدول هش ذخیره می‌کند که منجر به کند شدن سرور و در نهایت از کار افتادن سرویس می‌شود.

راه حل مقابله با این حمله، غیر قابل پیشبینی کردن تابع هش با استفاده از یک مقدار تصادفی می‌باشد. برای اطلاعات بیشتر اینجا را ببینید.