۹۸۴۱۱۴۳۲ — بسم الله الرحمن الرحیم — محمد عرفان زارع زرینی ①

Subject
Date

①

1)
```
void F1(int n, int sum){            c1   100n+1
    for(int i=0; i<n*100; ++i){      c2   (Σ_{p=1}^{N} p)+1
        for(int j=n; j>0; j--)      c3   Σ_{p=1}^{N} p
            sum++
        for(int k=0; k<i; ++k)      c4   (Σ_{p=1}^{100n} p)+1
            sum++                   c5   Σ_{p=1}^{100n} p
    }
}
```

$$\Rightarrow (100n+1).c_1\left(c_2\left(\sum_{p=1}^{N} p\right)+1\right)+c_3.\sum_{p=1}^{N} p+c_4.\left(\sum_{p=1}^{100n} p\right)+1+c_5.\sum_{p=1}^{100n} p$$

$$=((100n+1).c_1)(c_2(n+1)+c_3 n+c_4(100n+1)+c_5\, 100n) \qquad c_4+c_5=a$$

$$=(100n+1)c_1)(c_2(n+1)+c_3 n+a(100n))=(100n\, c_1)(b(n+1)+a(100n)) \qquad c_2+c_3=b$$

$$=O(100n^2)=O(n^2)$$

2)
```
void F2(int n, int sum){
    int j=n                 c1   1
    while(j>2){             c2   Log_2 n
        sum++;
        j=j/2;
    }
}
```

بدین ترتیب با توجه به تقسیم شدن j در شرط while در هر حلقه تا j در

از j در Logn بوده و بیانگر عامل تقسیم (۲) می‌باشد $T(n)=O(\log_2 n)$

(۲)

Subject
Date

```
۳) int F3 (int n){
    if(n<10){
        system.out.println("1");
        return n+3;
    }
    else{
        return F3(n-1) + F3(n-1)
```

با فرض n بزرگ تر از ۱۰ است.

تابع بازگشتی است و دو بار را اعمال را صدا می‌زند که این کار n-10 بار
تکرار می‌شود و با توجه به بازگشتی بودن از توان های عامل است (اینجا ۲) و در نتیجه ا در در آن برابر با $O(2^n)$

(۲)

۱) $T(n) = n^2 \log n + 2^n$

ما براساس میزان رشد اگر مقایسه کنیم. در اعداد بزرگ

$n^2 \log n < 2^n$

⇐ بدترین زمان اجرای آن $T(n) = 2^n$ است

۲) $T(n) = n^2 . (n + n\log n + 1000)$

با مقایسه کردن میزان رشد عوامل درون پرانتز، در اعداد بزرگ

$n < n\log n \Rightarrow T(n) = O(n^2 \log n)$

۳) $T(n) = n! + 2^{2n}$

در ابتدا 2^{2n} رشد سریع‌تری دارد ولی در اعداد بزرگتر که با توجه به اینکه در هر مرحله عنصر n برابری می‌شود ولی در n! ، n برابری شده رشد n! بزرگتر است ⇐ $T(n) = O(n!)$

۴) $T(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$

جمع آن ها

$\sum_{i=1}^{n} \frac{1}{i} = \int_1^n \frac{dx}{x} = \log i \rightarrow T(n) = O(\log n)$

۵) $T(n) = ((\frac{1}{2})(3n + 5 + n))^4$

$4n+5$

در رشد اعداد ثابت یک تاثیر اند اما با توجه به نادیده ...

$\frac{16 n^4}{24} \Rightarrow n^4$

$\Rightarrow T(n) = O(n^4)$

۴) $T(n) = T(n-2) + \log n$ در هر مرحله دو جمله شکل فراخوانده که در واقع با توجه به جایگذاری مرتب و اینکه n بار تکرار می‌شود و $\log n$ را دارا می‌شود

$O(n) = n \log_2^n$

۳

1) $f(n)=O(g(n)) \Rightarrow f(n) \le c*g(n)$ for all $(n>n_0)$
(c: عدد ثابت — n_0: عدد ثابت)

$f(n)=\Omega(g(n))$ $f(n) \ge c*g(n)$ for all $(n>n_0)$
(c: عدد ثابت — n_0: عدد ثابت)

اشتراک این دو $\Rightarrow f(n)=g(n)$

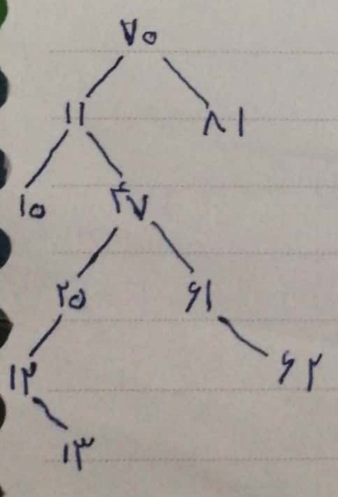
۲)

۳) بدترین حالت در binary heap هنگام (در max heap یا min heap) Delete یا کردن Root است که بزرگترین زمان را بر که نسبت به O(H)
می باشد که با توجه به اندازه n داده شده ارتفاع آن از O(Logn) می باشد چون باید مقایسه انجام شود در هر سطح‌ها دو Success هم در جای Root قرار گیرد و مرتب سازی شوند.

۴) بدترین حالت برای آن است که ورودی اعداد به ترتیب از کوچک به بزرگ یا از بزرگ به کوچک باشد
که بزرگترین تعداد مقایسه انجام می‌شود که می‌شود گفت تعداد مقایسه‌ها برابر $\sum_{i=1}^{n} i$ می باشد که از O(n)
است.

۴

۷۰ ۱۱ ۴۷ ۸۱ ۲۰ ۶۱ ۱۰ ۱۲ ۱۳ ۶۲



BST اعداد داده شده

1) Preorder: Root_Left_Right : ۷۰-۱۱-۱۰-۴۷-۲۰-۱۲-۱۳-۶۱-۶۲-۸۱

2) Postorder: Left_Right_Root : ۱۰-۱۳-۱۲-۲۰-۶۲-۶۱-۴۷-۱۱-۸۱-۷۰

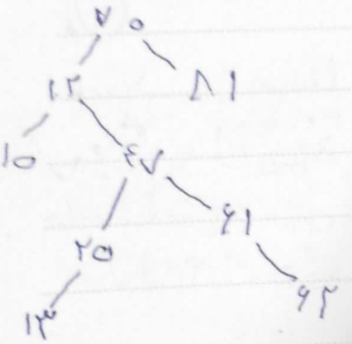
3) inorder: Left_Root_Right : ۱۰-۱۱-۱۲-۱۳-۲۰-۴۷-۶۱-۶۲-۷۰-۸۱

در پیمایش inorder برای این BST ما به حالت مرتب شده صعودی دست یافتیم
inorder به ما آرایه مرتب شده برمی‌گرداند.

Subject
Date

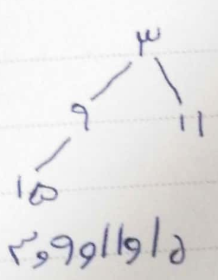
ادامه ۴

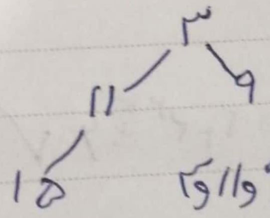
۴) جنبی successor 11 = 12 ⇐



⑤

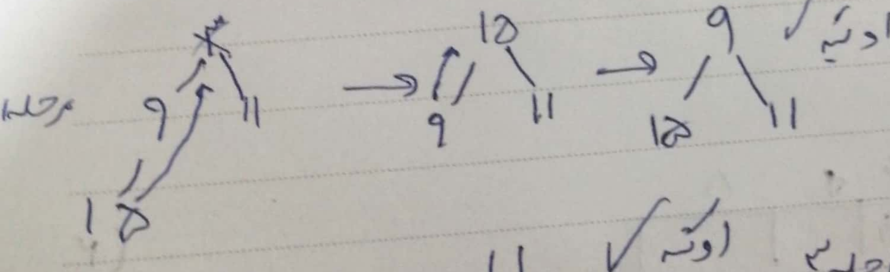
۱) اگر به ترتیب سوال باشد فقط دو حالت پیش اداریم.
در binary min heap ریشه از فرزند کوچکتره
و هر طبقه باید پر شه تا به طبقه دیگه بریم

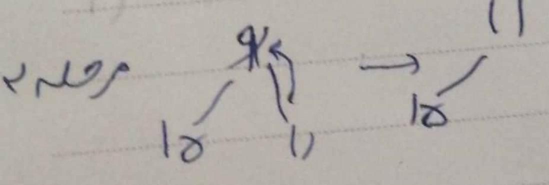


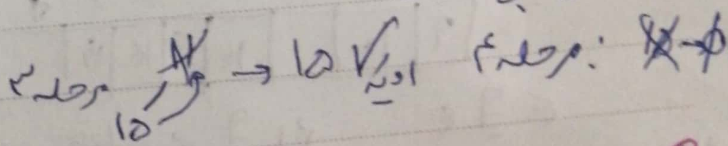


۲) خروجی عدد ۳
خروجی عدد ۹
خروجی عدد ۱۱
خروجی عدد ۱۵

در خروجی اعداد Sort
۱۵و۱۱و۹و۳







⑥

Subject

Date

۷

hash function mod ۱۳

با روش Linear probing عنصرها را بررسی کرده و جایگاهشان را یافتیم.

۱۵ $\overset{۱۳}{\equiv}$ ۲ → پره * ۲+۱ $\overset{۱۳}{\equiv}$ ۳ is full ۲+۲ $\overset{۱۳}{\equiv}$ ۴ is full ۲+۳ $\overset{۱۳}{\equiv}$ ۵ is full ۲+۴ $\overset{۱۳}{\equiv}$ ۶ ✓

خانه ۲ تا ۵ پره پس به ناچار وارد خانه index = ۶ می شود.

۳۹ $\overset{۱۳}{\equiv}$ ۰ ✓ ۵۳ $\overset{۱۳}{\equiv}$ ۱ ✓ ۶۷ $\overset{۱۳}{\equiv}$ ۲ ✓ ۸۱ $\overset{۱۳}{\equiv}$ ۳ ✓ ۹۵ $\overset{۱۳}{\equiv}$ ۴ ✓ ۱۸ $\overset{۱۳}{\equiv}$ ۵ ✓ ۳۳ $\overset{۱۳}{\equiv}$ ۷ ✓

۴۷ $\overset{۱۳}{\equiv}$ ۸ ✓ ۶۱ $\overset{۱۳}{\equiv}$ ۹ ✓ ۲۵ $\overset{۱۳}{\equiv}$ ۱۲ ✓

quadratic probing:

۱۵ $\overset{۱۳}{\equiv}$ ۲ is full ۲+۱۲ $\overset{۱۳}{\equiv}$ ۳ is full ⇒ ۲+۲۲ $\overset{۱۳}{\equiv}$ ۶ ✓

۸

۱) ۱) linear probing ۲) quadratic probing ۳) Double hashing

۲)

۰	۱	۲	۳	۴	۵	۶	۷	۸
۹	۱۸		۱۲	۴	۱۴	۳۱		

۴ $\overset{۹}{\equiv}$ ۴ ۲۱ $\overset{۹}{\equiv}$ ۳ —خطا→ ۳+۳ $\overset{۹}{\equiv}$ ۶

۹

۱) من از این نکته استفاده کردم که در BST عنصر تکراری وجود ندارد

این نکته باعث کمک در نوشتن بود. همچنین در شرط هایم از اینکه اشتراکات ۲ بار حساب شوند جلوگیری کردم.

```
int f(node x, node y)
  list<int> a = new list<int>();
  while(x != NULL || y != NULL)
    f=0;
    if(x != NULL){
      for(int i=0; i<a.Length; i++){
        if(a[i] == x.data)
          f=1
      }
      if(f==0){
        a.add(x.data)
      }
      if(x.parent != null)
        x = x.parent();
      else:
        x = NULL;
      ادامه بعد
    }
```

(۳)

الف ۹-۱)

```
f=0
if(y!=NULL){
  for(int i=0; i<a.Length; i++){
      if(a[i]==y.data)
         f=1
  }
  if(f==0){
    a.add(y.data)
  }
  y=y.parent();          <- if(y.Parent!=null)
 }else:
    y=NULL
}  -> while
int sum=0
for(int i=0, i<a.Length; i++)
{
   sum=Sum+a[i];
}
Return sum
```

۲) با توجه به این نکته که برای این روش باید عناصر موجود در برگ ها با بیشترین مقدار رو نسبت به پدرهای آنها دارند می باید آن عناصر بررسی شوند. همچنین با توجه به درخت دودویی کامل بودن، همه برگ ها در عمق آخر است. روش الگوریتمی من بدین گونه است که باید بیابیم برای هر کدام از برگ ها (x,y) A نظیر آنها را بیابیم. سپس با جویک متغیر باشد که بیشترین عدد یافته شده را

۱۳۹۹

ذخیره شده های آنها ببین رسیده اگر f(x,y) بیشترین شده خود آن دو را هم در دو متغیر ذخیره کرده و در نهایت برمی گردانیم. که در مرحله انتخاب ما به تعداد زیر حالت داریم:

$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2}$ حالت $= O(n^2)$

اما در حالت یافتن و جمع f(x,y) طبق بخش 1 اگر در O(nlogn) داریم که در نتیجه.

با توجه به اینکه این ها تو در تو اند و شرط وابسته اند اگر زمان می شود: $O(n^2 \log n)$

که اگر در بیا ربدی می باشد و می شود با جست و جو فقط بیشتر به اگر در دیگری رسید