علی اسد ۹۸۳۱۰۰۴

۱) در ۱۶ و ۳۳ تناقض رخ نمی‌دهد اما در insert ۲۱، Case۲ ایجاد می‌شود که با Right rotate

روی ۳۳ به Case ۳ تبدیل می‌شود. برای اصلاح Case ۳، ۱۶ را Red و ۲۱ را Black می‌کنیم

و روی ۱۶ left-rotate انجام می‌دهیم. ۱۷ را وارد می‌کنیم. Case ۱ رخ می‌دهد و ۱۶ و ۳۳

را Black و ۲۱ را Red می‌کنیم حال Root یعنی ۲۱ را دوباره Black می‌کنیم.

در ۱۳ و ۵۲ تناقض رخ نمی‌دهد. در ۱۲ Case ۱ رخ می‌دهد برای اصلاح ۱۳ و ۱۷ را Black

و ۱۶ را Red می‌کنیم. در ۱۹ تناقض رخ نمی‌دهد. در ۲۶، Case ۲ رخ می‌دهد که با

left rotate روی ۲۵ به Case ۳ تبدیل می‌شود. و ۲۶ را Black و ۳۳ را Red کرده و روی

۳۳، Right rotate انجام می‌دهیم. ۴۰ و ۲۲ بدون تناقض هستند.

برای ۲۴، Case ۲ رخ می‌دهد. روی ۲۲، left rotate انجام می‌دهیم.

۲۴ را Black و ۲۵ را Red می‌کنیم و روی ۲۵، Right rotate انجام می‌دهیم.

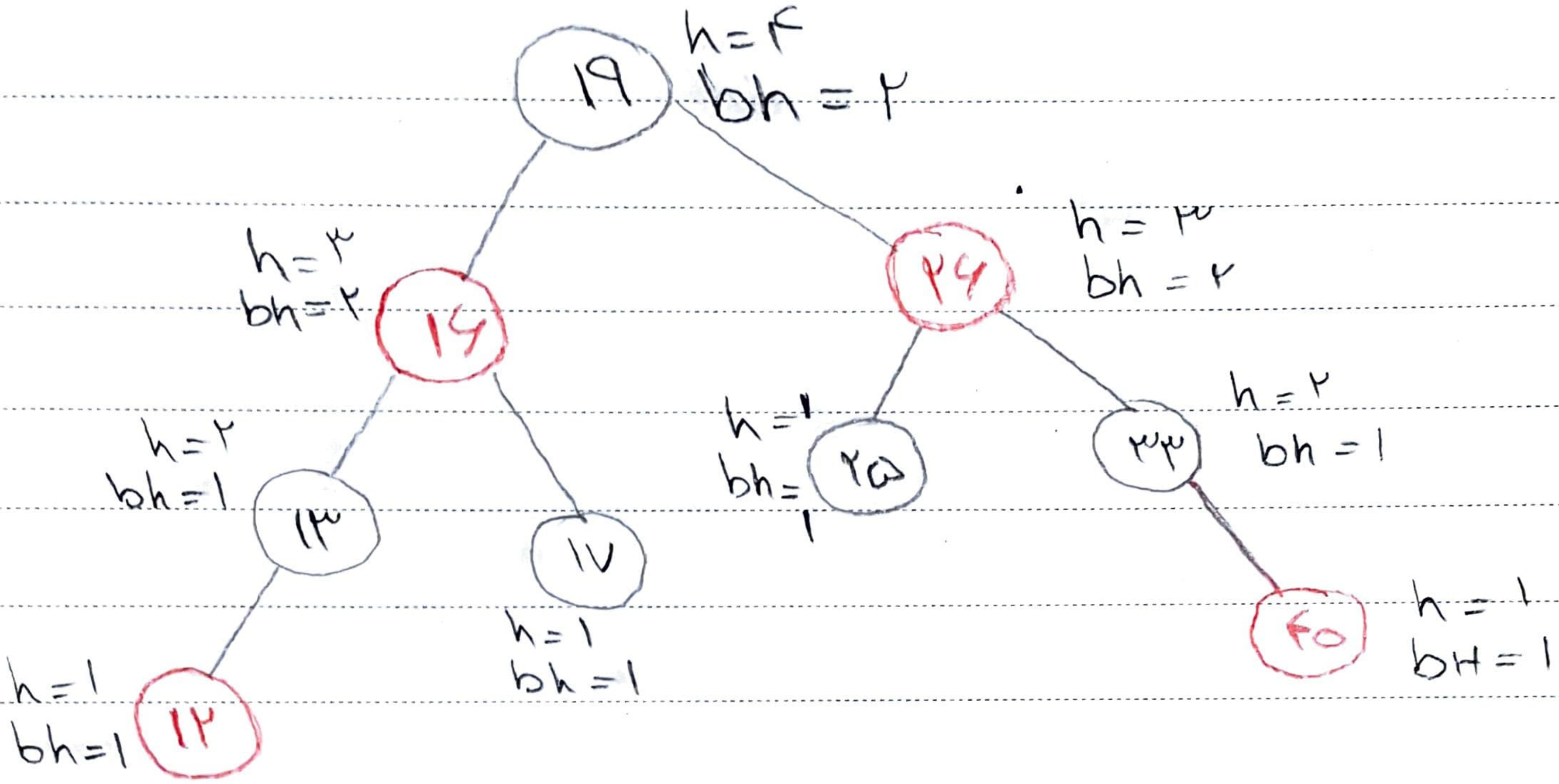
ب) برای حذف ۲۱، Predecessor آن یعنی ۱۹ را حذف می‌کنیم و key آن

را به جای ۲۱ می‌گذاریم چون ۱۹ Red است، اصلاح نیاز نیست. برای حذف ۲۲ چون

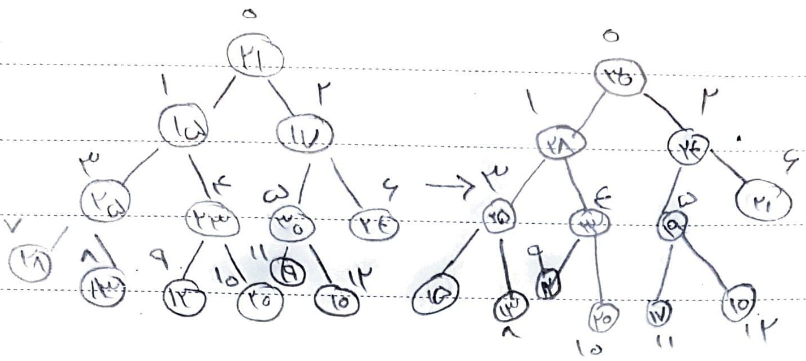
فرزندی ندارد و Red است اصلاح نمی‌خواهد. برای حذف ۲۴، تنها فرزندان را به جای آن می‌گذاریم
یعنی ۲۵

و رنگ آن را Black می‌کنیم.

ج)



h = ۴
bh = ۲
۱۹
h = ۳
bh = ۲
۱۶
h = ۳
bh = ۲
۲۶
h = ۲
bh = ۱
۱۳
۱۷
h = ۱
bh = ۱
h = ۱
bh = ۱
۲۵
h = ۲
bh = ۱
۳۳
h = ۱
bh = ۱
۱۲
h = ۱
bh = ۱
۴۰
h = ۱
bH = ۱



۲)

آرایه maxHeap نیست

چون arr[1] < arr[۳]

Build maxHeap : از عنصر ۶ = $\frac{n}{۲}$ ام یعنی ۲۴ شروع می‌کنیم، چون فرزندان آن nil هستند پس

maxHeapify برقرار است. عنصر ۵ ام هم با max-Heapify تغییراتی را مشاهده نمی‌کند.

در maxHeapify روی عنصر ۳ ام جای عنصر ۳ ام و ۷ ام عوض می‌شود و در ادامه چون فرزندان عنصر جدید ۷ ام

(یعنی ۲۵) nil هستند maxHeapify تمام می‌شود.

حال روی عنصر ۲ ام maxHeapify می‌زنیم در نتیجه جای آن با عنصر ۵ ام عوض می‌شود و دوباره روی عنصر جدید

خانه ۵ ام (یعنی ۱۷) max-Heapify می‌زنیم که نهایتاً جای آن با عنصر ۱۱ ام عوض می‌شود.

حال روی عنصر ۱ ام maxHeapify می‌کنیم که جای آن با ۲۸ عوض شده و سپس روی شاخه سمت چپ

maxHeapify می‌کنیم که جای عنصر ۳ ام (۱۵) و عنصر ۷ ام عوض می‌شود

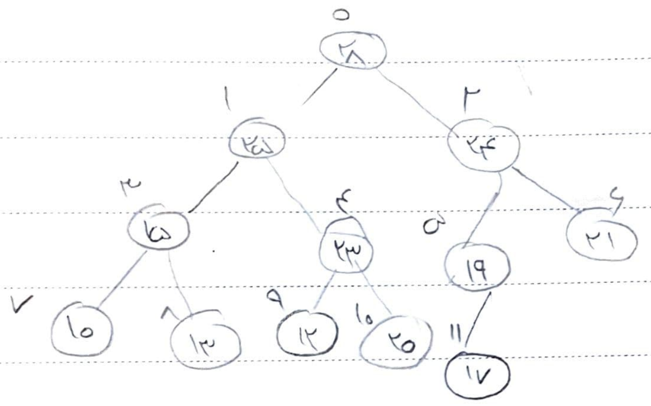
حال روی عنصر ۰ ام maxHeapify می‌زنیم که جای آن با عنصر ۲ ام عوض شده و روی شاخه سمت راست عنصر ۲ ام

maxHeapify می‌زنیم که جای آن با عنصر ۶ ام عوض می‌شود. آرایه maxHeap شد

ب) ۱. جای عنصر ۵ و ۱۲ را عوض کرده و سایز آن را واحد کاهش می‌دهیم تا عدد ۳۰

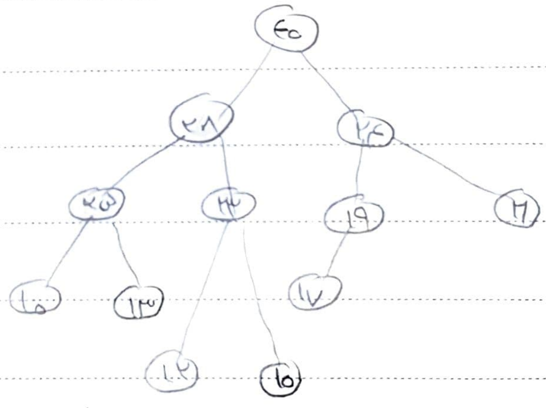
دیده نشود. سپس روی عنصر ۵ جدید (یعنی ۱۰) maxHeapify می‌زنیم که منجر می‌شود جای

۲۸ و ۱۰ ، ۲۵ و ۱۰ ، ۱۵ و ۱۰ عوض شود



۲. به جای ۱۵ ، ۴۵ می‌گذاریم و جای آن را تا آنجا که ممکن است با پدرش عوض می‌کنیم:

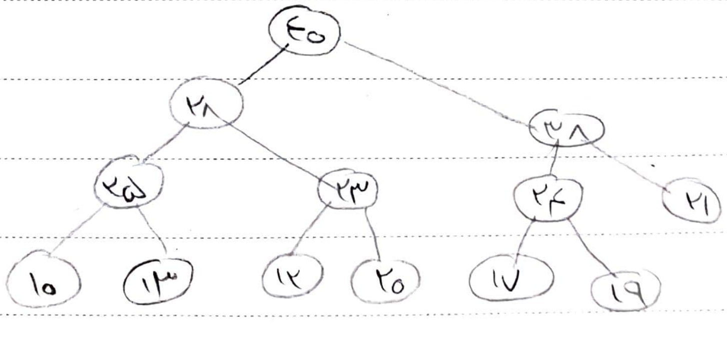
(۴۵ و ۲۵) ⟵ (۴۵ و ۲۸)



۳. ابتدا ۳۸ را به شاخه راست ۱۹ وصل کرده و maxHeapify می‌کنیم:

Swap:

(۳۸ و ۱۹) ⟵ (۳۸ و ۲۴)



Subject
Date

۳) برای بدست آوردن Count(x) به هر node یک ویژگی با اسم size اضافه می‌کنیم

که می‌شود اندازه زیر درخت آن node. حال تابع Count به صورت زیر تعریف می‌شود:

```
Count (T , x ):                          // size(x) = size(x.left)
    Counter = 0                           // + size(x.right) +1
    y = T.root
    while (y != x)
        if (x.key > y.key)
            Counter += y.left.size +1
            y = y.right
        else if (x.key < y.key)
            y = y.left
        else
            Counter += y.left.size
            y = y.right
    return Counter
```

تعداد اجرای حلقه while، متناسب با ارتفاع درخت می‌شود پس

$Cost(Counter(x)) = O(h(n)) = O(\log n)$

برای insertion و deletion نیز بعد از انجام مراحل، از root شروع کرده و size پدران

x را به ترتیب یک واحد زیاد و کم می‌کنیم. چون این traverse از مرتبه O(logn) انجام می‌شود cost آنها فرقی نمی‌کند.

برای find صرفا بجای return کردن true و false در search خود x را باز می‌گردانیم و متوجه تغییر خاصی نمی‌شویم.

PAPCO