پاسخ تمرین شماره ۴

دانشکده مهندسی برق و کامپیوتر

ساختمان داده – بهار ۱۳۹۹

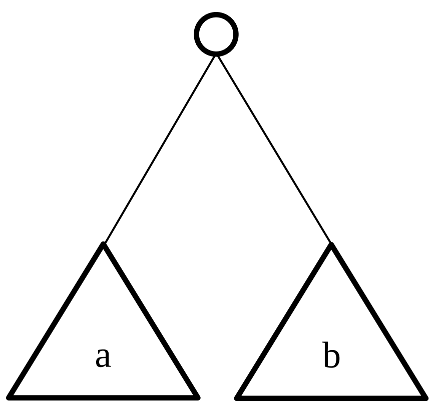
استاد : دکتر فقیه

مسئول تمرین : حمید تراشیون

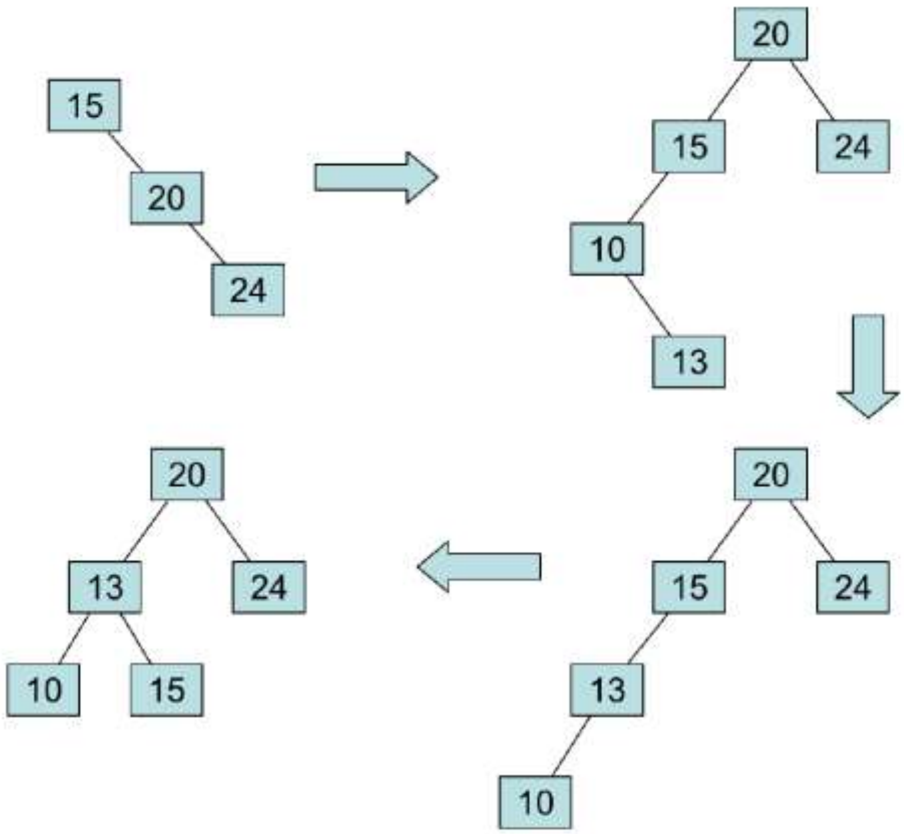
htarashion@gmail.com

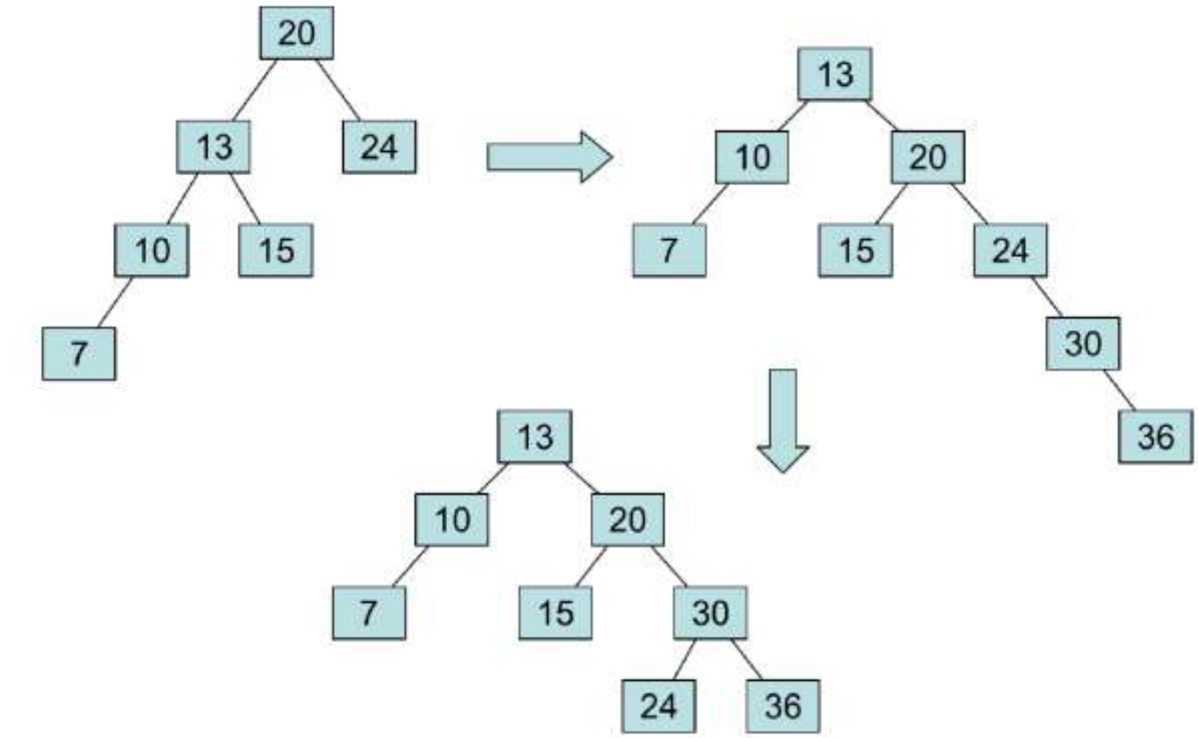
---

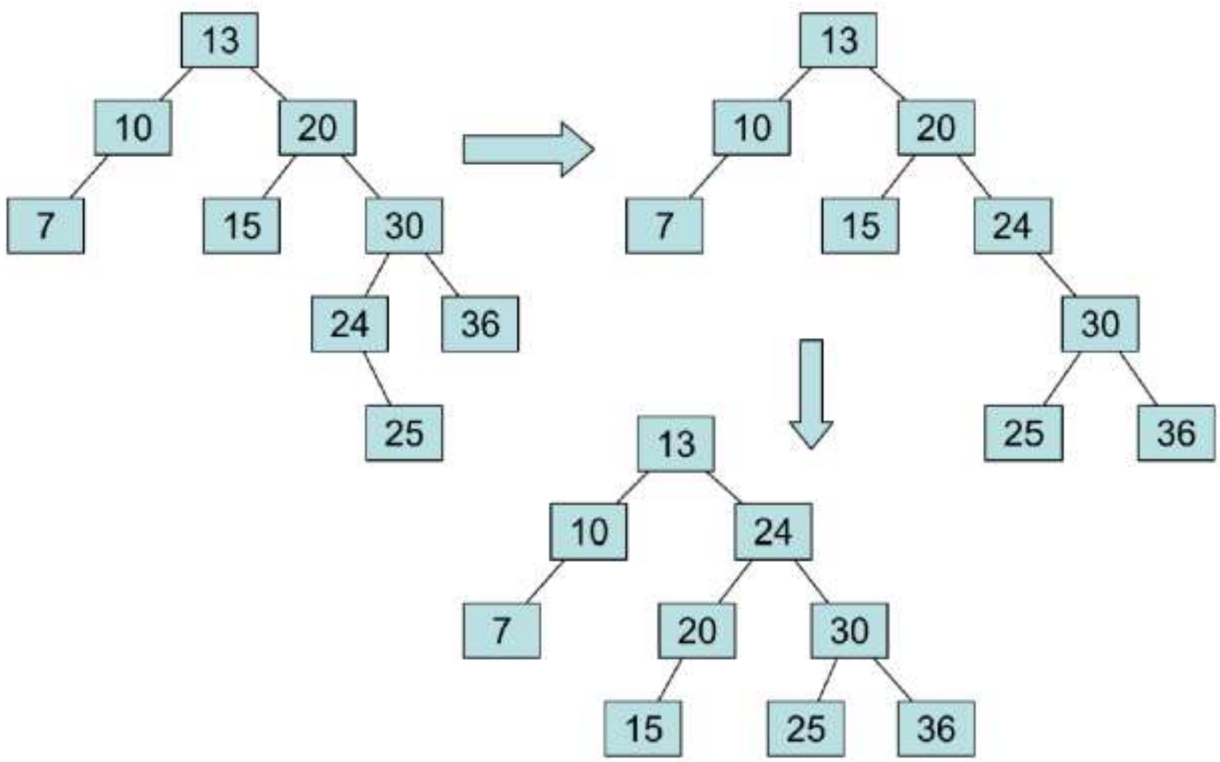
۱. این سوال را به روش بازگشتی حل می کنیم. چنانچه زیردرختی با تعداد رئوس بین $\frac{n}{3}$ و $2\frac{n}{3}$ بیابیم سوال حل شده است. برای پیدا کردن این زیردرخت از راس درخت شروع می کنیم و 2 زیر درخت موجود را بررسی می کنیم، چنانچه یکی از دو زیردرخت در شرط سوال صدق کنند، زیردرخت مطلوب را یافته ایم، در غیر این صورت یکی از 2 زیر درخت کمتر از $\frac{n}{3}$ راس و آن یکی بیشتر از $2\frac{n}{3}$ راس دارد. می دانیم که زیر درخت مدنظر در زیردرختی که کمتر از $\frac{n}{3}$ راس دارد یافت نمی شود، پس به سراغ زیردرختی می رویم که بیشتر از $2\frac{n}{3}$ راس دارد و در آن مشابه کاری که با راس اصلی درخت و 2 زیردرختش انجام دادیم انجام می دهیم. در این صورت دوباره یا یکی از 2 زیردرخت در شرط سوال صدق می کند یا این که یکی کمتر از $\frac{n}{3}$ و دیگری بیشتر از $2\frac{n}{3}$ راس دارد و این به آن علت است که باید جمع راس های 2 زیر درخت بیشتر از $2\frac{n}{3}$ باشد. به همین صورت اگر پیش برویم، بدترین حالتی که ممکن است پیش بیاید آن است که هر دفعه زیردرخت مدنظر را نیابیم و مجبور به مراجعه به زیردرخت جدید با تعداد بیشتر از $2\frac{n}{3}$ می شویم. نکته قابل توجه آن است که هر دفعه با پیش رفتن در زیردرخت ها حداقل 1 راس از تعداد زیردرخت بزرگ تر کم می شود، پس عددی که بیشتر از $2\frac{n}{3}$ است به مرور کم و کمتر می شود تا در شرط مساله صدق کند.



.٢




20
13
24
10
15
7
13
10
20
7
15
24
30
36
13
10
20
7
15
30
24
36


13
10
20
7
15
30
24
36
25
13
10
20
7
15
24
30
25
36
13
10
24
7
20
30
15
25
36

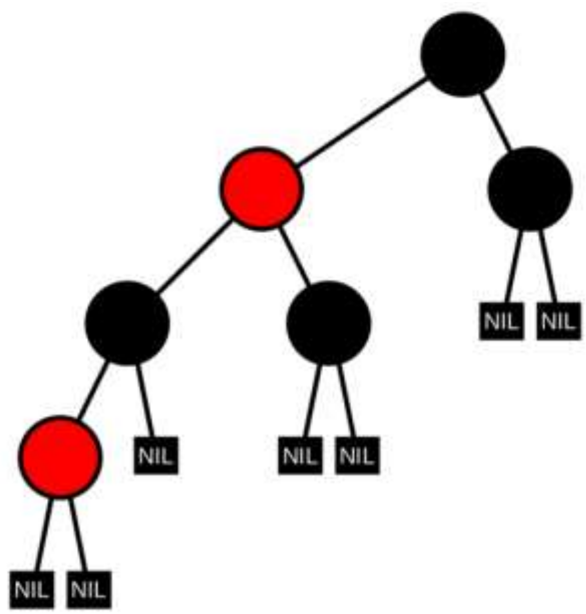
.۳

به صورت بازگشتی عمل می کنیم و از برگ ها شروع به پیمایش می کنیم تا به ریشه برسیم. ابتدا در برگ ها وزن خود آنها را نگه می داریم و هر بار در هر گره، سنگینترین زیر درخت دودویی تا این گره (در گره‌های زیرین و خود گره)، و سنگین ترین زیر درخت دودویی ای که این گره ریشه آن است را نگه میداریم. این کار را انقدر انجام می دهیم تا به ریشه برسیم. سنگین ترین زیر درخت ریشه جواب نهایی خواهد بود.
چون هر گره را با این فرایند یکبار طی می کنیم و محاسبه سنگین ترین زیر درخت‌ها در هر مرحله به بچه‌ی چپ و راست و خود گره بستگی دارد که$O(1)$ زمان می برد، پیچیدگی $O(n)$ دارد.

.۴

بله. کافی است درختی را در نظر بگیریم که زیر درخت سمت راست آن یک درخت دودویی کامل با عمق d باشد و تمام گره های آن نیز سیاه باشند. زیر درخت سمت چپ آن نیز یک درخت دودویی کامل با عمق $2d$ باشد با این شروط که گره های با عمق فرد قرمز و گره های با عمق زوج سیاه باشند.

مثال:

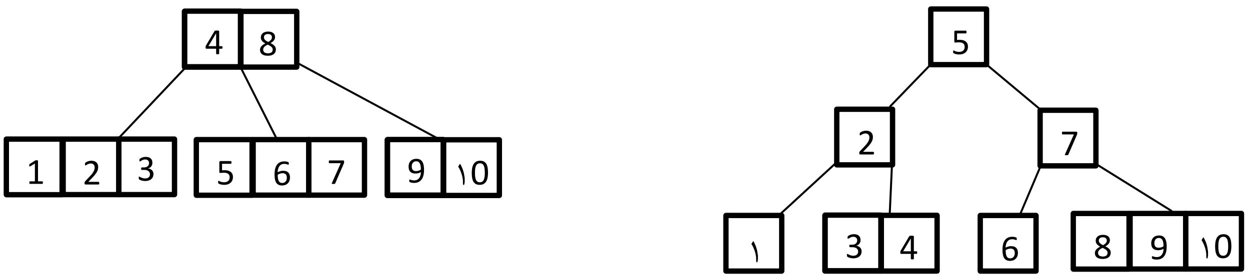


۵.

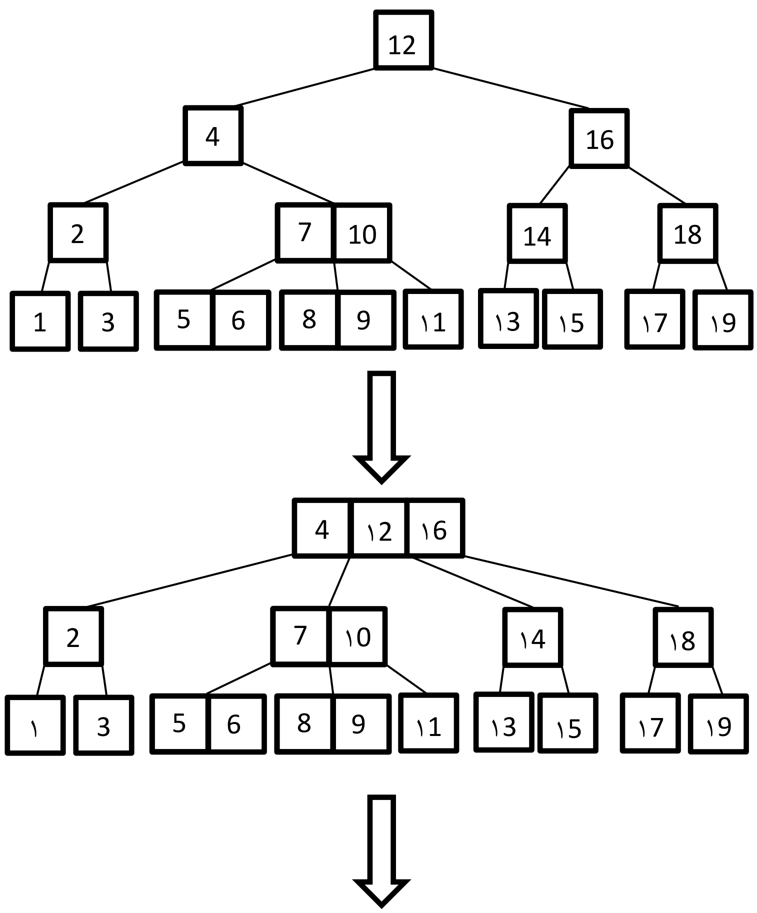
الف)

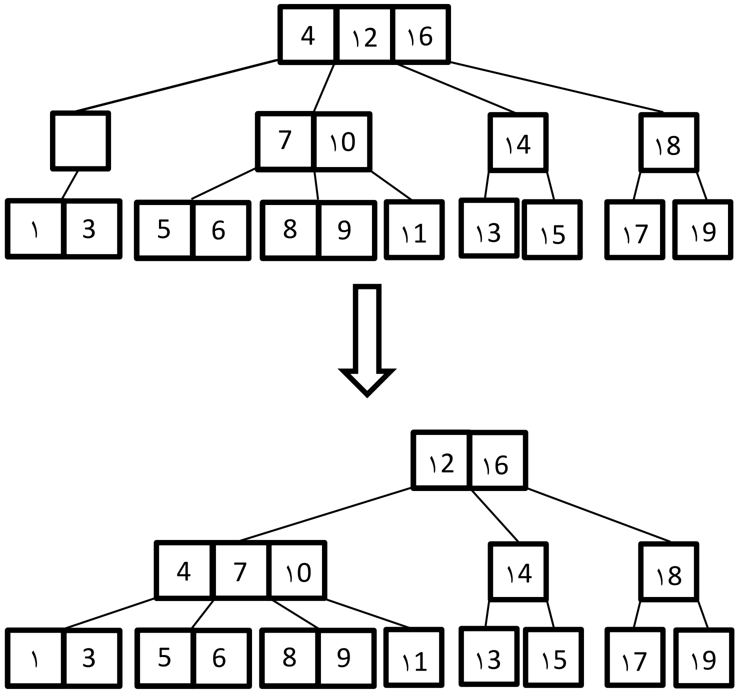
ترتیب وارد شدن برای max شدن نود ها(از چپ به راست): 7-3-5-9-1-6-10-2-4-8

ترتیب وارد شدن برای min شدن نود ها(از چپ به راست): 8-4-2-10-6-1-9-5-3-7



ب)





۶.

```
def b_tree_search(x, k):
    node = binary_search(x, k);
    if (node != null):
        return node
    else:
        if ((x.right == null) and (x.left == null)):
            return null
        else:
            input(next[x])
            return b_tree_search(next[x], k)


def binary_search(root, key):
    if root is None:
        return null
    if root.val == key:
        return root
    if root.val < key:
        return binary_search(root.right, key)
    return binary_search(root.left, key)
```

.۷

$$(2t-1)\left[(2t)^0 + (2t)^1 + (2t)^2 + \cdots + (2t)^h\right]$$
$$= (2t-1)\sum_{i=0}^{h}(2t)^i = (2t-1)\,\frac{(2t)^{h+1}-1}{2t-1}$$
$$= (2t)^{h+1} - 1$$

.۸

فرض می کنیم راس x دو فرزند داشته باشد، پس successor آن، کوچک ترین عنصر زیردرخت سمت راست x می باشد که برای بدست آوردن آن از x به زیر درخت راست می رویم و بعد از آن به چپ می رویم تا جایی که به null برسیم. پس تا جایی که فرزند چپ وجود داشته باشد باید به چپ برویم تا به successor برسیم. بنابراین successor نمی تواند فرزند چپ داشته باشد چون که اگر می داشت دیگر خود successor نبود.

.۹