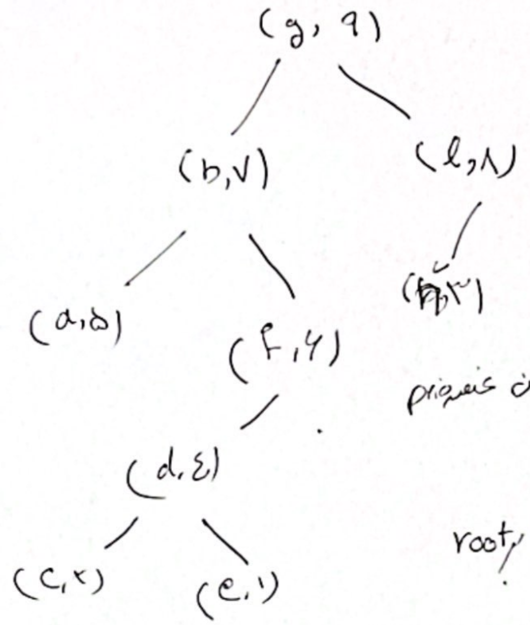
۱) الف)



ب) Q ها را به ترتیب بزرگترین عنصر prio
می‌گذاریم

۲) root را برابر (key[L], prio[L]) قرار

۳) الگوریتم را به ازای key و prio[n تا l+1] [n تا l+1] root = root.right اجرا در صورتی که (l+1 تا n) خالی شد

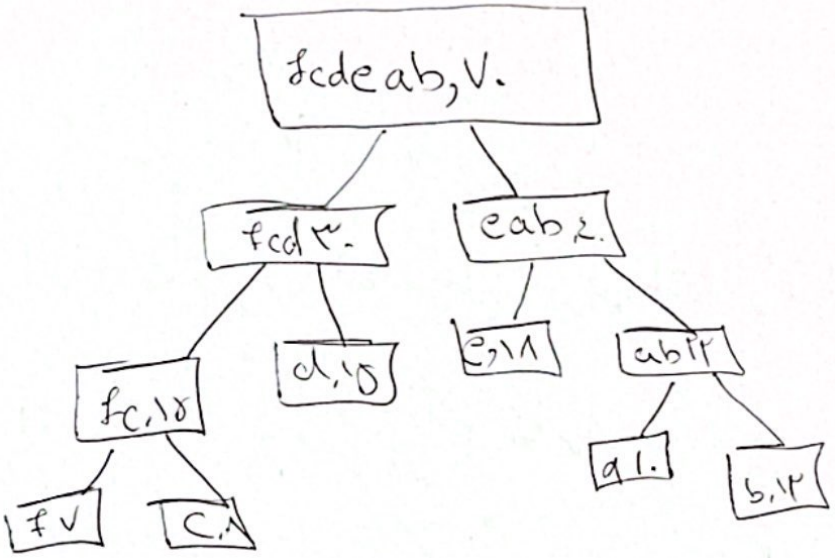
۴) الگوریتم را به ازای key[l-1 تا ۰] , prio[l-1 تا ۰] root = root.left اجرا می‌کنیم
در صورتی که (l-1 تا ۰) خالی شد

۵)

```
void q2(Node head){
    while(true){
        while(head.left != null) head = head.left;
        print (head.data);
        if (head.right != null) head = head.right;
        else while(true){
            if(head.parent.left == head)) head = head.parent
            else{
                while(true){
                    if (head.parent == null) return;
                    if (head.parent(left == head) break;
                    head = heat.parent;
                }
                head = head.parent;}
```

```
print(head.data);
if(head.right != null){
    head = head.right;
    break;
}
}
}
```

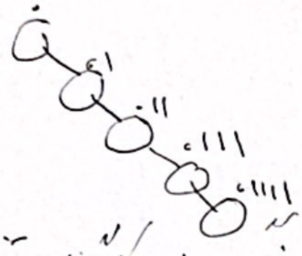
(۳)



۲(۷+۸+۱۰+۱۲)+۲(۱۵+۱۱) = ۱۷۷

(۴)

۵) شکل درخت به این صورت است

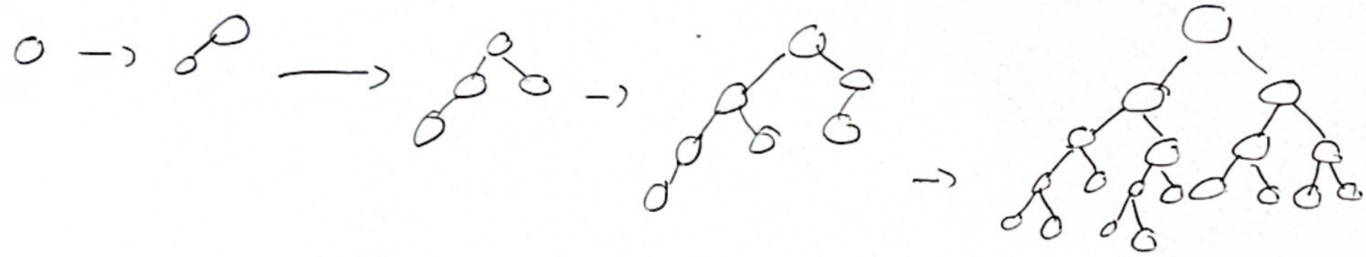


اگر انواع کاراکترها زیاد باشد طول کدها خیلی بزرگ می‌شود در واقع این کد فقط برای کدگذاری تنها بزرگی و کوچکی تعداد کاراکترها را در نظر می‌گیرد ولی در عین حال به میزان تفاوت کاراکترها نیز توجه می‌شود.

۶)

۰	۱	۲	۳	۴	۵
۱	۲	۴	۷	۱۲	۲۰

ارتفاع ۴ ⇐ ۲۰ > ۱۹ > ۱۲

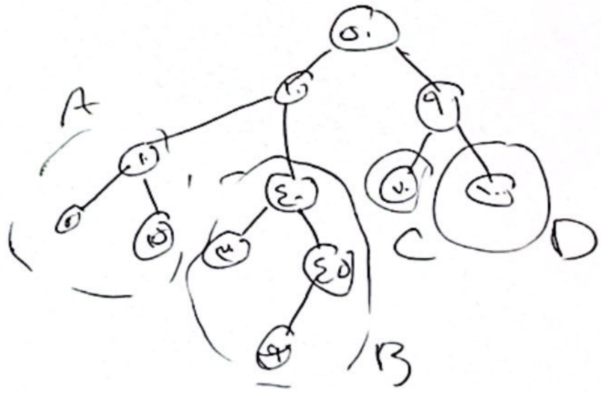


۱۲۹۷≤۱۹

۷) ۱) بله ۲) خیر

در کل برای کاهش ارتفاع باید ریشه را دوران راست یا چپ بدهیم اگر عمیق‌ترین ریشه (اینجا ۴۲) در گروه

A) راست حول ریشه
D) چپ حول ریشه
B) دوران نوع ۳
C) دوران نوع ۴



۹) O(k) ، چرا بیشتر نه زیرا پیمایش میانه‌ترتیب با O(n) انجام می‌شود و باید تا k امین عنصر ادامه دهیم و چرا کمتر نه زیرا اگر فرض کنیم ارتفاع درخت h-1 باشد حداقل کار پیمایش مسیر ریشه تا عنصر مورد نظر است.

۱۰) حلقه با $i = 0$ تا $i = 2^{\log_2(n+2)} - 2$

(۱) اگر $a[i] < a[2i+1]$ return False

(۲) اگر $a[i] < a[2i+2]$ return False

(۳) return true;

۱۲) برای تشخیص هیپ بودن حداکثر $n-1$ مقایسه نیاز داریم