پاسخ سوالات تمرین سوم

۱- یک الگوریتم غیربازگشتی برای پیمایش postorder درخت باینری ارائه دهید.(شبه کد و تحلیل زمانی الزامیست)

پیمایش postorder با دو تا استک

پیمایش postorder با یک استک

۲- الگوریتمی ارائه دهید که در مرتبه زمانی $O(n)$بررسی کند که آیا یک درخت دودویی، درخت دودویی جستجو هست یا خیر.

راه اول: طبق الگوریتم زیر پیش می‌رویم که در آن با شروع از ریشه، در هر مرحله مقدار آن راس با دو فرزند آن مقایسه شده و در صورتی که مطابق تعریف درخت BST باشد به سراغ راس دیگر می‌رود.

زمان اجرای برنامه O(n) خواهد بود.

```
int isBST(struct node* node)
{
  if (node == NULL)
    return 1;


  if (node->left != NULL && node->left->data > node->data)
    return 0;


  if (node->right != NULL && node->right->data < node->data)
    return 0;

  if (!isBST(node->left) || !isBST(node->right))
      return 0;

  return 1;
}
```

راه دوم: ابتدا یک پیمایش inorder از درخت انجام می دهیم(O(n)) و مقادیر درخت را در یک آرایه می‌ریزیم.سپس بررسی می‌کنیم که آیا آرایه مرتب شده است یا خیر که این مرحله هم در O(n) انجام می‌شود.

۳- فرض کنید دو درخت دودویی جستجوی متوازن T_1 و T_2 به ترتیب دارای n_1 و n_2 رأس، مجموعه‌های S_1 و S_2 را نمایش می‌دهند.

الف) الگوریتمی ارائه دهید که در مرتبه ی زمانی $O(n_1 log(n_2))$و با استفاده از حافظه‌ی اضافی $O(1)$ مشخص کند که آیا $S_1 \subseteq S_2$ هست یا خیر.

در درخت جستجو، عمل search از اردر $O(log(n))$ است. اگر روی اعضای S1 پیمایش کنیم $O(n_1)$ و برای هر عضو بررسی کنیم که آیا در S2 هست یا نه، پیچیدگی زمانی $O(n_1 log(n_2))$ می‌شود. شبه کد:

```
isSubset(T1,T2){
      for (i:1→n){//O(n1)
            x = find(T1[i],T2)//O(log(n2))
            if(!x){
                  return false;
                  }
      }
      return true;
}
```

ب) الگوریتمی ارائه دهید که در مرتبه ی زمانی $O(n_1 + n_2)$و با استفاده از حافظه‌ی اضافی $O(n_1 + n_2)$ مشخص کند که آیا $S_1 \subseteq S_2$ هست یا خیر.

برای هر دو درخت پیمایش inorder انجام می دهیم.($O(n_1 + n_2)$)

سپس این دو آرایه را با یکدیگر مقایسه می‌کنیم

```
isSubset(T1,T2){
      pointer=0;
      for (i:1→n2){
            if(T1[pointer]==T2[i]){
                  if(pointer==T1.length){
                  return true;
                  }
                  Pointer++;
            }
      }
      Return false;
}
```

۴- فرض کنید یک درخت Max Heap داریم. برای هر یک از کارهای زیر شبه کدی ارائه کنید که آن کار را انجام دهد.

الف) حذف یک گره مشخص شده از درخت

```
int heap_delete(int pos) {
        if(!heap_sz && pos >= heap_sz)
                return -1;
        int val = heap[pos];
        //swap the value 1
        swap(heap[heap_sz-1], heap[pos]);
        heap[heap_sz-1] = -1;
        // heapify down 1
        heapify_down(pos);
        //heapify up 1
        heapify_up(pos);
        heap_sz--;
        return val;
}
```

ب) اضافه کردن یک عنصر به درخت

```
void heap_insert(int val) {
        //update value 1
        heap[heap_sz] = val;
        // heapify up 2
        heapify_up(heap_sz);
        heap_sz++;
}
```

ج) تغییر مقدار کلید یک گره مشخص از درخت

```
void heap_update(int pos, int new_val) {
        if(!heap_sz && pos >= heap_sz)
                return;
        // update value 1
        heap[pos] = new_val;
        // heapify down 1.5
        heapify_down(pos);
        // heapify up 1.5
        heapify_up(pos);
}
```

```
void heapify_up(int pos) {

        while(pos > 1 && heap[pos] > heap[pos/2]) {
                swap(heap[pos], heap[pos/2]);
                pos = pos/2;
        }
}
```