به نام خدا

ساختمان داده ها

جلسه هجدهم

دانشگاه بوعلی سینا

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

مطالب

- مقدمه و تعاریف
- نمایش درختان
- درختان دودویی
  - ADT درختان دودویی
  - خواص درختان دودویی
  - نمایش درختان دودویی
  - نمایش درختان با لیست ها
  - پیمایش درختان دودویی
  - اعمال روی درختان دودویی
  - درختان دودویی نخ کشی شده
- هرمها (Heaps)
- درختان جستجو
- درختان انتخاب
- جنگلها

هرمها

Heaps

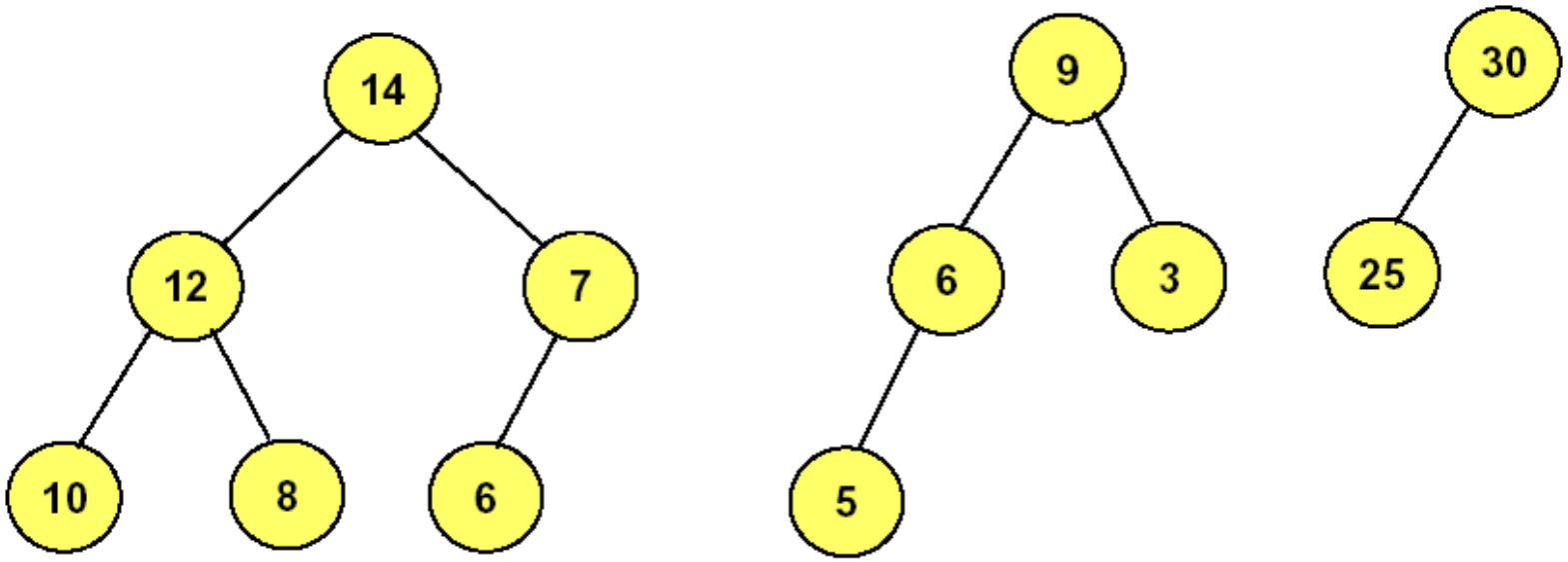
هرم‌ها

- کاربرد هرم ها بیشتر با صف اولویت است.
- در صف اولویت عنصری که حذف می شود باید دارای بالاترین اولویت باشد. در صف اولویت در هر زمان می توانیم هر عنصری با هر اولویتی اضافه کنیم.
- این ساختمان داده را Max PQ یا Min PQ می گوییم.

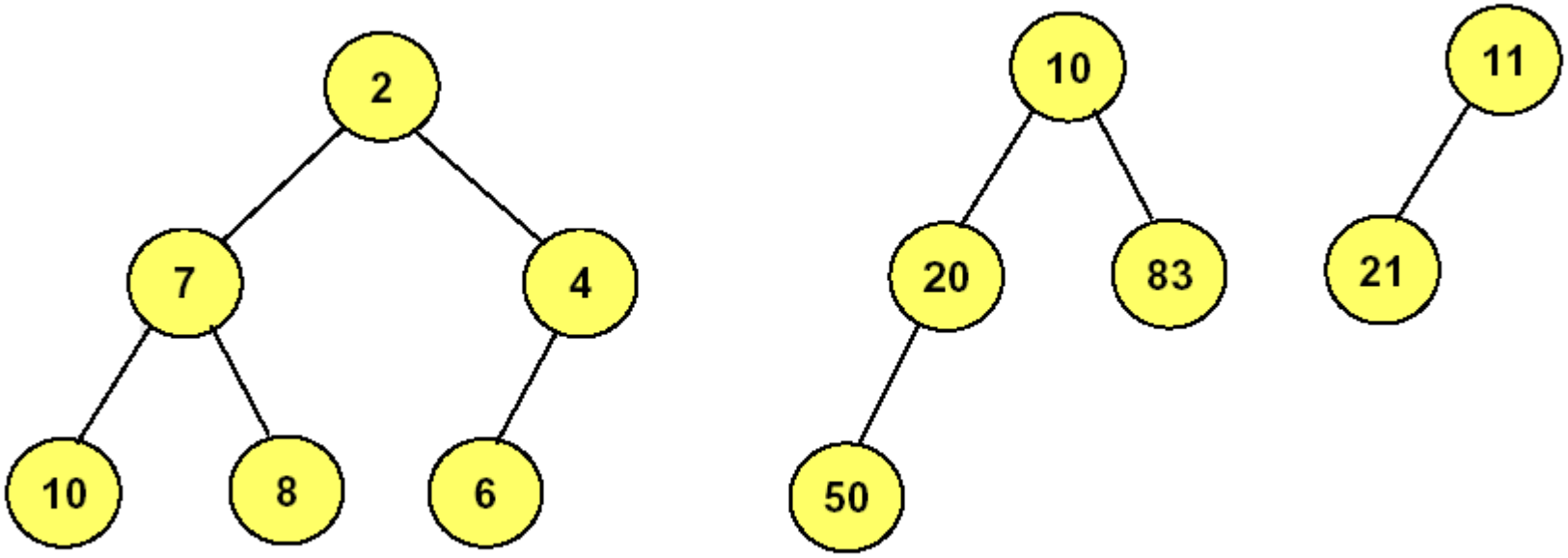
- برای پیاده سازی یک Max PQ می توان از Max Heap استفاده کرد.

Max Heap

- یک درخت حد اکثر درختی است که مقدار کلید هر گره کمتر از مقدار کلید فرزندانش نباشد. هرم حد اکثر یک درخت دودویی کامل است که یک درخت حد اکثر باشد.



Min Heap



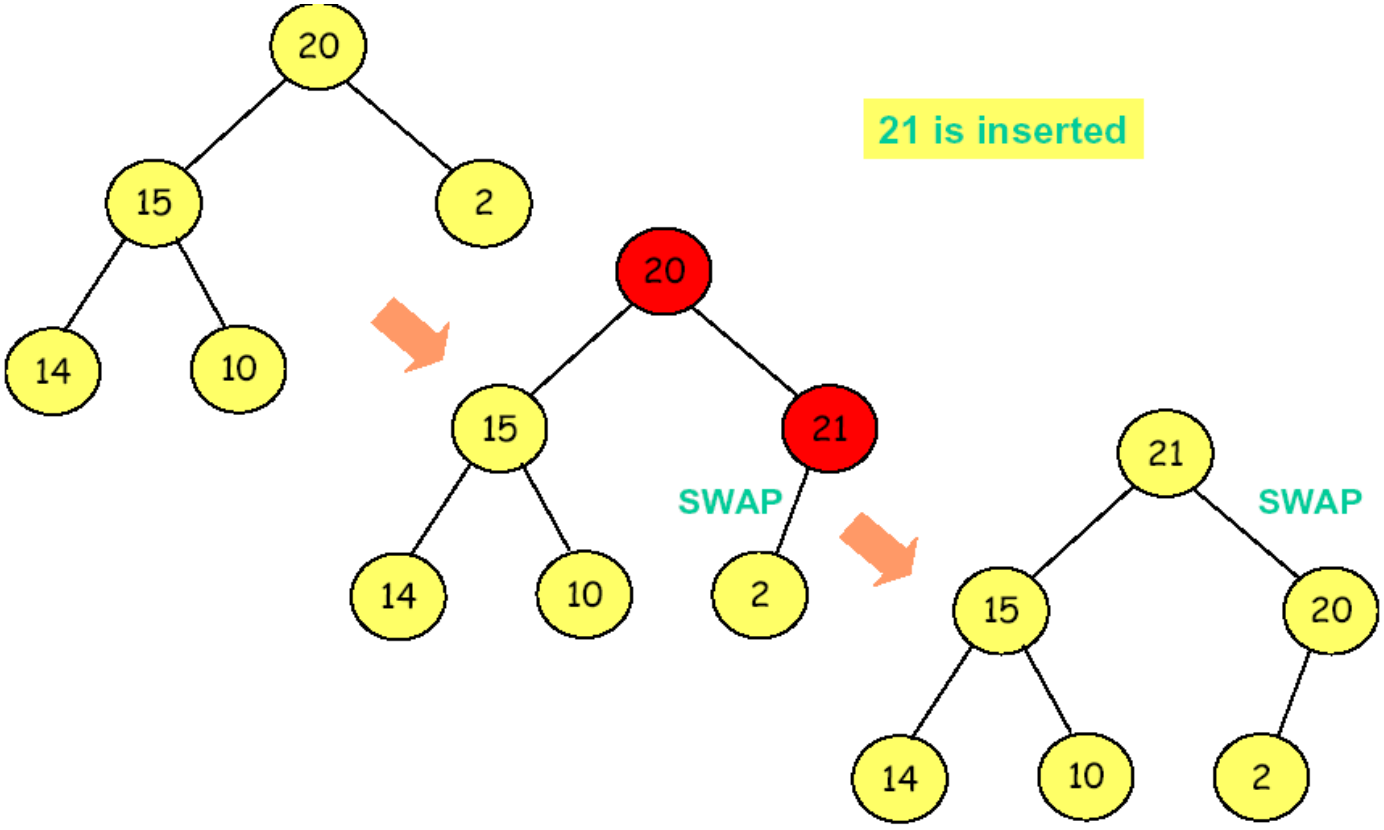
کلاس هرم

از آنجاییکه که یک هرم حداکثر یا حداقل یک درخت کامل است بهترین روش برای پیاده سازی آن استفاده از روش آرایه ای است.

```
template <class T>
class MaxHeap : public MaxPQ<T> {
  Element<T> *heap;
  int n;  // current size
  int MaxSize; // max heap size
public:
  MaxHeap(int sz = DefaultSize);
  bool IsEmpty();
  bool IsFull();
  void Insert(const Element<T>& x);
  Element<T>* DeleteMax(Element<T>& x);
};

template <class T>
MaxHeap<T>::MaxHeap(int sz)
  :MaxSize(sz), n(0){
  heap = new Element<T>[MaxSize + 1];
}
```

درج در یک هرم

- یک گره به اولین جای خالی در هرم اضافه می شود.
- مقدار گره با پدر مقایسه می شود. و اگر بزرگتر بود با گره پدر عوض می شود.
- تا رسیدن به گره ای که دیگر از گره پدر بزرگتر نباشد یا گره ریشه این کار ادامه پیدا می کند.



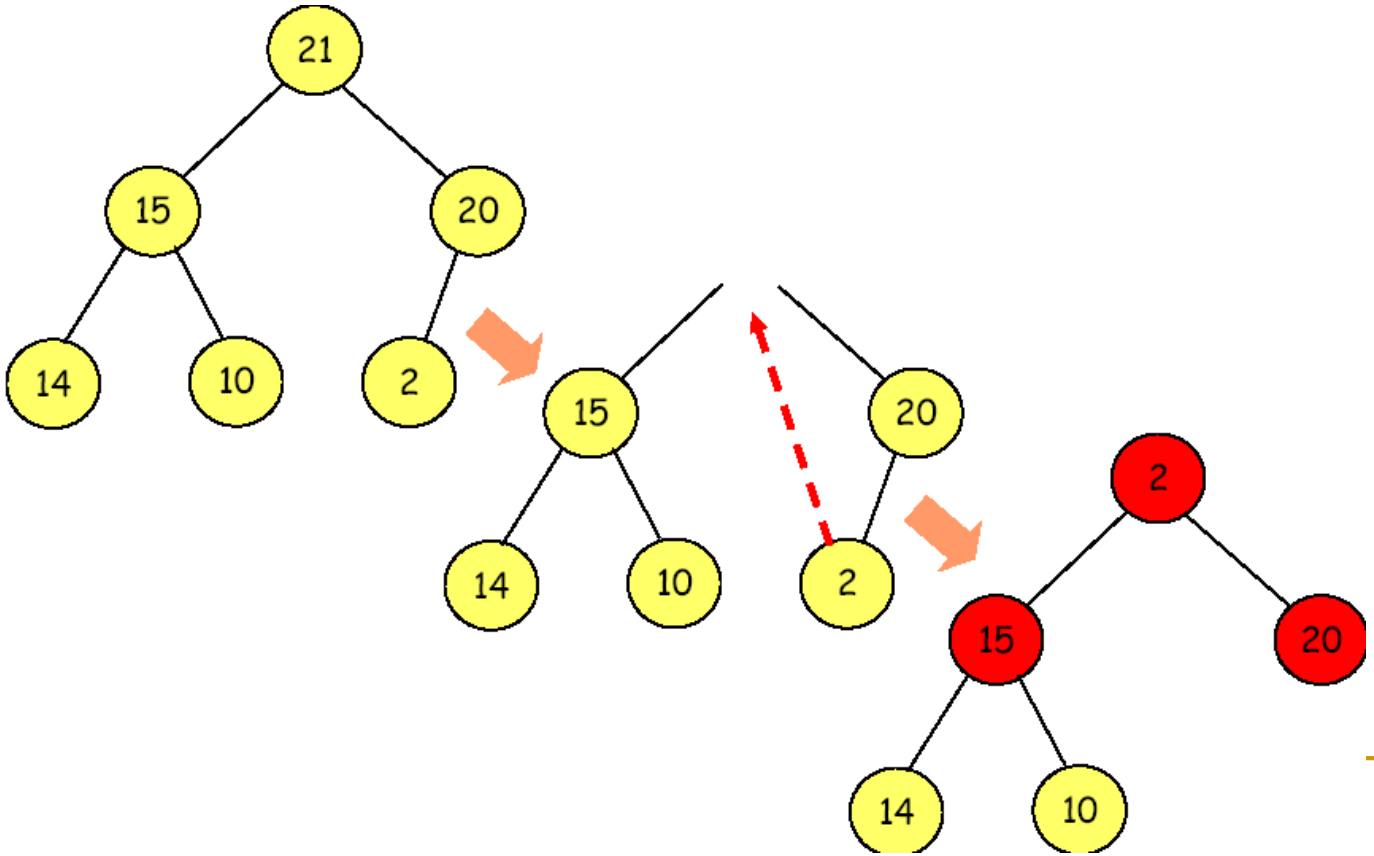
تابع درج در یک هرم

```
template <class T>
void MaxHeap<T>::Insert(const Element<T>& x) {
  if(n == MaxSize)  // heap is already full
  {  HeapFull(); return; }

  ++n;  // increment heap size by 1

  // move down x's ancestors with smaller key
  for(int i = n; i != 1; i /= 2) {
    if(x.key <= heap[i/2].key) break;  // parent is not smaller
    heap[i] = heap[i/2]; // move down the smaller parent
  }

  heap[i] = x;  // insert x into the right position
}
```

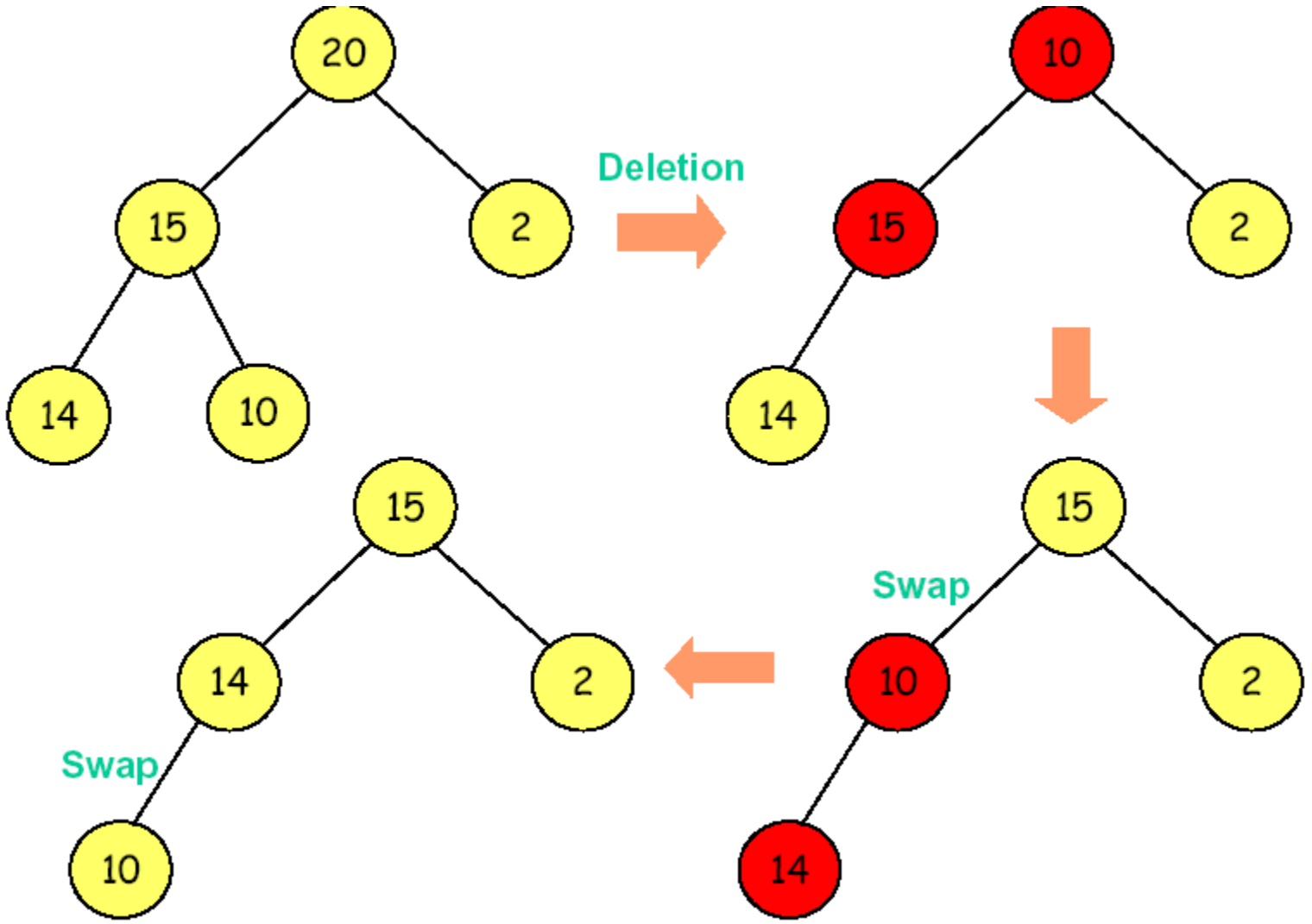
پیچیدگی زمانی؟

حذف از یک هرم

- حذف از هرم از ریشه صورت می گیرد.
- با حذف ریشه آخرین گره درخت جایگزین ریشه می شود.(سمت راسترین گره در آخرین سطح)
- گره ریشه با فرزندان مقایسه می شود و هرکدام بزرگتر بود جایگزین پدر می شود.
- تا رسیدن به حالتی که پدر از فرزندان بزرگتر باشد این کار ادامه پیدا می کند.



حذف از یک هرم

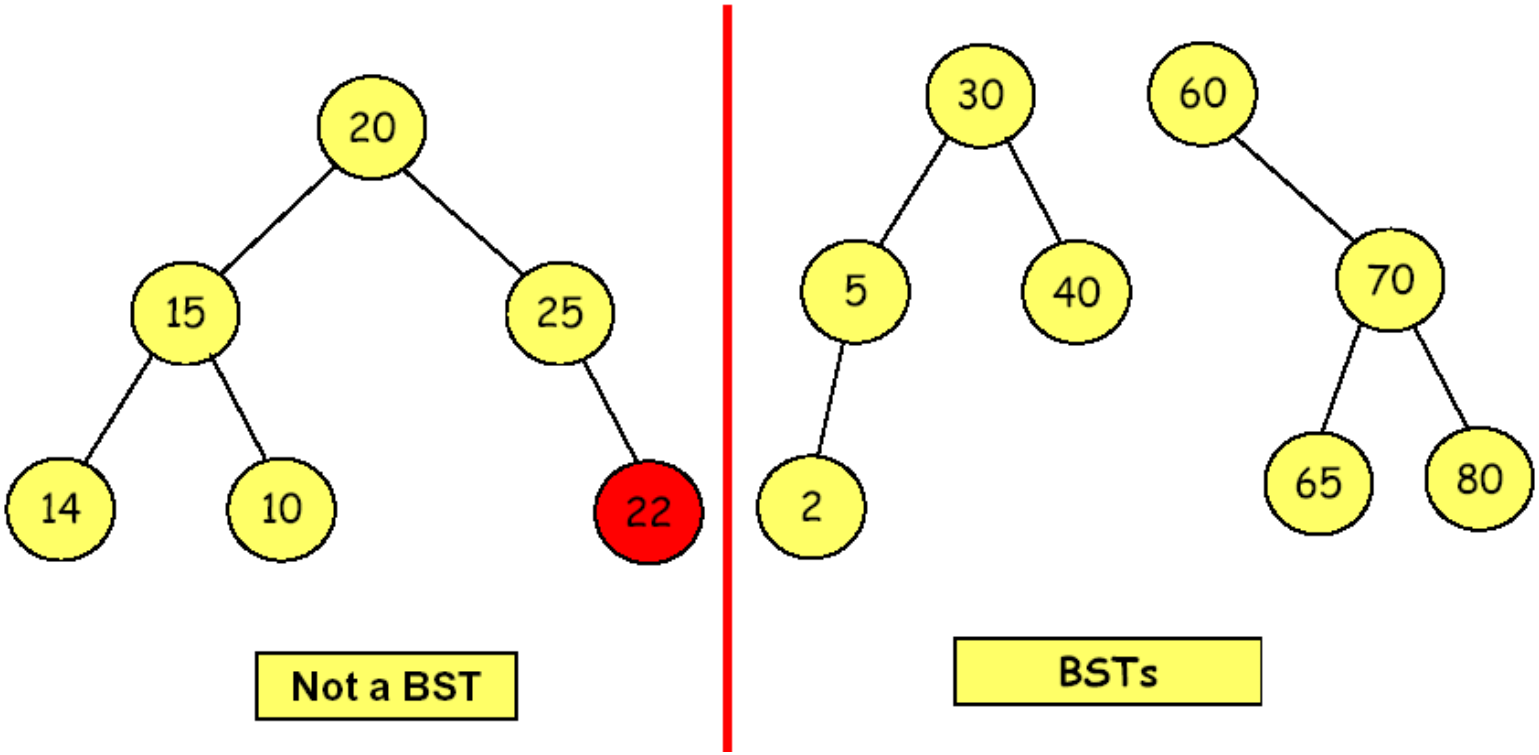


Template <class Type>

```
Element <type> *MaxHeap <Type>::DeleteMax(Element <type> &x){
if (!n) {HeapEmpty();return 0;}
x= heap[1]; Element <type>  k= heap[n];n--;
for (int i=1,j=2; j<=n;)
{if (j<n) if (heap[j].key< heap[j+1].key)j++;
//j points to the larger child
If(k.key >= heap[j].key) break;
heap[i]= heap[j];
i= j; j*=2;
}
heap[i]=k;
return &x;
}
```

درختان جستجوی دودویی

- **یک درخت جستجوی دودویی یک درخت دودویی است که اگر تهی نباشد دارای خواص زیر است:**
  - هر عنصری دارای یک کلید است. کلیدها باید منحصر بفرد باشند.
  - کلیدهای واقع در زیر درخت سمت چپ باید از کلید کوچکتر از ریشه باشند.
  - کلیدهای واقع در زیر درخت راست باید از کلید ریشه بزرگتر باشند.
  - زیر درختان چپ و راست نیز درختان جستجو هستند.



جستجوی یک درخت دودویی

- از الگوریتم زیر برای جستجو استفاده می شود.
  - اگر مقدار مورد جستجو با ریشه برابر است که پایان جستجو است و کلید پیدا شده است.
  - اگر مقدار مورد جستجو از ریشه کمتر باشد زیر درخت سمت چپ را جستجو می کنیم
  - اگر مقدار مورد جستجو از ریشه بیشتر باشد زیر درخت سمت راست را جستجو می کنیم.

تابع جستجو به صورت بازگشتی

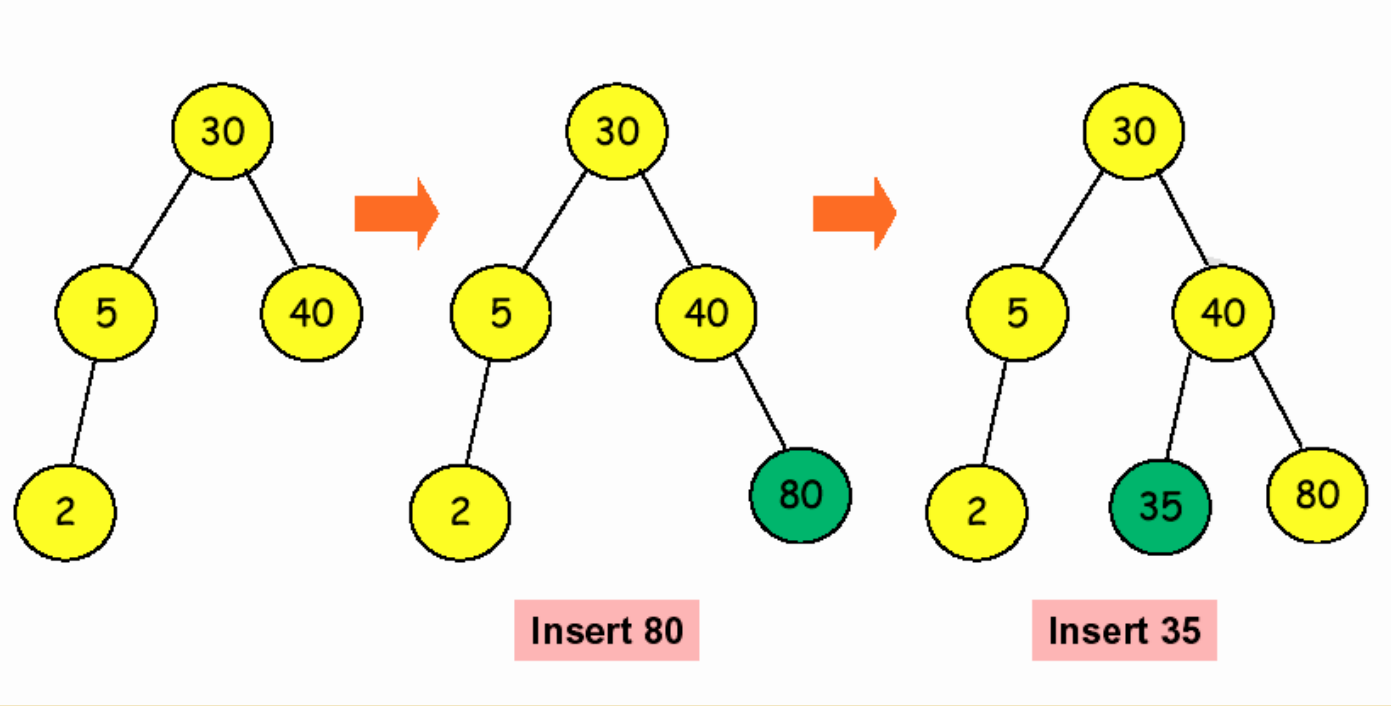
```
template <class T>
BstNode<T>* BST<T>::Search(const Element<T>& x) {
  return Search(root, x);
}

template <class T>
BstNode<T>* BST<T>::Search(BstNode<T>* b, const Element<T>& x){
  if(! b) return 0; // not found
  if(x.key == b->data.key) return b; // found
  if(x.key <  b->data.key)
    return Search(b->LeftChild, x);  // search left subtree
  return Search(b->RightChild, x);  // search right subtree
}
```

جستجوی درخت به صورت غیر بازگشتی

```
template <class T>
BstNode<T>* BST<T>::IterSearch(const Element<T>& x) {
  for(BstNode<T>* t = root; t; ) {
    if(x.key == t->data.key) return t; // found
    if(x.key <  b->data.key)
      t = t->LeftChild;  // search left subtree
    else
      t = t->RightChild; // search right subtree
  }
  return 0; // not found
}
```

ساختن یک درخت جستجو

- اضافه کردن یک گره به یک درخت جستجو
  - برای اضافه کردن یک گره جدید باید عمل جستجو انجام شود و آخرین گره ملاقات شده گره ای است که باید گره به آن اضافه شود. اگر بزرگتر بود به سمت راست و گرنه به سمت چپ.



```
template <class T>
bool BST<T>::Insert(const Element<T>& x) {
  BstNode<T> *p = root, *q = 0;
  while(p) {
    q = p;
    if(x.key == p->data.key) return false; // an existing key
    if(x.key <  p->data.key)
      p = p->LeftChild;  // move to left subtree
    else
      p = p->RightChild; // move to right subtree
  }
  p = new BstNode<T>;
  p->LeftChild = p->RightChild = 0; p->data = x; // make a copy
  if(! root) root = p;  // an empty BST originally
  else if(x.key < q->data.key) q->LeftChild = p;
  else q->RightChild = p;
  return true;
}
```