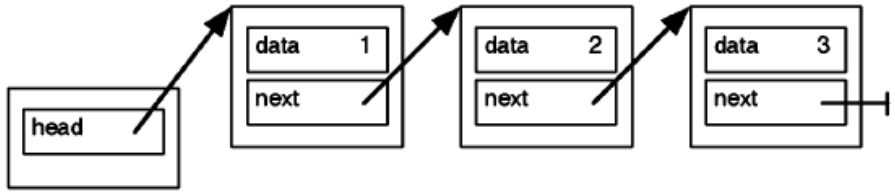
Recitation 08

Lists & BSTs

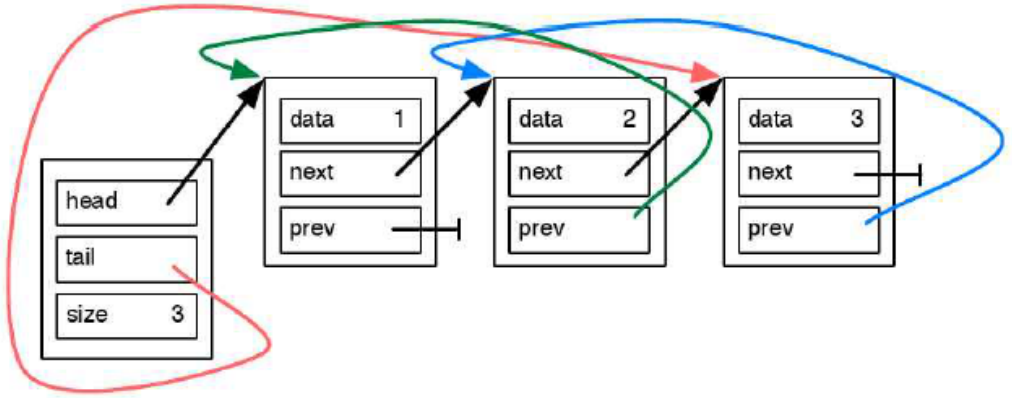
George Mappouras

11/3/2017

Lists



(a) An singly linked list with elements 1, 2, and 3.



(b) A doubly linked list with a tail pointer and a field containing the number of elements

Lets write a linked list class in C++

What do we need?

- Class list
  - head
  - tail (for doubly linked)
  - node class
    - data
    - next
    - prev

Lets write a linked list class in C++

```
template<typename T>
class LinkedList {
  class Node {
    public:
     T data;
     Node * next;
     Node * prev;
  };
  Node * head;
  Node * tail;
  size_t size;
public:
  LinkedList() : head (NULL), tail (NULL), size(0) {}
};
```

Adding functionalities:
1)Add Node to the front

```
void addFront(int data){
  head = new Node(data, head);
  if (tail == NULL){
    tail = head;
  }
  else{
    head->next->prev = head;
  }
  size++;
}
```

Adding functionalities:
2)Add Node to the back

```
void addBack(int data){
  tail = new Node(data, NULL, tail);
  if (head == NULL){
    head = tail;
  }
  else{
    tail->prev->next = tail;
  }
  size++;
}
```

Adding functionalities: 3)Proper Destructor

```
~LinkedList(){
  while (head != NULL){
    Node * temp = head->next;
    delete head;
    head = temp;
  }
```

Adding functionalities:
4)Add node in sorted list

```
void addSorted(const T & data){
  Node ** curr = &head;
  while (*curr != NULL && data > (*curr)->data){
        curr = &(*curr)->next;
   }
      *curr = new Node(data, *curr);
        size++;
  }
```

Adding functionalities: 5)Remove node

```
void removeNode(const T & data){
  Node ** curr = &head;
  Node * curr2 = head;
  while (*curr != NULL && data != (*curr)->data){
        curr = &(*curr)->next;
   }
   if (*curr != NULL){
        Node * tmp = *curr;
        *curr = (*curr)->next;
        (*curr)->next->prev = tmp->prev //check
        delete tmp
    }
        size--;
  }
```

Adding functionalities:
6)Iterators (sub class inside LinkedList)

```
class iterator {
  Node * current;
  public:
    iterator(): current(NULL) {}
    iterator(Node * c): current(c) {}
    iterator & operator++(){
      current = current->next;
      return *this;
    }
     iterator begin() {return iterator(head);}
     iterator end() {return iterator(NULL);}
}
```

Adding functionalities:
6)Iterators (sub class inside LinkedList)

```
….
class iterator {
  Node * current;
  public:
    iterator(): current(NULL) {}
    iterator(Node * c): current(c) {}
    iterator & operator++(){
      current = current->next;
      return *this;
    }
}
     iterator begin() {return iterator(head);}
     iterator end() {return iterator(NULL);}
}
```

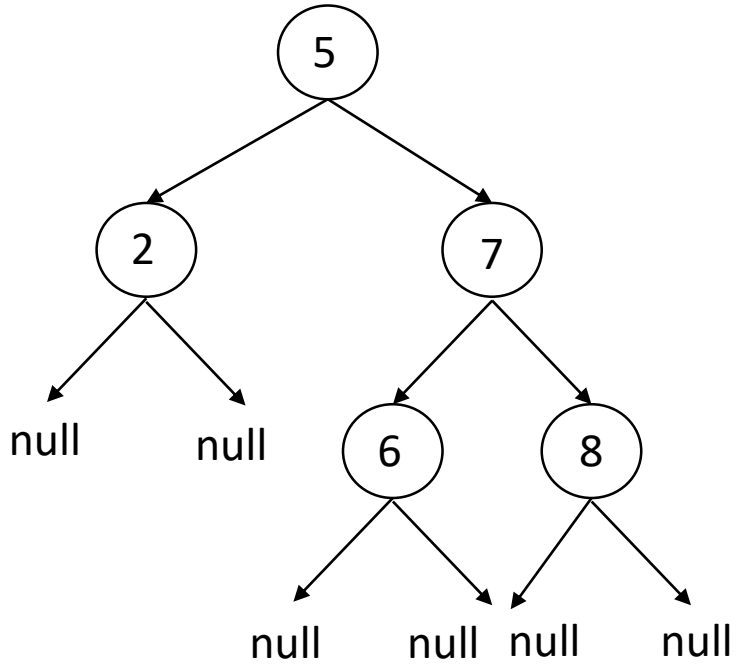
Example:

```
LinkedList<int> mylist = … ;
for (iterator it = mylist.begin(); it != mylist.end(); it++ ){
            //do stuff here
}
```

Adding functionalities:
6)Reverse List

```
LinkedList<T> reverse(const LinkedList & list){
  LinkedList<T> ans;
  for (typename LinkedList::iterator it = list.begin(); it != list.end(); ++it){
    ans.addFront(*it);
  }
   return ans;
}
```

Binary Search Trees (BSTs)



The BST class

```
template<typename T>
class BinaryTree{
  class Node{
    T data;
     Node * right;
     Node * left;
     Node * parent;
  }
  Node * root;
  size_t hight;
  BinTree() : head (NULL) hight (0){}
}
```

Search for a note

```
Node * findNode(T findData){
    Node * current = root;
    while (current != NULL){
        if(findData == current -> data) {
                return current
        }
        else if (findData < current->data){
                current = current->left;
        }
        else {
                current = current->right;
        }
}
return current
```

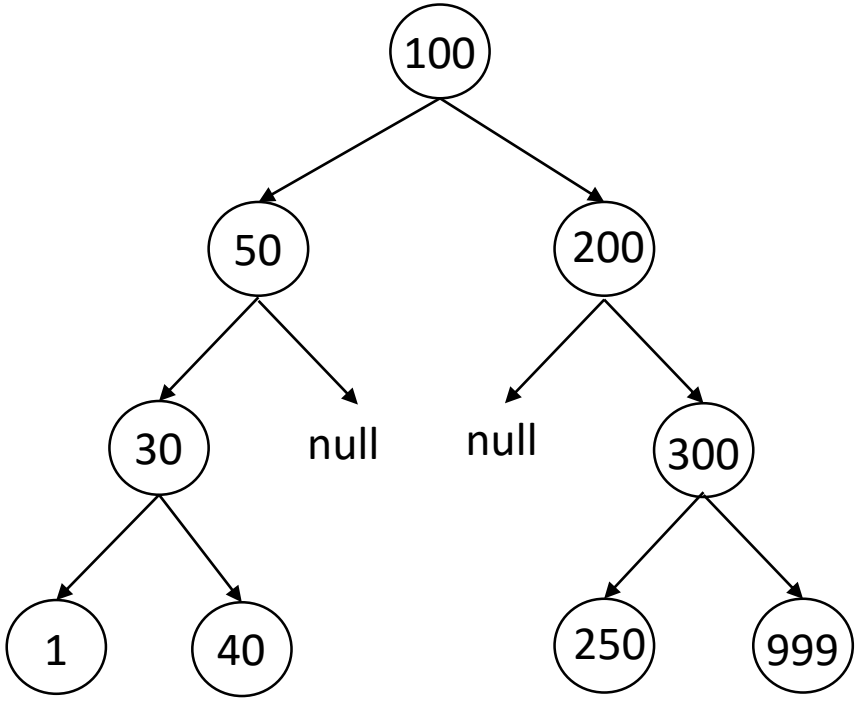
Adding Node

```
void add (T newData){
    Node ** curr = &root;
    while (*curr != NULL){
        if(newData < (*curr)->data){
            curr = &(*curr)->left;
        }
        else{
            curr = &(*curr)->right;
        }
    }
    *curr = new Node(newData);
}
```

Removing a Node

```
Node* deleteNode(Node * curr, T data){
  if(curr == NULL) return curr;
  else if(data < curr->data) curr->left = deleteNode(curr->left,data);
  else if(data > curr->data) curr->right = deleteNode(curr->right, data);
  else {
    // Case 1: No Child
    if(curr->left == NULL && curr->right == NULL){
      delete curr;
      curr = NULL;
    // Case 2: one child
    } else if(curr->left == NULL){
      Node *temp = curr;
      curr = curr->right;
      delete temp;
    } else if(curr->right == NULL){
      Node *temp = curr;
      curr = curr->left;
      delete temp;
    } else{
      Node *temp = findMin(curr->right);
      curr->data = temp->data;
      curr->right = deleteNode(curr->right, temp->data);
    }
  }
  return curr;
}
```

$$100, 50, 30, 200, 300, 250, 40, 1, 999$$



Binary Tree Traversals

1. Inorder

```
void printInorder(Node * current){
  if(current!=NULL){
    printInorder(current->left);
    std::cout<<current->data<<" ";
    printInorder(current->right);
  }
} //from left corner, down-up
```

2. Preorder

```
void printPreorder(Node * current){
  if(current!=NULL){
    std::cout<<current->data<<" ";
    printPreorder(current->left);
    printPreorder(current->right);
  }
}
```

3. Postorder

```
void printPostorder(Node * current){
  if(current!=NULL){
    printPostorder(current->left);
    printPostorder(current->right);
    std::cout<<current->data<<" ";
  }
}
```

1. Inorder

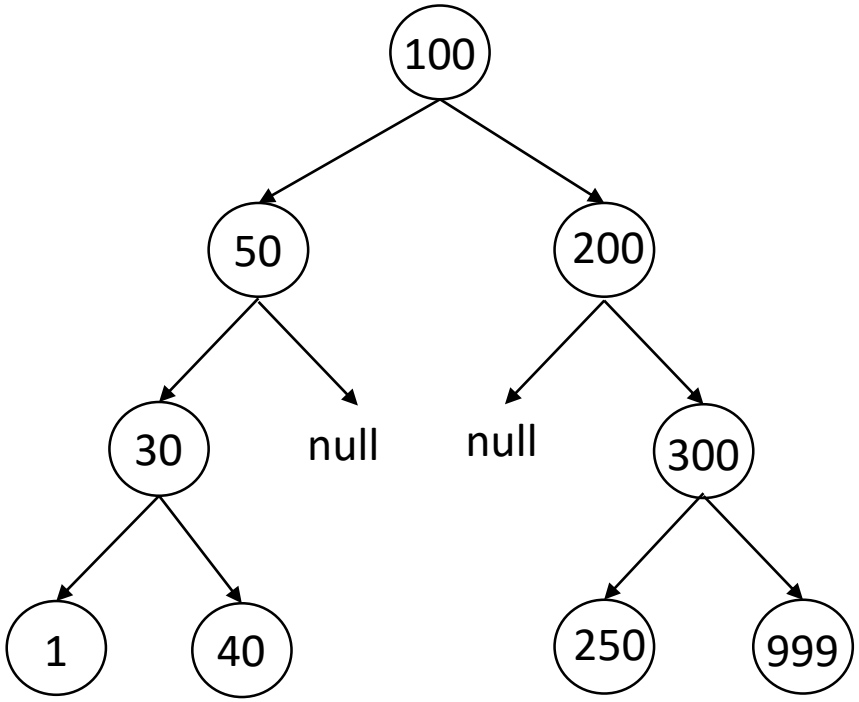
```
void printInorder(Node * current){
  if(current!=NULL){
    printInorder(current->left);
    std::cout<<current->data<<" ";
    printInorder(current->right);
  }
} //from left corner, down-up
```

2. Preorder

```
void printPreorder(Node * current){
  if(current!=NULL){
    std::cout<<current->data<<" ";
    printPreorder(current->left);
    printPreorder(current->right);
  }
}
```

3. Postorder

```
void printPostorder(Node * current){
  if(current!=NULL){
    printPostorder(current->left);
    printPostorder(current->right);
    std::cout<<current->data<<" ";
  }
}
```



Preorder 100, 50, 30, 1, 40, 200, 300, 250, 999

Inorder 1, 30, 40, 50, 100, 200, 250, 300, 999

Postorder 1, 40, 30, 50, 250, 999, 300, 200, 100

Binary Search on a sorted array

- Assume a sorted array (f.e. [1 3 5 10 25 36])
- How can we implement the idea of BST on that array?
- Tip: every element has larger numbers on its right and smaller on its left
  - Implement a BS on an array to find a given element
  - What is the big O time for a naive search?
  - What is the big O time for this implementation?

```
size_t binarySearch(int toFind, int * array, size_t n) {
  size_t lo = 0;
  size_t hi = n;
  while (lo < hi) {
    size_t mid = (hi + lo) / 2 ;
    if (array[mid] == toFind) {
      return mid;
    }
    if (toFind < array[mid]) {
      hi = mid;
    }
    else {
      lo = mid + 1;
    }
  }
  return -1;
}
```