CSE 100: C++ TEMPLATES AND ITERATORS (CONTD), AVERAGE CASE ANALYSIS OF FIND
Q1: What is the worst case time to insert into a binary search tree in terms of \(N\), the number of nodes in the tree? We make no assumptions here about the tree's structure.

A. \(O(1)\)
B. \(O(\log N)\)
C. \(O(N)\)
D. \(O(N^2)\)
Q2: Which of the following is one of the two probabilistic assumptions made during the average case analysis of a successful find in a BST?

A. The tree is approximately balanced
B. All keys in the tree are equally likely to be searched for
C. The data is inserted into the tree in increasing order
D. The tree contains only positive numbers

B. All keys in the tree are equally likely to be searched for
Q3: What is a recurrence relation used for in the average case analysis of a successful find in a BST?

A. The average BST total depth
B. The average number of nodes in the BST
C. The average of the elements in the tree
D. The average time to find an element in the tree.
Q4: How many different ways can N keys be inserted into a BST?

A. N
B. $2^N$
C. $N!$
D. $2^N - 1$
BST, with templates:

```cpp
template<typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }
};
```
BST, with templates:

```cpp
template<
    typename Data
>

class BSTNode { 
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }
};
```

1. How would you create a `BSTNode` object with integer data on the runtime stack?

```cpp
BSTNode<int> bl(1);
```

- Constructor needs parameters
2. How would you create a pointer to BSTNode with integer data?

```cpp
BSTNode<int>* p;
BSTNode<Data>* P;
```
BST, with templates:

```cpp
template<typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }
};
```

3. How would you create an `BSTNode` object on the heap?

```cpp
BSTNode<int> * p = new BSTNode<int>(1);
```
BST, with templates:

```cpp
template<
    typename Data>

class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = 0;
        }

};
```

BSTNodes will be used in a BST, and with a BSTIterator…
CHANGING GEARS: C++STL and BSTs

- The C++ Standard Template Library is a very handy set of built-in data structures (containers), including:

  - array
  - vector
  - deque
  - forward_list
  - list
  - stack
  - queue
  - priority_queue
  - set
  - multiset (non unique keys)
  - unordered_set
  - map
  - unordered_map
  - multimap
  - bitset

  Of these, set is one that is implemented using a balanced binary search tree (typically a red-black tree)
Imagining ourselves as C++ STL class designers…

- set’s find function has this prototype:

```cpp
template <typename T>

class set {
  public:
    iterator find ( T const & x ) const;
}
```

What does the final const in the function header above mean?

A. find cannot change its input argument ✗
B. find cannot change where its input argument, which is a pointer, points to
C. find cannot change the underlying set
Imagining ourselves as C++ STL class designers...

- set’s find function has this prototype:

  ```cpp
template <typename T>

class set {

public:
    iterator find ( T const & x ) const;

};
```

The documentation for set’s find function says:

*Searches the container for an element with a value of x and returns an iterator to it if found, otherwise it returns an iterator to the element past the end of the container.*
C++ STL Iterators

• What is an iterator?

'generic' pointer to containers.
What do we do with pointers on basic data structures like arrays?

**Access/Array Elements**

- \( \text{arr} \)
- \( \text{p} \)

**Set**

- \( *\text{p} \): access array elements
- \( \text{p}++ \): traverse array
- \( ++\text{p} \)

**Check if pointing to same element**

- \( \text{p} == \text{q} \)
- \( \text{p} != \text{q} \)
C++ STL Iterators

What is an iterator?

- In the iterator pattern of OO design, a container has a way to supply to a client an iterator object which is to be used by the client to access the data in the container sequentially, without exposing the container’s underlying representation.
C++ STL Iterators

```c++
set<int> c;
...
// get an iterator pointing to container's first element
set<int>::iterator itr = c.begin();
```

What do you think `begin()` returns?
A. The address of the root in the set container class
B. The address of the node with the smallest data key
C. The address of the smallest data key
D. None of the above
Iterator class template for BST

```cpp
template <typename T>
class BSTIterator {

private:
    Node<T>* curr;

public:
    /** Constructor */
    BSTIterator(Node<T>* n) : curr(n) {}

Is this definition of the BSTIterator class complete?
A. Yes
B. No
set<int> c;
...

// get an iterator pointing to container’s first element
set<int>::iterator itr = c.begin();

// get an iterator pointing past container’s last element
set<int>::iterator end = c.end();

// loop while itr is not past the last element
while (itr != end) {
    cout << *itr << endl; // dereference the itr to get data
    ++itr; // increment itr to point to next element
}
template<typename Data>
class BSTIterator : public std::iterator<std::input_iterator_tag, Data> {

private:
    BSTNode<Data>* curr;

public:
    /** Constructor. Use the argument to initialize the current BSTNode * in this BSTIterator. */ // TODO
    BSTIterator(BSTNode<Data>* curr) { // TODO }

    /** Dereference operator. */
    Data operator*() const {
        return curr->data;
    }

    /** Pre-increment operator. */
    BSTIterator<Data>& operator++() {
        curr = curr->successor();
        return *this;
    }
}
What kind of traversal is the above code doing?

A. In order
B. Pre order
C. Post order
D. None of the above

What kind of traversal is the above code doing?

A. In order
B. Pre order
C. Post order
D. None of the above
What are the different ways that we can implement an in order traversal of a BST?

Bored? Implement remove in a BST. Discuss the merits of the C++ iterator pattern.
Average case analysis

• Warning! There will be math 😊
• Why is it important that we do this?
  • So you have a hope of doing it yourself on a new data structure (perhaps one you invent?)
  • Mathematical analysis can be insightful!
Average case analysis of a “successful” find

Given a BST having:

- $N$ nodes $x_1, \ldots, x_N$, such that $\text{key}(x_i) = k_i$

How many compares to locate a key in the BST?

1. Worst case: $H$: height of the tree

2. Best case: 1

3. Average case:

Expected no. of compares over all key searches
Given a BST having:

- $N$ nodes $x_1, \ldots, x_N$ such that $\text{key}(x_i) = k_i$
- Probability of searching for key $k_i$ is $p_i$

What is the expected number of comparisons to find a key?

A. $\sum_{i=1}^{N} p_i \cdot (\text{No. of comparisons to find } k_i)$

B. $\sum_{i=1}^{N} p_i \cdot x_i$

C. $\left( \sum_{i=1}^{N} \text{No. of comparisons to find } k_i \right) / N$
Number of compares to find key $k_i$ is related to the Depth of $x_i$ in the BST

- **Depth** of node $x_i$: No. of nodes on the path from the root to $x_i$ inclusive
- Notation for depth of $x_i$:
Given a BST having:

- N nodes $x_1, \ldots, x_N$ such that key($x_i$) = $k_i$
- Probability of searching for key $k_i$ is $p_i$

What is the expected number of comparisons to find a key?

A. $\sum_{i=1}^{N} p_i \cdot (\text{No. of comparisons to find } k_i)$

B. $\sum_{i=1}^{N} p_i \cdot x_i$

C. $\left(\sum_{i=1}^{N} \text{No. of comparisons to find } k_i\right) / N$