CSE 100: BSTS, BIG O, AND C++

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to CSE100" << endl;
    return 0;
}
```
Announcements

- PA0 due Tuesday @10am. PA1 coming Tuesday (noon). Will be due a week from the following Friday.
  - Concentrate on the reading assignment until then
CLICKERS OUT

Frequency BD
Binary Search Tree – What is it?
Which of the following is/are a binary search tree?

A. 
```
42
32
12
```

B. 
```
42
12
32
```

C. 
```
42
12
32
65
```

D. 
```
42
32
56
12
45
```

E. More than one of these
Binary Search Trees

- What are the operations supported and their running times?

- Why do we get these running times?

- How do you implement the data structure (and operations supported by it)?
Binary Search Trees

• What is it good for?
  • If it satisfies a special property i.e. Balanced, you can think of it as a dynamic version of the sorted array
Operations supported: sorted array

Operations
- Search
- Selection
- Min/Max
- Predecessor/ Successor
- Rank

Output in sorted order

Running Time
(sorted array)

Ref: Tim Roughgarden (Stanford)
## Operations supported: sorted array

<table>
<thead>
<tr>
<th>Operations</th>
<th>Running Time (sorted array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>Selection</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Min/Max</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Predecessor/ Successor</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Rank</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>Output in sorted order</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Ref: Tim Roughgarden (Stanford)
<table>
<thead>
<tr>
<th>Operations</th>
<th>Running Time (Balanced-BST)</th>
<th>Running Time (sorted array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(\log_2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Selection</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Min/Max</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Predecessor/ Successor</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Rank</td>
<td>$O(\log_2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Output in sorted order</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>$O(n)$</td>
<td></td>
</tr>
</tbody>
</table>

Ref: Tim Roughgarden (Stanford)
## Operations supported: Balanced Binary Search Trees vs. Sorted Array

<table>
<thead>
<tr>
<th>Operations</th>
<th>(Balanced-BST)</th>
<th>(sorted array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(\log_2 n)$</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>Selection</td>
<td>$O(\log_2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Min/Max</td>
<td>$O(\log_2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Predecessor/ Successor</td>
<td>$O(\log_2 n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Rank</td>
<td>$O(\log_2 n)$</td>
<td>$O(\log_2 n)$</td>
</tr>
<tr>
<td>Output in sorted order</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log_2 n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log_2 n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Under the hood of the BST: Searching an element in the tree

To search for element with key \( k \)
1. Start at the root
2. If \( k < \text{key(root)} \), recursively search \( T_L \)
   Else recursively search \( T_R \)
Under the hood of the BST: Searching an element in the tree

Q: In the worst case how many comparisons do we need to find an element in any tree and in particular the given tree

A. The number of nodes in the tree, 5
B. The number of leaves in the tree, 3
C. The number of roots in the tree, 1
D. None of the above
Towards describing the running time of the search operation: Height of the tree

*Height of a node:* Longest path from node to any leaf node plus one

*Height of the tree:* Height of the root

What is the height of the given tree?
What about the running time of Search?

How long does it take to find an element in a tree with $N$ keys and height $H$ in the worst case?

A. $O(1)$  
B. $O(\log_2 N)$  
C. $O(H)$  
D. $O(N)$  
E. More than one correct answer
Relating H (height) and N (#nodes) for a Balanced BST

How many nodes are on level L in a completely filled binary search tree?
A. 2
B. L
C. 2*L
D. $2^L$
Relating $H$ (height) and $N$ (#nodes)

How many nodes are in a completely filled BST?

A. $N = \sum_{L=0}^{H-1} 2^L$
B. $N = 2^L$
C. $N = \sum_{H=0}^{N} 2^H$
D. $N = 2^{H+L} - 1$
Relating H (height) and N (#nodes)

OK, so…where do we go from here?

\[ N = \sum_{L=0}^{H-1} 2^L \]

A. We are done. We now have a formula that relates H and N.
B. We need to represent the sum in closed form.
C. We need to solve for H.
Relating $H$ (height) and $N$ (#nodes)

Representing the sum in closed form:

$$N = \sum_{L=0}^{H-1} 2^L = 2^H - 1$$
Relating \( H \) (height) and \( N \) (#nodes)

\[
N = \sum_{L=0}^{H-1} 2^L = 2^H - 1
\]

Finally, what is the height of the tree in terms of \( N \)?

A. \( H = (N + 1) / 2 \)  
B. \( H = \log_2(N) \)  
C. \( H = \log_2(N + 1) \)

And since we knew finding a node was \( O(H) \), we now know it is:
Summary: Running Time of Search in a BST

Worst case running time of search in any generic BST is $O(H)$

Worst case running time of search in a balanced BST is $O(\log_2 N)$
Which of the following statements is a genuine advantage of Binary Search Trees over a linked list data structure?

A. Binary search trees can be faster to retrieve information from
B. Binary search trees use less memory
C. Binary search trees are easier to implement
D. Binary search trees are typically built into most languages, while linked lists are not
And now… C++

C++’s main priority is getting correct programs to run as fast as it can; incorrect programs are on their own.

Java’s main priority is not allowing incorrect programs to run; hopefully correct programs run reasonably fast, and the language makes it easier to generate correct programs by restricting some bad programming constructs.

-- Mark Allen Weiss, *C++ for Java Programmers*

Why C++ for data structures?
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode(int d) {
        data = d;
    }
}
```

C++, attempt 1:

```cpp
class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode(const int & d) {
        data = d;
    }
};
```

Which of the following is a problem with the C++ implementation above?
A. You should not declare the types of your variables in C++
B. The class BSTNode should be declared public
C. The semicolon at the end of the class will cause a compile error
D. In C++ you specify public and private in regions, not on each variable or function
C++, attempt 2:

class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) {
        data = d;
    }
};

The code above in red specifies that \( d \) is passed by constant reference. Which of the following diagrams best represents what that means?

A. \[ \text{myInt: } 42 \quad \text{d: } 42 \]
   d is not allowed to change what’s in its box

B. \[ \text{myInt: } 42 \quad \text{d: } \]
   The address in d’s box can’t be changed

C. \[ \text{myInt: } 42 \quad \text{d: } 42 \]
   d can’t change myInt because there’s no connection
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 2:

```cpp
class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) {
        data = d;
    }
};
```

Which of the following is a problem with the C++ implementation above?
A. Because data is a constant variable, the constructor will cause an error.
B. You cannot pass an integer by reference into a function. Integers must be passed by value.
C. Since d is passed by reference, you cannot assign its value to data, which is an int. You need to dereference it first.
D. The constructor needs a semi-colon at the end of its definition.
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 2:

```cpp
class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) {
        data = d;
    }
};
```

Which of the following is a problem with the C++ implementation above?
A. Because data is a constant variable, the constructor will cause an error. We need to use an initializer list

Why make data const?
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ){
        data = d;
    }
}
```

C++, attempt 3:

```cpp
class BSTNode {
public:
    BSTNode left;
    BSTNode right;
    BSTNode parent;
    int const data;

    BSTNode( const int & d ) :
    data(d) {  }
};
```

What is the problem with how we have declared left, right and parent above?
A. They should be `BSTNode*` (pointers to `BSTNodes`) and not `BSTNode` type.
B. They should be declared to be `const`
C. They should be declared as `BSTNode&` (reference variables).
In Java:

```java
public class BSTNode {
    public BSTNode left;
    public BSTNode right;
    public BSTNode parent;
    public int data;

    public BSTNode( int d ) {
        data = d;
    }
}
```

C++, attempt 4:

```cpp
class BSTNode {
public:
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
    int const data;

    BSTNode( const int & d ) :
        data(d) { }
};
```