```
template <typename Key>
class Heap : public BST<Key> {
public:
    Heap();
    ~Heap();
    void push(const Key& newKey);
    void pop();
    const Key& top() const;  // throws std::out_of_range if empty
    bool empty() const;
private:
    // no data members necessary since we inherit BST
};

template<typename Key>
Heap<Key>::Heap()  // BST's constructor will be called automatically here
{
}

template<typename Key>
Heap<Key>::~Heap()
{
}// BST's destructor will be called automatically here

template<typename Key>
void Heap<Key>::push(const Key& newKey)
{  this->insert(newKey);  }
}

template<typename Key>
void Heap<Key>::pop()
{  this->remove(this->top());  }
}

template<typename Key>
const Key&
Heap<Key>::top() const
{
    if(nullptr == this->root_) {
        throw std::out_of_range("heap is empty");
    }
    Node<Key> *start = this->root_;
    while(start->right_ != nullptr)
    {
        start = start->right_;
    }
    return start->key_;
}

template<typename Key>
bool Heap<Key>::empty() const
{ return this->empty(); }
```

4.2. Analyze the runtime of your `top()` implementation (show a very short justification or work).

Answer: Theta(n)
Justification: Top() must move to the bottom right node of the BST. Since the BST does not guarantee balance this will be a linear runtime => $\sum_{i=1}^{height} 1 = \sum_{i=1}^{n} 1 = \theta(n)$