FINAL PRACTICE

Final exam: In person

Time: Tuesday March 15, noon - 3p

Location: BUCHN 1910

Read the instructions for the exam carefully:

https://ucsb-cs24.github.io/w22/exam/e02/ *

* 1 page of notes is okay.
* seating is assigned
* State your assumptions
* Bring your IDs.
* Dark pencil or pen

input size

$T(n) = O(n^3)$

- The runtime complexity of an algorithm is $T(n) = 5n^2logn + n + 1000$
- Show that $T(n) = \theta(n^2logn)$ using the definition of Big-Theta

$T(n) = \Theta(g(n))$

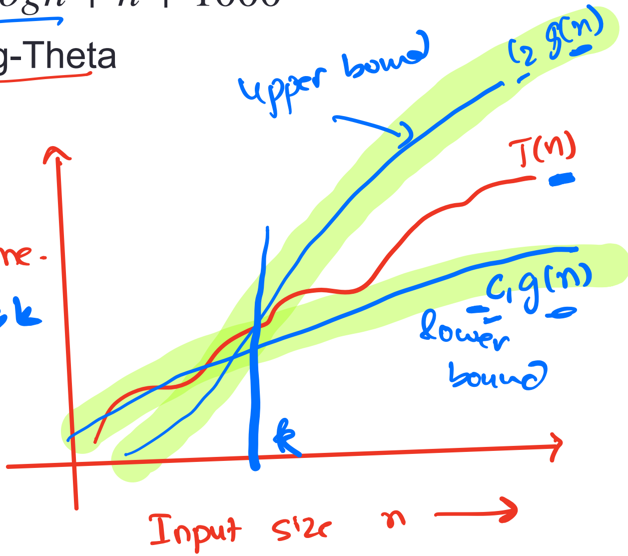
$= O(g(n))$

$c_1 > 0 \quad c_2 > 0 \quad g(n) \quad k \geq 0$

$c_2 g(n) \leq T(n) \leq c_1 g(n)$, for all $n \geq k$

$5n^2 \log n + \boxed{n} + 1000$

- $n \leq n^2 \log n$, for $n \geq 2$
- $1000 \leq 10\, n^2 \log n$, for $n \geq 10$



Therefore $T(n) \leq 5n^2\log n + n^2\log n + 10n^2\log n$

$= \underbrace{16}_{c_2} n^2\log n \qquad n \geq 10$

— (1)

$\underline{3}\, n^2\log n \leq \underbrace{5n^2\log n} + \cancel{n} + \underline{1000}, \quad n \geq 10$

$c_1 = 3$

— (2)

$T(n) = \Theta(n^2\log n)$

$c_1 = 3 \qquad c_2 = 16 \qquad k = 10$

- Assume **dataA** is some data structure and the input vector **v** has N key
- Describe algoX in a sentence

```
void algoX(vector<int>& v)
{
        dataA ds; // empty
        for(auto& elem: v)
            ds.insert(elem);
        for(auto& elem: v){
            elem = ds.min();
            ds.delete(elem);
        }
}
```

- Assume **dataA** is some data structure and the input vector to algoX has N numbers
- Given: running time of operations for dataA, where M is the number of keys stored in dataA

M → no. of keys already stored in the data structure.

worst case
- insert: O(log M)
- min: O(1)
- delete: O(log M)

← Make use to upperbound

```
void algoX(vector<int>& v)
{
        dataA ds;                 // empty
        for(auto& elem: v)
            ds.insert(elem);
        for(auto& elem: v){
            elem = ds.min();
            ds.delete(elem);
        }
}
```

O(1)

n. O(log n)

n.(O(1)+ O(log n))

What is the Big-O running time of algoX?

A. $O(N^2)$
B. O(N logN)
C. O(N)
D. O(log N)
E. Not enough information to compute

N-1 keys

$c_1 < c_2 < c_3 < \ldots < c_n$

→ maximum time O(log N)

ds.insert(elem) takes a variable time in each iteration but we can upper bound it by the time it takes to do the last insert

$$\text{ds.insert(elem)} \leq \text{ds.insert(last elem)} = O(\log N)$$

First delete will take the most time because ds has the most keys (N). Use that to upper bound the runtime of all subsequent deletions.

$$T(n) = \underbrace{O(1)}_{ds.} + \underline{n} \cdot O(\log n) + \underline{n}\left(\underbrace{O(\log n)}_{delete} + \underbrace{O(1)}_{min}\right)$$

$$= O(n \log n)$$

$$T(n) = \underline{n}^2 \underline{m} + \underline{n}^2 \underline{\log m}$$

$$= O(n^2 m)$$

$$O(\underline{n}^2 m + n \cdot m \cdot \underline{k}^3)$$

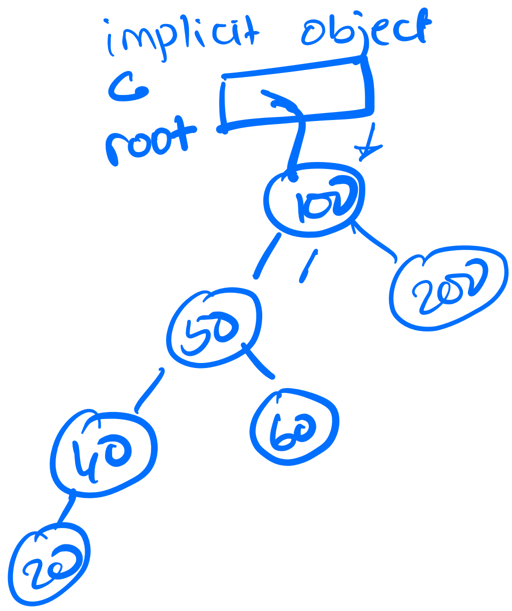
Big Four and the Rule of Three

Implement the copy constructor for a BST

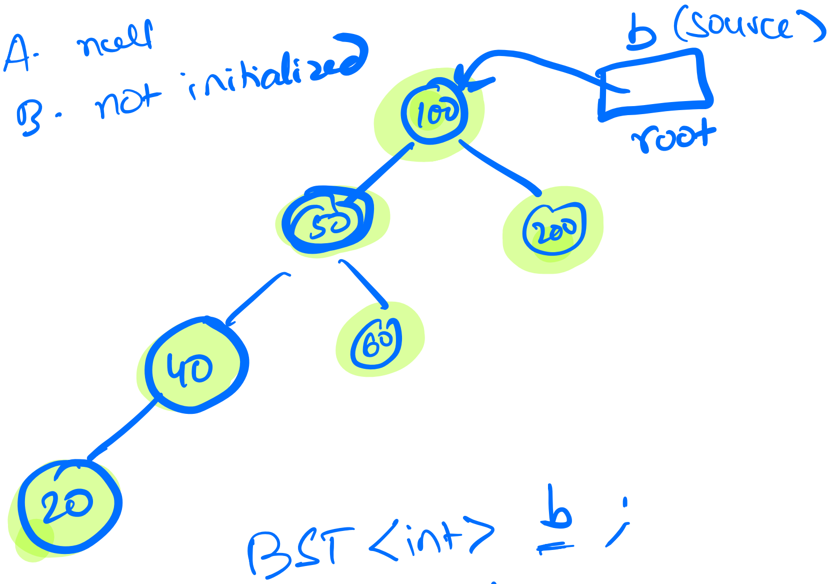
A. Very confident
B. Need to think!
C. I just don't know!

Big Four
~~constructor~~
destructor ✓
copy-constructor
copy-assignment operator

compiler will provide a default.



A. null
B. not initialized




```
BST<int> b;
// insert keys

BST <int> c(b);
```

```
template <class T>
class BST {

  BST(const BST<T> & source ) {
            root = nullptr;
            if ( ! source.root ) return;
            // Pre Order Insertion
  }
};
```

Data structure Comparison

	Insert	Search	Min	Max	Delete min	Delete max	Delete (any)
Sorted array vector							
Unsorted array vector							
Sorted linked list (assume access to both head and tail) list							
Unsorted linked list list							
Stack stack							
Queue queue deque							
BST (unbalanced)							
BST (balanced) set							
Min Heap priority que							
Max Heap "							

Data structure Comparison

	Insert	Search	Min	Max	Delete min	Delete max	Delete (any)
Sorted array	O(N)	O(logN)	O(1)	O(1)	O(N) if ascending order, else O(1)	O(1) if ascending, else O(N)	O(logN) to find, O(N) to delete
Unsorted array	O(1)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
Sorted linked list (assume access to both head and tail)	O(N)	O(N)	O(1)	O(1)	O(1)	O(1)	O(N) to find, O(1) to delete
Unsorted linked list	O(1)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N) to find, O(1) to delete
Stack	O(1) - only insert to top	Not supported	Not supported	Not supported	Not supported	Not supported	O(1) - Only the element on top of the stack
Queue	O(1) - only to the rear of the queue	Not supported	Not supported	Not supported	Not supported	Not supported	O(1) - only the element at the front of the queue
BST (unbalanced)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
BST (balanced)	O(logN)	O(logN)	O(logN)	O(logN)	O(logN)	O(logN)	O(logN)
Min Heap	O(logN)	Not supported	O(1)	Not supported	O(logN)	Not supported	O(logN)
Max Heap	O(logN)	Not supported	Not supported	O(1)	Not supported	O(logN)	O(logN)