Computer Science 2530
Spring 2020
Practice Exam 5 Answers

Answers to multiple-choice questions are marked with an arrow.

1. [MC] If $x = \log_2(60)$, then
   (a) $30 < x < 60$
   (b) $4 < x < 5$
   (c) $\rightarrow 5 < x < 6$
   (d) $6 < x < 7$
   (e) $7 < x < 8$

2. [MC] Only one of the following is true. Which one?
   (a) n^2 is $O(n)$
   (b) $\rightarrow n^2 + 5n$ is $O(n^2)$
   (c) n^3 is $O(n^2 + 3n + 10)$
   (d) n^3 is $O(n^2 + 10n)$
   (e) n is $O(\log(n))$.

3. [MC] To within a constant factor, how long does it take, in the worst case, to insert a value into a height-balanced binary search tree that has n values in it?
   (a) $\Theta(1)$
   (b) $\rightarrow \Theta(\log_2(n))$
   (c) $\Theta(n)$
   (d) $\Theta(n \log_2(n))$
   (e) $\Theta(n^2)$

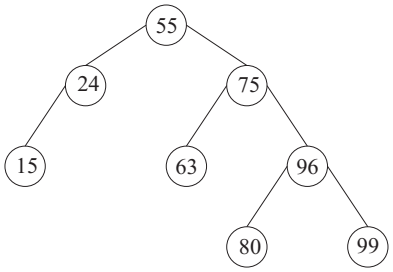
4. [MC] To within a constant factor, how long does it take, on the average, to look up a value in a hash table that has n values in it, assuming a high quality implementation?

   (a) $\rightarrow \Theta(1)$
   (b) $\Theta(\log_2(n))$
   (c) $\Theta(n)$
   (d) $\Theta(n \log_2(n))$
   (e) $\Theta(n^2)$

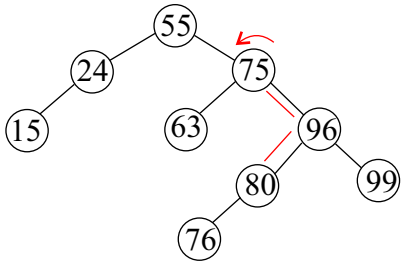
5. [MC] Suppose that you start with an empty height-balanced binary search tree. You successively insert n different values, into the binary search tree. How much time does it take, in the worst case, to do all of the insertions? (Give the cumulative time for all insertions, not just the time for one of them.)

   (a) $\Theta(1)$
   (b) $\Theta(\log_2(n))$
   (c) $\Theta(n)$
   (d) $\rightarrow \Theta(n \log_2(n))$
   (e) $\Theta(n^2)$

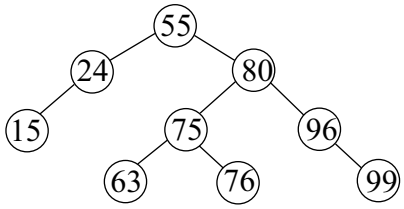
Consider the following binary search tree, T_0.



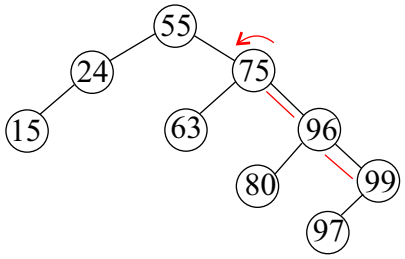
6. Show the tree that you get if you insert 76 into tree T_0, **using the basic algorithm that does not perform any rotations**.



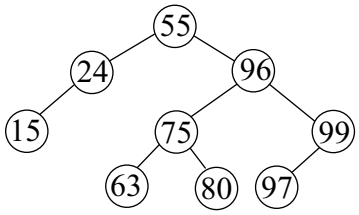
7. Show the tree that you get if you insert 76 into tree T_0, **using the algorithm that performs rotations to keep the tree height-balanced**. Start with the original tree T_0 that does not contain 76.



8. Show the tree that you get if you insert 97 into tree T_0, **using the basic algorithm that does not perform any rotations.** Start with the original tree T_0, which does not contain 76.



9. Show the tree that you get if you insert 97 into tree T_0, **using the algorithm that performs rotations to keep the tree height-balanced**. Start with the original tree T_0 that does not contain 76 or 97



Question 10 refers to the following type definition, which defines a type of nodes in binary trees.

```
struct Node
{
  int    item;
  Node*  left;
  Node*  right;

  Node(int i, Node* L, Node* R)
  {
    item  = i;
    left  = L;
    right = R;
  }
};
```

10. Assume that tree t contains only positive integers. Write a C++ definition of function largest(t), which returns the largest value in tree t. If t is an empty tree, then largest(t) should return 0. Tree t is *not* necessarily ordered like a binary search tree. You can use the **max** function. A heading is given.

```
int largest(const Node* t)
{
  if(t == NULL)
  {
    return 0;
  }
  else
  {
    return max(t->item, max(largest(t->left), largest(t->right)));
  }
}
```

11. Suppose that t is a tree that is not necessarily a binary search tree. You would like to define function member(x, t), which returns true if x occurs in at least one of the nodes int tree t, and returns false otherwise.

    Write a definition of member(x, t). A heading is given.

```
bool member(int x, const Node* t)
{
  if(t == NULL)
  {
    return false;
  }
  else
  {
    return x == t->item || member(x, t->left) || member(x, t->right);
  }
}
```

    Alternatively,

```
bool member(int x, const Node* t)
{
  if(t == NULL)
  {
    return false;
  }
  else if(x == t->item)
  {
    return true;
  }
  else if(member(x, t->left))
  {
    return true;
  }
  else
  {
    return member(x, t->right);
  }
}
```