

CS230 JEOPARDY: THE HOME VERSION

Asymptotics

[1] Consider the function $f(n) = 3n + 7$. List all of the following sets that do *not* contain f .

$$\begin{array}{lll} O(1) & \Theta(1) & \Omega(1) \\ O(n) & \Theta(n) & \Omega(n) \\ O(n^2) & \Theta(n^2) & \Omega(n^2) \end{array}$$

[2] List the following eight asymptotic complexity classes in order from smallest to largest:

$$\begin{array}{ll} \Theta(n) & \Theta(1) \\ \Theta(3^n) & \Theta(n * \log(n)) \\ \Theta(\log(n)) & \Theta(n^2) \\ \Theta(n^3) & \Theta(2^n) \end{array}$$

[3] Give the asymptotic complexity class (i.e., Θ notation) for the best algorithm we have studied for each of the following problems:

- a. Inserting n elements, one at a time, into a sorted array.
- b. Popping the top element off of a stack of n elements.
- c. Searching for an element in a balanced binary tree.
- c. Sorting a list of n elements.
- e. Cloning a mutable list with n elements.

[4] Match up each of the following algorithms with the recurrence equation that best characterizes it:

binary search	$T_1(n) = 1 + T_1(n - 1)$
linear search	$T_2(n) = 1 + 2 * T_2(n - 1)$
merge sort	$T_3(n) = n + T_3(n - 1)$
selection sort	$T_4(n) = 1 + T_4(n/2)$
towers of Hanoi	$T_5(n) = n + 2 * T_5(n/2)$

[5] List *all* of the following recurrence equations whose solution is $\Theta(n)$.

$$T_1(n) = 1 + T_1(n - 1)$$

$$T_2(n) = 1 + 2 * T_2(n - 1)$$

$$T_3(n) = n + T_3(n/2)$$

$$T_4(n) = n + T_4(n - 1)$$

$$T_5(n) = n + 2 * T_5(n/2)$$

$$T_6(n) = 1 + T_6(n/2)$$

$$T_7(n) = 1 + 2 * T_7(n/2)$$

Running Times

[1]

List *all* of the following sorting algorithms that have $\Theta(n * \log(n))$ worst-case running times:

- a. selection sort
- b. merge sort
- c. quick sort
- d. insertion sort
- e. heap sort (using a complete heap)
- f. tree sort (i.e., build a BST and list elements in in-order)

[2]

List *all* of the following that can be done in linear time in the worst case:

- a. Determining if x is in a length- n list.
- b. Sorting a list of n elements.
- c. Building a BST from a list of n elements.
- d. Listing in sorted order all elements of an n -element BST.
- e. Building a complete heap from a vector of n elements.
- f. Listing in sorted order all elements of an n -element complete heap.

[3]

List *all* of the following that can be done in logarithmic time in the worst case:

- a. Determining if x is in an array.
- b. Determining if x is in a sorted vector.
- c. Determining if x is in a balanced binary tree.
- d. Determining if x is in a balanced binary search tree.

- e. Inserting x into a complete heap.
- f. Deleting x from a 2-3 tree.

[4]

Consider the following method:

```
public static ObjectList inOrderList (ObjectTree t) {
    if (OT.isLeaf) {
        return OL.empty();
    } else {
        return OL.append(inOrderList(OT.left(t)),
            OL.prepend(OT.value(t),
                inOrderList(OT.right(t))));
    }
}
```

For each of the following assumptions, write a recurrence equation describes the worst running time of the method and give the asymptotic solution of the equation: (1) t is a balanced tree (2) t is an arbitrary tree.

[5]

(1) Write a recurrence equation for the worst case running time of the following method; (2) give the asymptotic solution of the equation; and (3) describe a simple modification to the method that dramatically improves its asymptotic running time.

```
public static int f (ObjectList L) {
    if (OL.isEmpty) {
        return 0;
    } else {
        return OL.length(L)*(OL.length(L) + f(OL.tail(L)));
    }
}
```

Gotchas

[1] What is the bug in the following Java method?

```
public int sum (Vector v) {
    int s = 0;
    for (int i = 0; i <= v.size(); i++) {
        s = s + v.get(i);
    }
    return s;
}
```

[2] What is the bug in the following Java method?

```
public static int toInt (Object x) {
    if (x instanceof Integer) {
        return x.intValue();
    } else {
        return 0;
    }
}
```

[3] What is the bug in the following Java class?

```
public class Location {

    private Point where;

    public Location (Point w) {
        Point where = w;
    }

    public int x () {
        return where.x;
    }

}
```

[4] Draw a binary tree that is *not* a binary search tree but for which the following buggy isBST method returns true.

```
// *INCORRECT* version of isBST
public static boolean isBST (IntTree t) {
    return
        IT.isLeaf(t)
        || (isBST(IT.left(t))
            && isBST(IT.right(t))
            && (IT.isLeaf(IT.left(t))
                || (IT.value(t) >= IT.value(IT.left(t))))
            && (IT.isLeaf(IT.right(t))
                || (IT.value(t) <= IT.value(IT.right(t))))));
}
```

[5] What output is displayed by the following Java method when invoked on the following vector of strings: ["a", "b", "c", "d", "e"]?

```
public static void removeAndDisplay (Vector v) {
    for (int i = 0; i < v.size(); i++) {
        System.out.println(v.remove(i));
    }
}
```

Lists

[1] An instance of `QueueTwoEndedMList` maintains pointers to both the first and last nodes of a mutable list. Why not just have a pointer to the first node?

[2] What is wrong with the following method for testing if an integer list is sorted?

```
public static boolean isSorted (IntList L) {
    return (IL.isEmpty(L)
        || ((IL.head(L) <= IL.head(IL.tail(L)))
            && (isSorted(IL.tail(L))));
    }
}
```

[3] Describe (1) one advantage of a mutable list over an immutable list *and* (2) one advantage of an immutable list over a mutable list.

[4] Flesh out the body of the following method for turning a *non-empty, non-cyclic, mutable* list into a cyclic list (i.e., a list whose last node's tail points to its first node).

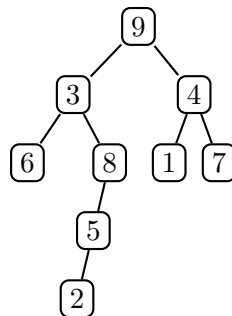
```
public static void cyclify (ObjectMLList L) {  
    // flesh this out;  
}
```

[5] Answer both of the following: (1) what is the running time of the following method on a list of length n ? (2) rewrite the body of the method so that it has the same behavior but has an asymptotically better running time.

```
public static IntList revDouble (IntList L) {  
    if (IL.isEmpty(L)) {  
        return L;  
    } else {  
        IL.postpend(revDouble(IL.tail(L)),  
                    2*IL.head(L));  
    }  
}
```

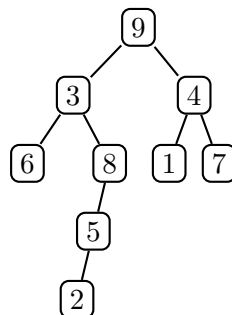
Trees

[1] List the values of all nodes in the following tree at which the heap condition is *not* satisfied.



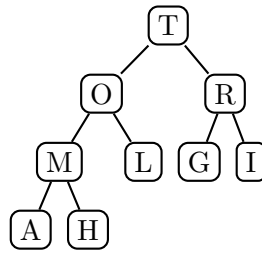
[2]

List the values of all nodes in the following tree at which the binary search tree condition is *not* satisfied.



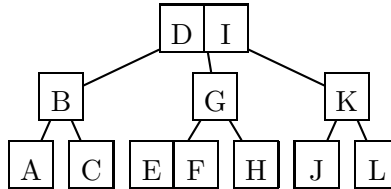
[3]

Draw the tree that results from dequeuing an element from the following complete heap:



[4]

Draw the tree that results from deleting J from the following 2-3 tree:



[5] What is the largest n such that all the distinct integers in $[1..n]$ can be arranged into a single binary tree that is both a BST and max complete heap?

Potpourri

[1] List all of the following collections for which a balanced binary search tree is an efficient representation:

- bag
- priority queue
- queue
- set
- stack
- table

[2]

Match up each of the following collections with the data structure that is most appropriate for representing it.

priority queue	2-3 tree
queue	non-cyclic linked list
file	complete heap
set	cyclic linked list
stack	string

[3]

For each of the following real-life collections, give the data structure that would most appropriately model the situation:

cities visited during a trip	bag
student grades	priority queue
emergency room waiting area	queue
pile of papers	set
shopping cart of grocery items	stack
supermarket checkout line	table

[4]

Describe the appearance and behavior of the following button:

```
final Button b = new Button("0");
b.addActionListener
    (new ActionListener() {
        public void actionPerformed (ActionEvent e) {
            b.setLabel(b.getLabel() + "1");
        }
    });
```

[5]

Complete the following definition of a class that enumerates the characters of a string:

```
public class StringChars implements Enumeration {
    private String s; // the string
    private int i; // current index

    public StringChars (String str) {s = str; i = 0;}

    public boolean hasMoreElements() {...}

    public Object nextElement() {...}
}
```