Tristan Rice (25886145, q7w9a)
2015-11-25
CPSC 221: Written Assignment 2

No partner

Problem 1.
a)
T(1) = c1
T(n) = c2 + T(n/2) + T(n/2)
T(n) = 3 *c2 + 4 *T(n/4)
T(n) = 7 *c2 + 8 *T(n/8)
T(n) = 15*c2 + 16*T(n/16)
T(n) = 31*c2 + 32*T(n/32)
T(n) = c2*(n-1) + c1*n
It does n-1 comparisons, but calls findMaxHelper with a single element n times.

b)
Proof by induction.
Base case: n = 1, left = right
If n = 1, there is only one element which is thus the max.

Inductive Hypothesis: findMaxHelper(A, left, right) is correct for all values of 0 <= left <=right < A.length.

Inductive step:
Let findMaxHelper(A, a, b) returns the maximum value of a for all values (a, b) = {(left, (left+right)/2), (left+right)/2, right)}.
The induction hypothesis tells us that the two sub steps of findMaxHelper, correctly compute the max.
findMaxHelper(A, left, right) = max(findMaxHelper(A, left, (left+right)/2), findMaxHelper(A, (left+right)/2, right)
By definition of max, this will return the maximum value between the left half and the right half.
Thus, by induction, findMaxHelper(A, left, right) always returns the maximum value of the array.

Problem 2.
a) Invariant: x[i] <= x[iom]

Base case: loop invariant x[i] >= x[iom] holds on loop entry since iom = i

Inductive hypothesis: On any iteration of the loop, x[i] >= x[iom]
Inductive step:
Case 1: If x[j] < x[iom], update iom = j
Case 2: x[j] >= x[iom], do nothing
Since x[iom] always gets smaller or doesn't change, x[i] >= x[iom] holds

b) Invariant: x[i] >= x[i-1], i <= n - 1
Base case: loop invariant i <= n -1 holds on loop entry since i = 0
Inductive hypothesis: On any iteration of the loop i <= n - 1
Inductive step:
Case 1: if i < n-1 then i = i + 1
Case 2: if i == n-1 then do nothing
Thus, the inductive hypothesis holds.

c)
The first invariant shows that the inner loop will find the smallest value in the remaining array x[iom].
The second invariant shows that the loop will be called for every i < n-1.
Thus, every element--except the last--will be swapped with the smallest value past that element. The last will then have the largest value. Therefore, ss sorts the first n elements of array x.

Problem 3.
A room contains 6 computers. Each computer is directly connected to 0 or more of the other computers in the room.

Hypothesis: There are at least two computers in the room that are directly connected to the same number of other computers.

The room contains 6 computers. This means that each computer can have 0-5 connections.
When you create a connection, the connection count of both computers increases by one.

If a computer is connected to all other machines (5 connections), there are no machines with 0 connections. If a computer is connected to 0 machines, there can't be a computer connected to all of the machines (5 connections).

Case reduction: if there are any computers with 0 connections, there can't be a computer with 5 connections and vice versa.

Assume there are no computers with 5 connections. Only 0-4 connections, or 1-5 connections.
Thus there are only 5 possible states and 6 computers. Therefore, there must be two computers with the same number of connections.

Problem 4.
There are 8 buckets. Four colors of balls. Picking 4 buckets should return at least one of every ball.
If you make the number of balls 8-3=5, there are only 3 buckets without that color ball and thus impossible to pick 4 buckets without that color.
4 colors * 5 buckets * 1 ball each = 20 balls

Example buckets (5 of each color)
0 RGBY
1 RGBY
2 RGBY
3 RGBY
4 RGBY
5 <none>
6 <none>
7 <none>
We can look at a single ball color at a time without loss of generality. In this above case, there are no 4 buckets we can pick that would not have every color. Order is also irrelevant.

Example buckets (4 of each color)
0 RGBY
1 RGBY
2 RGBY
3 RGBY
4 <none>
5 <none>
6 <none>
7 <none>
When there are 4 of each color, it is clear it is possible to pick 4 buckets without every color (4-7).

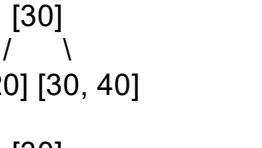
Problem 5.
Draw the Dictionary data structure obtained after inserting:
10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 93, 97, 96
one after the other into the following initially empty structures.

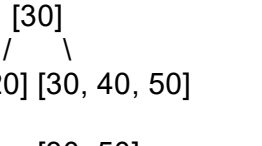
Note: If a node isn't completely filled, the empty spaces are omitted.

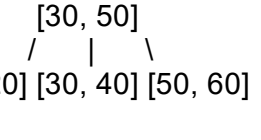
a) hash(k) = k mod 11
0: 110
1: 100
2: 90
3: 80
4: 70
5: 93 -> 60
6: 50
7: 40
8: 96 -> 30
9: 97 -> 20
10: 120 -> 10

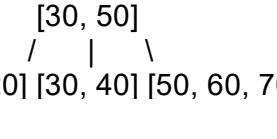
b) hash1(k) = k mod 23, hash2(k) = 13 - (k mod 13)
0:
1: 70, probed 1
2:
3:
4: 50, probed 4
5: 120, probed 5
6:
7: 30, probed 7
8: 100, probed 8
9:
10: 10, probed 10
11: 80, probed 11
12: 93 (1 + 1*11), probed 1, 11
13: 97 (4 + 4*8) mod 23, probed 4, 12, 20, 5, 13
14: 60, probed 14
15:
16:
17: 40, probed 17
18: 110, probed 18
19: 97 (5 + 2*7), probed 5, 12, 19
20: 20, probed 20
21: 90, probed 21
22:

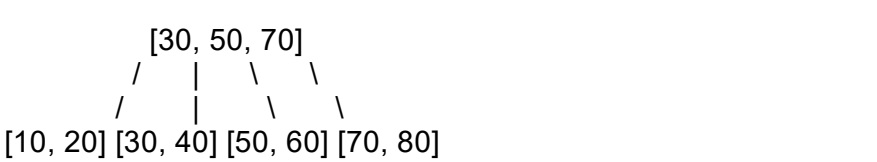
c) M = 5, L = 3
initial: null
[10, 20, 30]

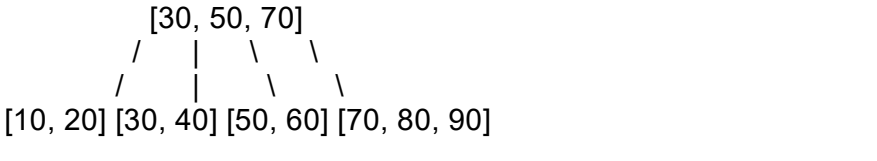


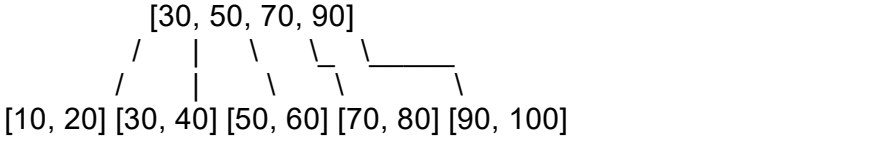


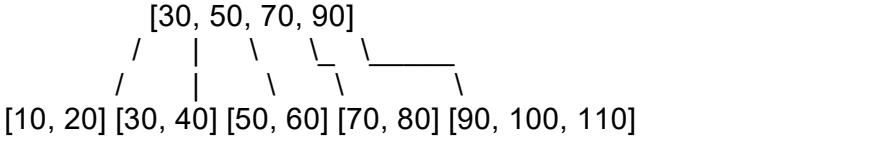


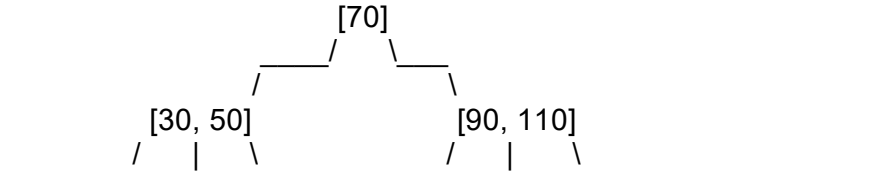


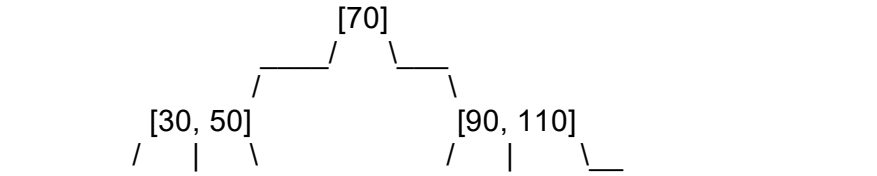


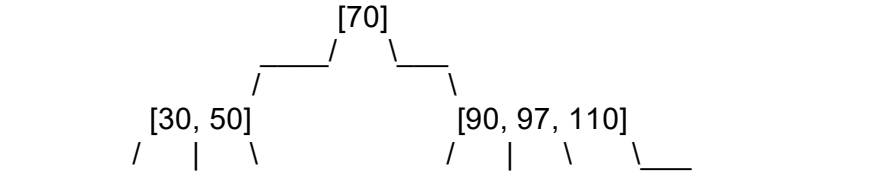


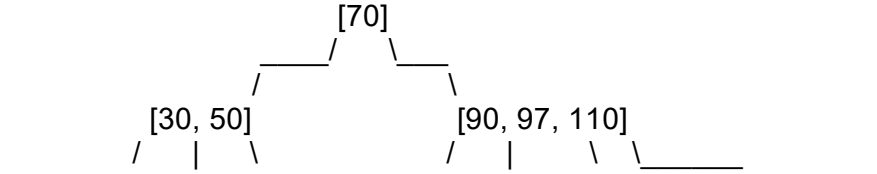












d) M = 100, L = 20. initial: null
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 93, 97, 96]
Since L = 20, and we're only inserting 15 nodes it never splits.

Problem 6. Special AVL Tree
Assuming 64 bit machine. Standard AVL tree node then has two 8-byte pointers.
Space nodes = (16 + c)*n
Space array = c*n

a)
N(0) = 1
N(1) = 2
N(2) = 4
N(3) = 7
N(4) = 12
N(h) = 1 + N(h-1) + N(h-2)

b)
N(h) = 1 + 1 + N(h-2) + N(h-3) + 1 + N(h-3) + N(h-4)
Fib(0) = 1
Fib(1) = 1
Fib(2) = 2
Fib(3) = 3
Fib(4) = 5
Fib(6) = 8
Fib(7) = 13
N(h) = Fib(h+3) - 1

c)
AVL height (n)= 1.44*lg(n)
Array size (h) = 2^h - 1

With pointers space: (base + 16)*n

Without pointers:
In the best case, an perfectly filled AVL tree in an array saves a fair amount of space since it saves 16 bytes per node for the 64 bit pointers to the other nodes.
Space: base * n

In the worst case, $2^{1.44 \cdot \lg n} = 1.44 \cdot n$, Thus, it takes 1.44 times the space.
Space: base * 1.44 * n

It doesn't always save space. It generally depends on how well the tree is formed and how big the base data is. For instance, if the base data is a single pointer to something else, it takes 8 * 1.44 * n = 12 * n vs. (8+16)*n = 24 * n. Once the base data is larger than 5 bytes, it isn't always better.