**Homework Assignment #4**

**General instructions:**

A. Submission is in pairs. On each page write down your names and ID numbers.

B. The solution must be submitted in the submission box on the Moodle site of one of the submitters. In the other submission box write the name and ID number of the box with the solutions.

C. Submit the solutions as a pdf file files only.

D. The exercise must be submitted on time. The submission boxes close at 23:55 on the same day as the deadline for submission. It will not be possible to submit late except for special cases according to the student regulations, such as birth and reserve duty.

E. Each exercise has a number of questions. Usually each exercise covers more than one topic. You do not have to wait until you learn the material to start solving, but start solving those questions that are already possible and do not wait until the last minute. Of course, do not have to wait until the last day to submit your work.

**Note : You only need to submit questions 3, 5, 6 and 8**

**Question 1**

1. Build an AVL tree from the following values. Draw the tree after each insertion and write which rotation you used when a rotation was needed.

28, 42, 17, 8, 12, 14, 18, 32, 39, 7.

1. Delete from the above tree the following values. Draw the tree after each deletion and write which rotation you used when a rotation was needed.

17, 12, 42, 32

**Question 2**

What is the minimal amount of nodes in a AVL tree of height 5?

1. 12
2. 20
3. 21
4. 63

**Question 3**

Write a linear algorithm (that is, that runs in Θ(n)) that builds an AVL tree from an array sorted in non-descending order. [146689]

We are using suggestion A Dealing with duplicates. An AVL tree is a BST containing unique values and that for every node x in the tree:

The difference between the height of the left tree and the height of the right tree is no greater than 1. That is, either 1 or 0 or -1.

**AVL(Arr, n)** // Array passed through, n holds the index

if (Arr[n] = NULL) // last condition

return

if (n = 1) // first condition

T = creatTree()

insert(T, Arr[n])

return AVL(Arr, n+1)

else

insert(T, Arr[n])

set(x) // x is root(T)

rotate(T)

return AVL(Arr, n+1)

**rotate(T)**

if (T = NULL)

return

else

if ( BF(T) > 1 OR BF(T) < -1)

LL(T)

return

return rotate(right(T))

**LL(p)**

B = p

A = left(B)

AR = right(A)

parent(A) = parent(B)

right(A) = B

parent(B) = A

left(B) = AR

parent(AR) = B

Return(A)

**insert ( T, x)**

p = createNode(x,null,null)

if root(T) == null

root(T) = p

else

p1 = root (T)

while (p1 != null)

p2 = p1

if key(p) < key(p2)

p1 = left(p1)

else p1 = right(p1)

parent(p) = p2 // if there is an attribute

if key(p) < key(p2)

left(p2) = p

else right(p2) = p

**Question 4**

1. What is the maximum number of possible nodes for an AVL tree of height h? Explain.
2. Given a BST, *T*, can you make it an AVL by using rotations only? Explain.

**Question 5**

a.Write an efficient algorithm that accepts as input an array of size *n*, and checks if it is a min-heap.

**isMinHeap (Arr, i) // i is the index in the array starting with 1**

if( Arr(i) == null ) return true

if( i\*2 < length(Arr) )

if( Arr(i) > Arr(i\*2) ) return false

if( i\*2 + 1 < length(Arr) )

if( Arr(i) > Arr(i\*2 + 1) ) return false

return ( isMinHeap (Arr, i\*2) && isHeap(Arr(i\*2 + 1)) )

1 2 3 4 5

1. 14 7 18 28

b. In the worst case, what is the run-time complexity of your algorithm. Explain.

log(n), In the worst case it will run through all of the numbers in the array checking that they are in a min heap order. This would mean that the numbers were in mean hip order.

**Question 6**

Write an algorithm that accepts a max-heap containing *n* elements, and index *i* of an element in the heap, and a natural number *m*. The algorithm should add *m* to all the values in the sub-tree whose root is *i*, and returns the updated heap.

Your algorithm has to run in O(n) time. // how to put a tree in an array… breadth algo or can we use an array?

addN(T, m, i) // T is a maxheap tree

if (isEmpty(T))

return

else(count >= i)

key(count) += m

**Question 7**

Given a max-heap of size *n*, implemented using an array, containing distinct values.

1. In which indexes (indices , for non-americans 😊) of the array can the minimum value be found? Prove your answer.
2. In which indexes of the array can the second smallest value be found? Prove your answer.

**Question 8**

1. Describe how can you implement the insertion and deletion functions of a priority queue using two stacks?

Given two stacks, each element having a key representing its priority – the highest number being the most urgent.

**Insertion** is the same as regular queue, which would be implemented by popping all of the elements into the second stack and inserting the desired element into the first. Finally popping them all back.

**Deletion** is done by popping the top element and saving it in a variable called current. Then we pop each element off and compare the value to current. **If** the value is **greater**, that element becomes the current and the previous value is pushed into the second stack. **If** the value is **equal**, we will have a bool condition, called duplicate that will only now be true. In that case it will keep going through the stack looking for a greater number and if it finds one, duplicate will be flagged false. If the duplicate bool condition is true in the end, we will pop and compare each value from the second stack to the current before pushing it into the first stack. If the value of the popped element patches the current we delete it. Otherwise, we pop all of the values from the second stack to the first and delete the current. We do this process until the first stack is empty if we are looking to go through all of the elements. If we just want one delete we would do the process once.

1. Analyze the run-time complexity of the above functions.

Theta(n^2) Giant loop to wait for all the elements to be emptied from first stack. 2 while loops inside to pop and push elements from one stack to the other.

**Question 9**

Given: A max-heap containing *n* ( n > 248) distinct elements. If we transverse the heap in preorder transversal, which of the following assumptions are correct?

1. The minimal value can be found in the first place of the preorder transversal.
2. The minimal value will also be in the last places in the preorder transversal.
3. The minimal value can be found in the place in the preorder transversal.
4. If the heap is completely full including the lowest level, then the minimal value will be exactly in the place in the preorder transversal.