Problem 1.  
Starting with an empty AVL tree insert the following keys in the order shown (leftmost key first). Show the state of the tree (a) after the key is inserted and before a rebalancing takes place, and (b) after the key is inserted and a rebalancing operation caused by the insertion is completed.

20, 10, 5, 7, 8, 15, 17, 22, 21

Then delete 8, 5, 17.

Problem 2.  
Suppose that we have numbers between 1 and 1000 in a binary search tree and want to search for number 363. Which ones of the following sequences (one or more) could NOT be the sequence of nodes examined?

i. 500, 700, 640, 670, 660, 641, 363.
ii. 5, 800, 700, 600, 200, 470, 363.
iii. 925, 202, 911, 200, 900, 245, 363.

Justify your answer.

Problem 3.  
Argue that since sorting $n$ keys takes $\Omega(n \lg n)$ time in the worst case in the comparison model, any comparison-based algorithm for constructing a binary search tree from an arbitrary list of $n$ elements takes $\Omega(n \lg n)$ time in the worst case.

Problem 4.  
Give a data structure $S$ that implements a MIN priority queue (operations ExtractMin and Insert are implemented efficiently) and a MAX priority queue (operations ExtractMax, Insert) efficiently and simultaneously (i.e. in worst case $O(\lg n)$ time). You therefore need to implement efficiently three operations: Insert($x$), ExtractMin, and ExtractMax.

A complete solution first describes the data structure $S$, and then presents the algorithms for the three operations. Finally an analysis of the worst case running time is performed and shown to be $O(\lg n)$, where $n$ is the number of elements in the data structure before the operation is performed.

Note: You are allowed to use without proof all operations that have been introduced in this class. If for example you wish to use MaxHeapify, you do not need to present its code, you just call it using the class described interface MaxHeapify(A,i).

Problem 5.  
You want to sort the $n$ keys stored into an AVL tree. How fast can you do it? Explain.

Problem 6.  
(Precursor to Problem 5.) You have $n$ keys and you want to use an AVL tree to sort them. Do so and explain so.

Problem 7.  
We have $n$ keys that take $k$ distinct values (integer or not). Give an algorithm that sorts those keys in $O(n \lg k)$ time. (Assume $n \gg k$.)

Problem 8.  
You are given a set $A$ of $n$ numbers (real, integer, positive, or negative) out of which only $m$ are distinct where $m \leq \sqrt{n}$. Find the $m$ distinct values and print them in sorted order. For example if $A$ looks like

$\{\pi, e, 5, 10^7, \pi, 10^7, e, 5, 5, 10^7, e, \pi, 5, 10^7, e, \pi, 10^7\}$

we have $n = 16$ but $m = 4$. Your algorithm should be faster than an $O(n \lg n)$ generic algorithm for a variety of choices of $m$. For example, if $m = O(1)$ your algorithms should be $O(n)$ which is faster than $O(n \lg n)$. Similarly if $m = O(\lg n)$.

Hint. Give an $O(n \lg m)$ algorithm.

Additional question. How would you sort the $n$ keys in the same asymptotic time?
Problem 9.
You are given six polynomials $f_1, \ldots, f_6$ of degrees $1, 2, 3, 1, 4, 5$ respectively. We are interested in finding the product $f = f_1 f_2 f_3 f_4 f_5 f_6$. Assume that the cost of multiplying two polynomials of degree $a$ and $b$ is $a \cdot b$. Find a schedule for multiplying the six polynomials that is of the lowest possible total cost.

Problem 10.

a. What is the largest $k$ such that if you can multiply $3 \times 3$ matrices using $k$ multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{k7})$? What would the running time of this algorithm be?

b. V. Pan has discovered a way of multiplying $68 \times 68$ matrices using 132464 multiplications, $70 \times 70$ matrices using 143640 multiplications, $72 \times 72$ matrices using 155424 multiplications. Which method yields the best asymptotic running time when used in a divide and conquer matrix multiplication algorithm? How does it compare to Strassen’s algorithm?

Problem 11.
Suppose that we insert $n$ keys into a hash table of size $m$ using open addressing and uniform hashing. Let $p(n, m)$ be the probability that no collisions occur. Show that $p(n, m) \leq \exp(-n(n-1)/(2m))$. Argue that when $n > \sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

Hint: Use induction... Also use $\exp(x) \geq 1 + x$ for any real $x$. Note that $\exp(x) = e^x$.

Problem 12.
Suppose that we are given a key $k$ to search for in a hash table with positions $0, \ldots, m-1$, and suppose that we have a hash function $h$ mapping the key space into the set $\{0, \ldots, m-1\}$. The search scheme is as follows.

1. Compute the value $i = h(k)$ and set $j = 0$.
2. Probe in position $i$ for the desired $k$. If you find it, or if this position is empty, terminate the search.
3. Set $j = (j+1) \mod m$ and $i = (i+j) \mod m$ and return to step 2.

Assume that $m$ is a power of 2.

a. Show that this scheme is an instance of the general quadratic probing scheme by exhibiting the appropriate constant $c_1, c_2$ for $h(k, i) = (h(k) + c_1 i + c_2 i^2) \mod m$.

b. Prove that this algorithm examines every table position in the worst case.
Problem 13.

Given two strings \( a = a_0a_1 \ldots a_k \) and \( b = b_0b_1 \ldots b_m \), where each \( a_i \) and \( b_j \) belongs to some ordered set of characters (e.g. English alphabet, binary set) we say that string \( a \) is lexicographically less than string \( b \) if either

1. there exists an integer \( j \), where \( 0 \leq j \leq \min(k, m) \), such that \( a_i = b_i \) for all \( i = 0, \ldots, j-1 \) and \( a_j < b_j \), or
2. \( k < m \) and \( a_i = b_i \) for all \( i = 0, \ldots, k \).

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
</tr>
</tbody>
</table>
/-------\
0 / 1
/ \ 
|--|--| |----|
|Red | |Black|
------ -----
Figure: A binary radix tree.

For example, 010 is lexicographically less than 0100 which is less than 0101. A binary radix tree is a tree built out of a collection of binary strings and one is depicted in the attached Figure. In a binary radix tree an edge leading to a left child can be labeled 0 and to a right child can be labeled 1. A node is either black or red; a black node indicates that the string (using the implied edge labels) corresponding to the path from the root to the black node is NOT in the tree, whereas a red node indicates that the string is in the tree. When searching for \( a \) in such a tree we traverse a left child at depth \( i \) if \( a_i = 0 \), and a right child if \( a_i = 1 \). If a red node is reached, \( a \) is in the tree, otherwise it is not. Searching for 101 leads to a black node and for 1011 to a red node: the former string does not exist, the latter does. Let \( A \) be a set of distinct binary strings whose total length sum is \( n \). Show how to use a radix tree to sort \( A \) in lexicographic order in \( \Theta(n) \) time. Sorting the tree above with \( n = 13 \) would give the strings 0, 011, 10, 100, 1011 in lexicographic order.

Problem 14.

What’s the best way to find the product \( M_0 \times M_1 \times M_2 \times M_3 \) where the matrices are of dimension \( 100 \times 200 \), \( 200 \times 500 \), \( 500 \times 10 \), and \( 10 \times 1000 \) respectively? (In a prior problem, you had to deal with polynomials. The polynomials have now become matrices.)
Problem 1.

Insertion:

\[
\begin{array}{ccccccc}
20 & 10 & 5 & 7 & 8 & 10/0 & 10/+1 \\
\hline
20 & 10 & 5 & 7 & 8 & \text{-----> 20 } & \text{-----> 20/+1 } \\
& & & & & (a,b,c)= & \\
& & & & & 10/0 & 10/+1 \\
& & & & & (x,y,z)= & \\
& & & & & 10/0 & y \\
& & & & & 10/+1 & x \\
& & & & & 10/0 & 5 \\
& & & & & 10/+1 & 20 \\
& & & & & 10/0 & 5 \\
& & & & & 10/+1 & 20 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
10/+2 & 10/+1 & 15 & 17 & 10/-1 & 10/-2 & 10/-1 \\
\hline
10/+2 & 10/+1 & 15 & 17 & 10/-1 & 10/-2 & 10/-1 \\
& & & & & (a,b,c)= & \\
& & & & & 10/+2 & z \\
& & & & & 10/0 & v \\
& & & & & 10/+1 & 17 \\
& & & & & 10/0 & v \\
& & & & & 10/+1 & 17 \\
\end{array}
\]

Deletion:

\[
\begin{array}{ccccccc}
7 & 17 & 8 & 5 & z & 10/-2 \\
\hline
7 & 17 & 8 & 5 & z & 10/-2 \\
& & & & & (a,b,c)= & \\
& & & & & 7 & v \\
& & & & & 17/-1 & y \\
& & & & & 5 & 15 \\
& & & & & 20/-1 & 20 \\
& & & & & 15 & 15 \\
& & & & & 21/0 & 21 \\
& & & & & 22 & 22 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
17 & 20 & 10 & 15 & 22 & 17 \\
\hline
17 & 20 & 10 & 15 & 22 & 17 \\
& & & & & (a,b,c)= & \\
& & & & & 20 & 22 \\
& & & & & 20 & 22 \\
& & & & & (y,x,z) & \\
& & & & & 17/-1 & q \\
\end{array}
\]

Problem 2.

i. It can’t be. 363 is in the right subtree of 640 which is impossible since in a BST 363 must be then greater than 640 which is obviously not the case.

ii. Looks ok.

iii. It can’t be. 200 is in the right subtree of 212 but it is not greater than 200!. This cannot be the case for a BST.

iv. Looks ok.

Problem 3.

Suppose we can construct a binary search tree on \( n \) nodes in \( T \) time. Then we can run an inorder traversal of the BST in \( \Theta(n) \) time and thus get in total time of \( T + \Theta(n) \) the nodes in sorted form, since we know that an inorder traversal of a BST produces a sorted sequence of the keys stored in the tree.

If \( T = o(n \log n) \) this means we can sort \( n \) keys in \( o(n \log n) + \Theta(n) = o(n \log n) \) time: first build a binary search tree and then perform an inorder traversal of its nodes. However we know from the lower bound for sorting that comparison-based sorting can not be performed in \( o(n \log n) \) time, i.e. it requires \( \Omega(n \log n) \). Therefore we can not have \( T = o(n \log n) \), i.e. we must have \( T = \Omega(n \log n) \).

Thus any algorithm for creating a BST of \( n \) keys must have running time (at least) \( T = \Omega(n \log n) \). This bound is tight for AVL trees (and also red-black trees not covered in class but material related to them is available in the textbook) since the time an AVLTreeInsert requires to insert \( n \) keys in an empty tree is also tight \( \Theta(n \log n) \).
Problem 4.

Use an AVL tree as the data structure. Insert is an AVL tree insert, ExtractMin, ExtractMax, are implemented by first issuing a FindMin or FindMax and then a Delete on the found min or max key. Insert, Delete on AVL trees are in the notes/textbook, and so are FindMin, FindMax. Worst case performance of all these operations on an AVL tree of \( n \) nodes is \( O(\lg n) \).

Problem 5.

Perform an inorder traversal in \( O(n) \) time. The keys are sorted.

Problem 6.

Insert the keys into an AVL tree and then run the algorithm of Problem 6. The latter step is \( O(n \lg n) \). The former step is \( O(\lg n) \) per key or \( O(n \lg n) \) total. Total running time is \( O(n \lg n) \) as well.

Problem 7.

Radix-sort does not (necessarily) work. We don’t know the range of the key values. Let alone the fact that the keys might not be integer. Build an AVL tree and for each node maintain also a counter (how many keys have that specific value). Maintaining this AVL is \( O(\lg k) \) per operation as the number of distinct values is \( k \). Thus run the algorithm in Problem 7. In addition, when you do the inorder traversal and you are about to print the value of node \( x \) you print that value as many times as indicated by the counter. (See example below.)

<table>
<thead>
<tr>
<th>Value (Count)</th>
<th>Inorder traversal Sorts</th>
</tr>
</thead>
<tbody>
<tr>
<td>10(3)</td>
<td>..................................-&gt; 5 10 10 12 12 12</td>
</tr>
<tr>
<td>5(1)</td>
<td>12(4)</td>
</tr>
</tbody>
</table>

Problem 8.

No solution provided. Think about binary search trees. Great as an interview question though.

Problem 9.

Multiply them any way you like. The final result is a polynomial of degree 16 and the total cost is 100. Note that 100 is the sum of all products \( d_i \cdot d_j \) for every combination of two polynomials \( f_i \) and \( f_j \) among the six and \( d_i \) and \( d_j \) are respectively the degrees of \( f_i \) and \( f_j \). In our example we have \( 6(6-1)/2 \) such pairs of polynomials.

\[
100 = d_1(d_2 + d_3 + d_4 + d_5 + d_6) + d_2(d_3 + d_4 + d_5 + d_6) + d_3(d_4 + d_5 + d_6) + d_4(d_5 + d_6) + d_5d_6
\]

\[
= 1(2 + 3 + 1 + 4 + 5) + 2(3 + 1 + 4 + 5) + 3(1 + 4 + 5) + 1(4 + 5) + 4 \cdot 5
\]

\[
= 15 + 26 + 30 + 9 + 20
\]

This sum is independent of the order of the evaluation of the products; an easy proof is by induction (eg. with reference to the 6 polynomials above, the product of five of them \( f_2, \ldots, f_6 \) takes time equal to the 4 summands except the first one \( d_1(d_2 + d_3 + d_4 + d_5 + d_6) \); in the last step of an inductive proof the polynomial \( f_1 \) whose degree is \( d_1 \) is multiplied by the result \( f_2 \ldots f_6 \) whose degree is \( d_2 + \ldots + d_6 \) and the cost of that multiplication is the missing/absent (first) term. Permute the polynomial and the result remains the same except that the pairs of the products \( d_i d_j \) get rearranged.)

Note. One of the reasons that this is so is that polynomials satisfy commutativity therefore \( f_1 f_2 = f_2 f_1 \). What happens if instead of polynomials you have matrices and the operation is matrix multiplication? The cost then depends on the specific schedule that you choose. [Check Subject 8 (to appear), pp. 4-7, on how to determine the optimal schedule using dynamic programming].
Problem 10.

a. Strassen's algorithm uses the pattern of multiplication for $2 \times 2$ matrices and divide-and-conquer by splitting a matrix multiplication of $n \times n$ matrices into a matrix multiplication of $n/2 \times n/2$ matrices to derive a time bound for matrix multiplication given by the recurrence $T(n) = 7T(n/2) + O(n^2)$. In such a recurrence the 7 indicates the number of scalar multiplications in multiplying $2 \times 2$ matrices or the number of $n/2 \times n/2$ matrix multiplications in the recursive (divide-and-conquer) decomposition. Note that Strassen's method on $2 \times 2$ matrices can not be improved. We cannot multiply $2 \times 2$ matrices with say 6 (scalar) multiplications.

Now if we can multiply $3 \times 3$ matrices and $k$ is the number of scalar multiplications required and we use a recursive decomposition of an $n \times n$ matrix into $n/3 \times n/3$ matrices we get a recurrence for the running time of this method of

$$T(n) = kT(n/3) + O(n^2).$$

The solution of this recurrence (master method) yields $T(n) = \Theta(n^{\log_3 k})$. Therefore if we compare the exponent $\log_3 k$ to $\log 7$, then for $k = 21$ we have $\log_3 21 \approx 2.771 < 2.8073 = \log 7$.

For that value of $k = 21$ or smaller a by 3 decomposition beats Strassen's method.

b. This is similar to part (a). If we decompose not by-2 or by-3, but by-$t$, where $t$ now is 68, 70, 72 we obtain that the running time for such a decomposition is given by the recurrence,

$$T(n) = kT(n/t) + O(n^2),$$

with an asymptotically tight solution given by $T(n) = \Theta(n^{\log_t k})$. Therefore if we compare the exponent $\log_t k$ to $\log 7$, then for $t = 68$, $\log_{68} 132464 = 2.795128$, $t = 70$, $\log_{70} 143640 = 2.795122$, $t = 72$, $\log_{72} 155424 = 2.795147$. All three of them beat Strassen's exponent. The best of the three is the $t = 70$ split. Calculations used the Unix bc calculator and hopefully are correct!

Problem 11.

Consider $p(n+1,m)$ and argue similarly to the derivation for Insertion done in class. For the $n+1$st key not to cause any collision in a hash table with $n$ keys inserted with no collisions, it must hit an empty slot an event that occurs with probability $1 - n/m$. Therefore $p(n+1,m) = (1 - n/m)p(n,m)$, since the probability that the insertion of say $n+1$ key cause no collision is the probability that the insertion of the $n+1$st key causes no collisions and the prior insertion of the remaining $n$ keys caused no collisions; the latter is $p(n,m)$. Therefore

$$p(n+1,m) = (1 - n/m)p(n,m) = (1 - n/m)(1 - (n - 1)/m)p(n - 1,m) = (1 - n/m)(1 - (n - 1)/m)\ldots(1 - 1/m)(1 - 0/m)p(0,m) = (1 - n/m)(1 - (n - 1)/m)\ldots(1 - 1/m)$$

Since $e^x \geq 1 + x$, setting $x = -i/m$ we obtain that $1 - i/m \leq e^{-i/m}$. Therefore

$$p(n+1,m) = (1 - n/m)(1 - (n - 1)/m)\ldots(1 - 1/m) \leq \exp(-n/m)\exp(-(n-1)/m)\ldots\exp(-1/m) \leq \exp(-(1 + 2 + \ldots + n)/m) = \exp(-n(n+1)/2m)$$

Therefore $p(n,m) \leq \exp(-(n-1)n/(2m))$ as required. Consider $n \gg \sqrt{m}$, the $n(n-1)/2m \gg 1$, and thus $p(n,m)$ is upper bounded by $\exp(-A)$, where $A$ is large and positive. However $\exp(-A)$ goes to zero and thus $p(n,m)$ goes to zero as well.

This problem is also known as the Birthday problem. Let $m = 365$ (forget about leap years). Suppose you have $n$ people in a room. What is the probability of having no two with the same birthday? For what value of $n$ is the probability of having two people with the same birthday close to 1? One can see that for $n = \sqrt{m}$ collisions start to show up with a significant (non negligible) probability, i.e. people exist with the same birthday.

Problem 12.

a. The hashing scheme searches entries $h(k), h(k) + 1, h(k) + 2, h(k) + 1 + 2, h(k) + 1 + 2 + 3, \ldots, h(k) + 1 + 2 + \ldots + j \mod m$, where $m$ is a power of two.

The value of $i$ after $j$ iterations (i.e. after $j$ probes) is $i = h(k,j) = h(k) + 1 + 2 + 3 + \ldots + j = h(k) + j(j + 1)/2 = h(k) + 1/2j + 1/2j^2 \mod m$, i.e. we indeed have a quadratic probing scheme with $c_1 = c_2 = 1/2$ and hash function $h(k,j)$.

b. (Proof by contradiction) Suppose that the algorithm does not examine every table position in the $m$ probes $h(k,j)$, $j = 0, \ldots, m - 1$. This means that at least one table entry is visited at least twice (and at least one is not visited at all) i.e. there exist $0 \leq l, r \leq m - 1$ such that $h(k,l) = h(k,r)$, $l \neq r$ and therefore $h(k,l) - h(k,r) \equiv 0 \pmod{m}$. Therefore
Without loss of generality, we assume \( l < r \). We shall then show that for \( l, r \) as above we have that \( h(k, l) \neq h(k, r) \), and therefore \( h(k, l) - h(k, r) \neq 0 \pmod{m} \), i.e. a contradiction. It is true that \( h(k, r) - h(k, l) = r(r + 1)/2 - l(l + 1)/2 = (r - l)(r + l + 1)/2 \equiv 0 \pmod{m} \).

Consider \((r - l)(r + l + 1)/2 \equiv 0 \pmod{m}\) or in the other words. \((r - l)(r + l + 1)/2 = Am\) for some integer \( A \). For this to be true, \( m \) or a power of two must divide \( r - l \) or \( r + l + 1 \) or both. We distinguish two major cases.

1. Let \( r - l \) be even (i.e. \( r, l \) are both odd or both even). Then, \( r + l + 1 \) is odd and it is not divisible by \( m \) or 2 as both \( m \) (a power of 2) and 2 are even and \( r + l + 1 \) was shown to be odd. On the other hand, \( r - l \leq m - 1 \) as \( r, l \leq m - 1 < m \). Then \((r - l)/2 < m\) and therefore \( m \) cannot divide \((r - l)/2\) either. This means that \((r - l)(r + l + 1)/2 \neq 0 \pmod{m}\), since \( m \) can divide neither \( r + l + 1 \) nor \( r - l \).

2. Let \( r - l \) be odd. Then, if one of \( r, l \) is odd, the other one is even, and the sum \( r + l + 1 \) is even. As \( r - l \) is odd it is not divisible by \( m \) or 2. We shall show that \((r + l + 1)/2\) is not divisible by \( m \) as well. As \( r, l \leq m - 1 \), \( r + l + 1 \leq 2(m - 1) + 1 = 2m - 1 \). \( r + l + 1 \) is even, \( 2m - 1 \) is odd, thus \( r + l + 1 \) can not be equal to \( 2m - 1 \) and therefore \( r + l + 1 \leq 2m - 2 = 2(m - 1) \). Then \((r + l + 1)/2 \leq m - 1 < m\), and therefore it is not divisible by \( m \) (as this would imply that \((r + l + 1)/2 \geq m\)). This means that \((r - l)(r + l + 1)/2 \neq 0 \pmod{m}\) as well.

For both cases we derived a contradiction \( h(k, l) - h(k, r) = (r - l)(r + l + 1)/2 \neq 0 \pmod{m}\) to the assumption that \( h(k, l) - h(k, r) \equiv 0 \pmod{m} \). Since no two values among the 0 through \( m - 1 \) collide with each other, the cover all the values/indices/slots of the hash table as required.

**Problem 13.**

First insert the strings and then perform a preorder traversal. In the preorder traversal a string is printed only for the red nodes (not for the black nodes that do not correspond to strings).

Note that the preorder is the correct traversal as a node is printed before its descendants. A printed node \( x \) is a red node and the string corresponding to that node should be printed before the strings corresponding to its descendants (in the left or right subtree of \( x \)). Strings corresponding to nodes of \( x \)’s left subtree must be printed before the ones corresponding to nodes of \( x \)’s right subtree as up to node \( x \) both strings are the same, but just after \( x \) the strings in the left subtree have a leading zero and the ones in the right subtree have a leading 1; the former must be printed before the latter ones. Preorder traversal guarantees that.

Running time of algorithm is \( O(n) \), as insertion takes time proportional to length of individual string being inserted and sum of all strings is \( n \).

**Problem 14.**

\((M_0 \times (M_1 \times M_2)) \times M_3\) at a cost of 2200000.