1 AVL Trees: Note that a tree containing only the root has zero height.

1(a) Draw a height 5 AVL tree containing a minimum number of nodes.

Answer:

1(b) What are the minimum and maximum numbers of nodes in an AVL tree of height 8?

Answer:

The minimum number of nodes in a tree of height $h$ is given by:

$$ N_h = N_{h-1} + N_{h-2} + 1 $$

$N_0 = 1$ and $N_1 = 2$, so $N_2 = 4$, $N_3 = 7$, $N_4 = 12$, $N_5 = 20$, $N_6 = 33$, $N_7 = 54$, and $N_8 = 88$. The maximum number of nodes is $2^{h+1} - 1 = 2^9 - 1 = 511$. 
1(c) Draw the AVL tree that results from the insertion of each of the key values 4, 6, 7, 8, 10, 12 in the order given into an initially empty AVL tree (i.e. draw 6 trees). For each rotation required during this process write $SR(n)$ or $DR(n)$ for Single Rotation and Double Rotation, where $n$ is the out-of-balance node’s key value.

**Answer:**

```
  4     4     4     SR(4)     6     7
  6     6

  6     6
  4     4
  7     7
  8     8

SR(7)

  6
  4
  8
  7

SR(6)

  6
  4
  8
  7

OR

2
```
2(a) Give an equation for the number of times the function $f()$ is executed in the following code fragment. Your answer may include summations.

```java
for(int i=1;i<=N/2;i++)
    for(int j=N-i;j>=i;j--)
        for(int k=i;k<j;k++)
            f(i,j,k);
```

Answer:

$$
= \sum_{i=1}^{N/2-1} \sum_{j=i}^{N-i} \sum_{k=i}^{j} 1
= \sum_{i=1}^{N/2-1} (j - i)
= \sum_{i=1}^{N/2-1} (N - 2i)(N - 2i + 1)
= \frac{N(N - 2)(2N + 1)}{24}
$$

2(b) Give a recursive function `bool three_sum(int *A, int key)` in C++ or Java to determine if any three values in the integer array $A_0, A_1, \ldots, A_{N-1}$ sum to an input value $k$. You can assume each value in the array is not repeated. Your function should return a `bool` to indicate success or failure.

Answer:

```java
bool three_sum(int *A, int key) {
    for(int i=0;i<N-2;i++)
        for(int j=i+1;j<N-1;j++)
            for(int k=j+1;k<N;k++)
                    return true;
    return false;
}
```
2(c) Would it help if the values in $A$ were in ascending order? Explain.

**Answer:**
Yes, because the $i$ loop could be stopped as soon as $A[i] > \text{key}$, and the $j$ loop could be stopped as soon as $(A[i] + A[j]) > \text{key}$.
This would not affect the worst case, but it would improve the average case.

2(d) What is the time complexity of the problem in part (b)?

**Answer:**
The above solution takes $(N - 2)(N - 3)(N - 4) = O(N^3)$ time, but we are asked for the complexity of the problem. It is easy to see how to reduce the complexity to $O(N^2 \log N)$ time, by first sorting the array, taking $O(\log N)$ time, and then using the above code with binary search in place of the $k$ loop.
class Node {
private:
  int value;
  Node *next;
public:
  Node(int val=-1, Node* p=NULL):
    value(val), next(p) {}
  Node *get_next()
    {return next;}
  int get_value()
    {return value;}
  void put_next(Node *p)
    {next = p;}
  void put_value(int x)
    {value = x;}
};

class LinkedList {
private:
  Node *head;
public:
  LinkedList()
    {head = new Node();} // dummy element
~LinkedList();
  int get_length(); // inserts a new node after dummy element
  void insert(int key); // removes & returns a pointer to the
  Node *remove(); // element after the dummy or returns
                   // NULL if the list is empty
  Node *search(int key); // returns a pointer to the node containing
                         // key or returns NULL if key isn’t present
  void remove(int key); // deletes the node containing key or
                        // does nothing if key isn’t present
};
3(a) Give the body of the LinkedList destructor.

Answer:

```cpp
LinkedList::~LinkedList() {
    Node *p = head;
    while(p!=NULL) {
        head = p->get_next();
        delete p;
        p = head;
    }
}
```

3(b) Give the body of the insert(int) function.

Answer:

```cpp
void LinkedList::insert(int key) {
    Node *p = new Node(key, head->get_next());
    head->put_next(p);
}
```

3(c) Give the body of the remove(int) function.

Answer:

```cpp
void LinkedList::remove(int key) {
    Node *p = head;
    while(p!=NULL) {
        Node *q = p->get_next();
        if(q!=NULL && q->get_value() == key) {
            p->put_next(q->get_next());
            delete q;
            break;
        }
        p = p->get_next();
    }
}
```
3(d) Give pseudo code, or explain the workings of, an efficient member function, `void delete_duplicate_keys()`, that modifies the list such that each of the key values in the input list appears just once. What are the space and time complexities of your method?

**Answer:**
The Naive way would be to select each element in order of its presence in the list and to scan all elements beyond deleting duplicates. This would take $O(N^2)$ time and constant space complexity.

A faster way would be to copy the elements into an array, taking $O(N)$ time, then sort the array, taking $O(N \log N)$ time, then insert the elements back into a list, skipping over duplicates, taking $O(N)$ time. The total time complexity would be $O(N \log N)$, and the space complexity would be $O(N)$. 