Q1. Describe an algorithm for reversing a singly linked list L using only a constant amount of additional space and not using any recursion.

Solution:

Algorithm reverse(SLinkedList L)
Input: A singly linked list L.
Output: The reversed singly linked list of L.
{
    if L.head!=null
    {
        currentNode=L.head
        previousNode=null;
        while currentNode!=null do
            temp=currentNode.getNext();
            currentNode.setNext(previousNode);
            previousNode=currentNode;
            currentNode=temp;
        }
        L.head=previousNode;
    }
}

Q2. Describe a recursive algorithm for converting a string of digits into the integer it represents. For example, “12520” represents the integer 12,520.

Solution: Assume that the string length is n and the string is stored in an array a, where a[n-1] stores the most significant digit and a[0] stores the least significant digit.

Algorithm stringToInt( int a[], int i)
Input: a: the array storing the string, and integer i.
Output: The integer of the string stored in a[n-1] a[n-2], …, a[i-].
{
    if i=1
        return a[n-1] - '0';
    return stringToInt(a, i-1)*10+a[n-i] - '0'
}

Q3. Describe a fast recursive algorithm for reversing a singly linked list.

Solution:

Algorithm reverse(SLinkedList L, Node P)
Input: A singly linked list L and a node P.
Output: The reversed singly linked list of the sublist starting with P.
{
    if P.getNext()==null // P is the last node
        { oldhead=L.head;
          L.head=P;
          return P;
        }
    temp=reverse(L, P.getNext());
    temp.setNext(P);
    if P==oldhead // P is the first node in the original list
Q4. Describe a recursive algorithm that checks if an array A of integers contains an integer A[i] that is the sum of two integers that appear earlier in A, that is, such that A[i]=A[j]+A[k] for j,k<i.

Solution:

Algorithm Pairfinding(A,i)
Input: An n-element array A of integers and a non-negative integer i.
Output: A pair of nonnegative integers j and k (j,k<i) such that A[j]+A[k]=A[i], or (0,0) if such a pair does not exists.
{
  if i<=1 // base case
    return (0,0);
  else
    for j=1, ..., i-1 do
      for k=0, 1, ..., j-1 do
          return (j, k);
    return Pairfinding(A,i-1);
}

The worst-case running time analysis:
In the worst case, Pairfinding(A,i) is called for each i=n-1, n-2, ..., 1. The running time of the else part is O(1)+ ... + O(i) = O(i^2). The running time of the if part is O(1). So each call of Pairfinding(A,i,k) takes O(1)+O(i)=O(i) time. Therefore, the worst-case running time of the algorithm is O(n^2)+O((n-1)+ ... +O(1))=O(n^3).

Q5. Write a linear time, non-recursive algorithm for computing the nth Fibonacci number, and prove that the time complexity of your algorithm is O(n).

Solution:

Algorithm Fibonacci(n)
Input: A non-negative integer n.
Output: The nth Fibonacci number.
{
  previous=-1;
  result=1;
  i=0;
  while i<n
    [ sum=result+previous;
      previous=result;
      result=sum;
      i++;
    ]
  return result;
}