Part I – Algorithm Analysis

What is the number of additions and subtractions that the following algorithm does?

```plaintext
index = 1
sum = 0
while index <= 90 do
  if (index mod 2 == 0) then
    sum = sum + index
    index = index - 1
  else
    temp = sum + sum
    sum = sum + temp + index
    index = index + 5
  end if
end while
```

First, we notice that index will alternate between being odd and even. When index is odd, we do the else clause and add 5 to it making it even. When index is even, we do the then clause and subtract 1 making it odd. So, in two subsequent passes, we will do the then and else clauses once each. We now examine how many times the loop is done. We see that the index takes on the 44 values in the sequence: 1, 6, 5, 10, 9, 14, 13, 18, 17, 22, 21, 26, 25, 30, 29, 34, 33, 38, 37, 42, 41, 46, 45, 50, 49, 54, 53, 58, 57, 62, 61, 66, 65, 70, 69, 74, 73, 78, 77, 82, 81, 86, 85, 90. For every pair of values, we will do 6 additions and subtractions, so we do 132 additions and subtractions overall.

Part II - Algorithms and their analysis

Write an algorithm that will find, in a sorted list, the location of the largest element that is less than or equal to a given input. (For example, with the list 1 3 5 7 9, and an input of 6, the algorithm would return 3, the location of the 5.) Then analyze that algorithm to determine how many comparisons it does in the best, worst, and average cases.

```plaintext
int Find( list, N, key )
I = 1
while I <= N do
  if ( list[I] >= key) then
    return I-1
  end if
  I++
end while
return N
```

The following answer does not count the comparison of I and N in the while loop, it only counts the comparison of the key and the list elements. Remember that when we analyze an algorithm we do not count comparisons of loop variables and limits, but we count all comparisons of keys, and list elements.

The best case would be if the key is smaller than all of the elements in the list, which does one comparison. The worst case would be if the key is larger than all of the elements of the list, which does N comparisons. In the average case, we notice that when the value returned is 0, there is one comparison and I is 1, and when the value returned is 1, there are two comparisons and I is 2. On the last pass of the loop, when I is N and we are doing the N comparison, if the conditional is true we return N-1 and when it is false we return N. This means we have N+1 cases to consider. In the first N cases, we do 1, 2, 3, ..., N comparisons, and in the last case we also do N comparisons. So the average case is given by:

\[
\sum_{i=1}^{N} \frac{i+N}{N+1} \approx \frac{N(N+1)+N}{2(N+1)} = \frac{N^2 + N + 2N}{2(N+1)} = \frac{N}{2}
\]
Part III – Algorithms and their analysis

Write an algorithm that will find, in an array of integers, if there are any three adjacent locations that have consecutive values in order. If there is such a set of numbers, the algorithm should return the location of the first of these numbers. If the numbers do not show this property, the algorithm should return -1.

Analyze your algorithm to determine how many comparisons it does in the worst and average cases.

```c
int FindSpot( list, N )
for (j = 1 to N - 2)
  if (list[j]+1 == list[j+1]) then
    if (list[j+1]+1 == list[j+2]) then
      return j
    end if
  end if
end for
return -1
```

Worst Case Analysis:
It is easy to just say that in the worst case, there will be two comparisons done for each of the N-2 passes of the loop, making the worst case 2(N-2). But this is not true. Watch what happens on the first pass. If we do two comparisons and fail on the second, that means the first three values must be something like 2, 3, and 5. This means that on the second pass, the first conditional will fail.

In general, if we do two comparisons on any pass, we will do exactly one on the next. If we do one comparison on a pass, we can do any number on the next pass after that. This means that the complete set of cases for three consecutive locations is:

<table>
<thead>
<tr>
<th>location</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>j+1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

We see that the worst case does three comparisons over two passes or 1.5 comparisons per pass. This gives a worst case of 1.5*(N-2).

Average Case Analysis:
We can use this knowledge to determine the average case. Since the previous table shows all of the possible cases for two consecutive locations, if we average these values, we will find that the average for two passes is about 2.67, making the average for one pass about 1.333.

We now need to consider all of the possibilities. If there are three consecutive numbers in a row, the last pass will do 2 comparisons, but all of the previous passes will do on average 1.33 comparisons. If there is no place in the list with three consecutive numbers in a row, that possibility will do 1.33*(N-2) comparisons. So we have N - 2 possible places we could find three consecutive numbers in a row, and one possibility we will find none. The overall equation for this is:

\[
\sum_{i=0}^{N-3} (1.33*i + 2) + 1.33*(N - 2) = \frac{1.33*(N-3)(N-2)}{2} + 2*(N - 2) + 1.33*(N - 2)
\]

\[
= \frac{0.67*(N - 3)(N - 2) + 2*N - 4 + 1.33*N - 2.67}{N - 1}
\]

\[
= \frac{0.67*(N^2 - 5*N + 6) + 3.33*N - 6.67}{N - 1}
\]

\[
= \frac{0.67*N^2 - 3.33*N + 4 + 3.33*N - 6.67}{N - 1}
\]

\[
= \frac{0.67*N^2 - 10}{N - 1}
\]

\[
\approx 0.67 * N
\]
Part IV – Recursive Algorithms

```c
int Sample( N )
// N   the Nth Fibonacci number should be returned

if N < 4 then
    return 1
else
    value1 = N – 1
    value2 = N – 1
    return Sample( value1 ) + Sample( value2 )
end if
```

For the above example, what is the direct solution and how many “additions” does it do? What is the division of input and how many multiplications does it do? How many smaller sized problems are there and how big are they? What is the combination of solutions and how many additions does it do? After answering all of these questions, create the recurrence relation for this algorithm.

The direct solution is the return of 1 and does no additions. The division of input is the calculation of value1 and value2 and there are two additions done. There are two smaller sized problems and both are N-1. The combination of the solutions is the addition of the recursive call results and this does one addition. The recurrence relation for this algorithm is:

\[
A(N) = \begin{cases} 
0 & \text{if } N < 4 \\
2 + 2 * A(N-1) + 1 & \text{if } N \geq 4 
\end{cases}
\]

Part V – Recursive Algorithms

The result of raising one number to a positive integer power can be calculated recursively based on the equation \( x^y = x * x^{y-1} \). Write a recursive function that calculates the power using this equation. Based on your algorithm, what is the direct solution and how many “multiplications” does it do? What is the division of input and how many multiplications does it do? How many smaller sized problems are there and how big are they? What is the combination of solutions and how many multiplications does it do? After answering all of these questions, create the recurrence relation for this algorithm.

```c
int Power(x, y)
if y == 0 then
    return 1
else
    return x * Power(x, y-1)
end if
```

The direct solution is the return of 1 and does no multiplications. The division of input is the calculation of y-1, which does no divisions. There is one smaller sized problem and it is of size N-1. The combination is the multiplication of x and the recursive result, which does one multiplication. The resulting recurrence relation is:

\[
M(N) = \begin{cases} 
0 & \text{if } N == 0 \\
M(N-1) + 1 & \text{if } N > 0 
\end{cases}
\]