1. a) i) Tower\_of\_hanoi (disk, source, dest, spare)

IF disk equals 0 DO

Move Disk from source to dest

Else DO

Tower\_of\_hanoi (disk – 1, source, spare, dest)

Move Disk from source to dest

Tower\_of\_hanoi (disk -1, spare, dest, source)

ii) Tower\_of\_hanoi (3, source, dest, spare)

Tower\_of\_hanoi (2, source, spare, dest)

Tower\_of\_hanoi (1, source, dest, spare)

Tower\_of\_hanoi (0, source, spare, dest)

Tower\_of\_hanoi (0, spare, dest, source)

Tower\_of\_hanoi (1, dest, spare, source)

Tower\_of\_hanoi (0, dest, source, spare)

Tower\_of\_hanoi (0, source, spare, dest)

iii) The big O is N (if N == 0) or (N to the power of how many disks used) – 1 My reasoning behind this is the more elements involved, the more steps needed. It always needed one step less than 2 to the number of elements within the programme. The more disks added, the more loops are needed.

b) i)

AddNums (Total, current\_num)

IF Total equals amount of numbers in list

Return current\_num

Else

AddNums(Total + 1, current\_num + next number)

END

ii) AddNums(0,0)

AddNums(0+1 ,0+8)

AddNum(1+1, 8+3)

AddNum(2+1, 11+7)

AddNum(3+1, 18+4)

AddNum(4+1, 22+2)

AddNum(5+1, 24+1)

iii) The big O is N as it is only looping around once. All it is doing is adding each number one by one. If done iterative, this would require only one loop.

1. a) i) The sieve of erastosthenes algorithm starts at the first prime number and marks all numbers that allocated number can divide evenly into. After it has checked every number, It go to the next unmarked number, mark that one as prime, and mark every unmarked number that this can divide into. Repeat this until we have marked every number, either prime or not. We start at two.

ii) Start Programme

SET two as Prime & mark as Prime

WHILE we are not at the end of number list

IF num mod Prime equals zero DO

IF number is not marked “Not-prime” DO

Mark as “Not-prime”

END IF

END IF

Search for next unmarked number

SET as Prime & mark as Prime

GOTO WHILE

END PROGRAMME

b) i) The first problem is the min assignment. This itself is right. It goes using that number as a position, which is incorrect. In the first sequence, the comparison is “if A [0] (A position 1 equals 2) < A [min] (A position 2 equals 1) It should if A[0] < min. Since this is never to be right in the first time, this will not change the min. After the if statement the temp will store an element out of the bounds. It then overwrites position min, and replace it with the element that is out of position.

ii) Insertion sort because it will go through 6 moves only. My other reason is the list is small, so any sorting method that would be much quicker on a larger set would be approximate the same speed as the insertion sort itself.

1. BINSEARRCH(low, mid, high)

IF mid = number wanted DO

Return mid

Else DO

IF num < mid

BINSEARCH(low, mid/2, mid)

Else DO

BINSEARCH(mid, (mid + high)/2, high)

END Programme

BINSEARCH(1,6,12)

BINSEARCH(6, 9, 12)

BINSEARCH (9, 10, 12)

BINSEARCH(10, 11, 12)