1)

1. Suppose that the algorithm is incorrect for some negative integer z>0
2. For that integer z >=0 the value of product should not be equal to y*z (product ≠ y*z).
3. The input range for the two variables is y= [-inf ,inf] and z= [0,inf] , so the base cases are z = 0 and z =1.
4. For z=0 the product is zero according to step1. For z=1 the product is y according to step 2. Hence for the base cases we are getting the correct output.
5. For input z>1 there should be some z for which the product ≠ y*z . This happens only when the y is not summed z times.
6. From steps 3 to 6, we can see that the y is summed with itself for z times for every input z>1, I.e
7. Product = y + y + y ....(z times).
8. Hence the final product = y*z always for every input where z >1
9. And from the above statements we can say that the product = y*z for every input z>=0.
10. Hence the assumption that there exists a z>=0 for which product ≠ y*z is false.
11. Hence the assumption that the Algorithm is incorrect is false. The Algorithm is correct.

2)

Initialization:

Before the start of the loop, the value of i=1. It is an empty array and an empty array has no elements to be sorted. So it is sorted array of zero elements and LI holds true.

Maintainance:

1. Let us assume that the LI holds true for i = p , that is all the elements from 1 to p-1 are in sorted order (A[1]> A[2]>...........>A[p-1]) at the end of the iteration i = p.
2. Before the beginning of the loop i = p+1, we have the sorted array from 1 to p-1 in the descending order. At the end of the iteration we should be having the sorted array from 1 to p to say that LI holds true.
3. Now at the iteration i= p+1, the inner loop goes from j =n down to p+1. According to steps 2 to 6, at each j we compare j with j-1 and the largest of the two number stays at j-1 position.
4. At the end of this inner loop we will have the largest element of Array[p to n] at the position p. That is at the end of the inner loop we will have A[p] > A[p+1....n].
5. Before the start of the iteration we have A[1] >A[2]>A[3]>........>A[p-1]. We also have A[p-1] > A[p......n] I.e A[p-1] > A[p].
6. At the end of the iteration i = p+1 we have A[p] > A[p+1......n] and from point 5 we have A[p-1]>A[p]. Hence we can say that at the end of the iteration we have a sorted array from A[1.....p] which is in decreasing order.
7. Hence LI holds true at the end of the iteration and for next iteration.

Termination:

Now given the initialization and maintenance proofs, the LI holds true for the iteration i = k+1 and at the end of the iteration we will have a sorted array from from 1 to k. Hence we will have the kth largest number at position k , and we can return A[k]. Hence the algorithm produces the correct output.

3)

```
Bubble-sort (A: Array [1..n] of numbers)
1 i=1
2 while i≤k
3         j=1
4         while j≤(n–i)
5                 if A[j]>A[j+1] then
6                         temp=A[j]
7                         A[j]=A[j+1]
8                         A[j+1]=temp
9                 j=j+1
10        i=i+1
11 return A[n-k-1]
```

In step2 its enough if we run the loop k times we need not run it n-1 times to get the kth largest element.

4)

```
power-iterative(x,y: non-negative integers)
    result = 1
    while  y>0 :
            if  odd(y ) then
                    result =  result * x
                    y = y-1
            end if
            x = x*x
            y= y/2
    return result
```

5)

A)

```
Reverse-String(A[1…n])
1.  i = 1
2.  While i  < = n/2
3.       Temp = A[i]
4.        A[i] = A[n-i+1]
5.        A[n-i+1] = Temp
6.        i = i +1
```

B)

This algorithm has 3 functions and algorithm for 3 functions is as follows.

```
Reverse_String(A[1…n])
1.  Split(A,1 , n )
```

```
Split(A[1…n], l , r )
If l <  r
     Split(A, l, m)
     Split(A, m+1, r)
     Merge(A, l, m,  r)

Merge(A[1…n], l, m, r)
Left_Array[1…n1] = A[l…..m]
Right_Array[1…n2] = A[m+1…..r]
j1 = 1  , j2 = 1 , i = l
while j2 <= n2
    A[i] = Right_Array[j2]
    j2 = j2 + 1
    i = i + 1
while  j1  > 0
    A[i] = Left_Array[j1]
    j1 = j1 + 1
    i = i + 1
```