## Exercise 2·1

~~#~~2)

Loop invariant:- At the start of each iteration of the loop, the variable 'sum' is the sum of all the elements in the sub-array ~~&~~ A[~~0~~1 : i-1]

Initialisation: Before the first iteration of ~~i=~~ the for loop, i = 1, ~~so~~ and is then sum = 0. ~~Since~~ The sub-array A[1 : ~~0~~1-1] = A[1 : 0] which has no elements, and so the sum of its elements is 0 and the loop invariant is true.

Maintenance: Suppose the loop invariant is true before ~~#~~ an iteration. ~~That means for some integer k < n, the suban 'sum' is the sum of the elements in the sub array A[1 : k]. Then after an iteration~~
~~Then after during the ireiterat~~
Then during ~~after~~ the iteration, ~~8~~ A[i] is added to ~~th~~ the 'sum', which was already the sum of A[1 : i-1], and so by definition of sum, it is now the sum of the elements of A[1 : i]. ~~Since~~ Since i = i+1 ~~e~~ before the next iteration, the variable sum is hence the sum of the elements of the ~~a~~ subarray A[1 : i-1] at the start of the next iteration and hence the invariant is still true.

Termination:- Once the loop variable ~~terminates,~~ 'i' exceeds n, i.e. $i=n+1$, the loop terminates. Letting $i=n+1$ in the wording of the loop invariant yields that the variable 'sum' is the sum of the elements of the subarray $A[1:(n+1)-1] = A[1:n]$ which is equal to the entire array. Hence, the algorith is correct.

3) Insertion sort (A, n) // Decreasing

~~for~~ ~~for~~ i = 2 ~~to~~ n
    ~~key~~ =

for i = n downto 2
    key = A[i]
    j =

3)

Insertion sort (A,n) //Decreasing order

```
for i=2 to n
    key= A[i]
    j= ~~j-1~~ i-1
    while j > 0 and A[j] < key
        A[j+1] = A[j]
        j = j-1
    A[j+1] = key
```

4)

Linear Search (A,x)

```
  n = A.length
  for j=1 to ~~A.length~~ n
      if A[j] == x
          return j
  return NIL
```

Loop invariant :- Before each iteration of the loop with index j, there is no element in the sub array A[1 : j-1] ~~such~~ that equals x

Initialisation:- At the start of the first iteration j = 1, and so ~~there is no~~ invariant is true because there are no elements in the sub array A[1 : 0] in the first place

Maintenance :- ~~If the After each~~ During any iteration, either the loop is exited because an index i is found such that $A[i] == x$ (and thus the algorithm is correct) or ~~the index~~ $A[j] == x$ turns out to be falso, and so ~~A[~~ the sub array $A[1:j]$ does not contain the variable $x$. Before the next iteration, j is incremented by 1, so the sub array $A[1:j-1]$ does not contain ~~the element~~ $x$, and so the invariant stays true

Termination :- The loop terminates either when an index i is found such that $A[i] == x$ ~~(and~~ and then it is returned (thus the algorithm is correct) or when $j = n+1$~~, in Subst~~, which means there is no $x$ in the sub array $A[1:n+1-1] = A[1:n]$ (which is simply the entire array) and so NIL is returned and the algorithm is correct.

5)

5) ~~AD~~

```
ADD-BINARY-INTEGERS (A,B,n)
    C= new integer[n+1]
    carry=0
    for i=0 to n-1
        sum= carry+A[i]+B[i]
        if sum==0
            C[i]=0
        else-if sum==1
            C[i]=1
            carry=0
        else-if sum==2
            C[i]=0
            carry=1
        else-if sum==3
            C[i]=1
    C[n]=carry
    return C
```