Pseudocode For Data Structures and Algorithms

Stack Data Structure

Initialize an empty stack:

```
stack =  [ ]
push:

increment top
stack[top] assign value

pop :

store value of stack[top]
decrement top
return value
```

Queue Data Structure

Initialize an empty stack:

```
queue = [ ]
front
rear

enqueue:

increment rear
queue[rear] assign value

dequeue:
```

store value of queue[front]
increment front
return value

Linear Search

```
linearSearch(items, target)
  for i from 0 to length – 1
  if items[i] == target
          return i
    end if
  end for
  return not found
```

`to length(arr) - 1`: This specifies the condition for the loop to continue. The loop will continue iterating as long as `i` is less than or equal to the index of the last element in the array (`length(arr) - 1`).

Binary_Search

Set first index to 0. Set last index to the last subscript in the array
Set found to false. Set position to -1

While found is not true and first is less than or equal to last
    Set middle to the subscript half-way between array[first] and array[last]

    If array[middle] equals the desired value
          Set found to true.
          Set position to middle.

   Else If array[middle] is greater than the desired value
          Set last to middle - 1.
      Else
          Set first to middle + 1.
    End If

End While
Return position.

Bubble Sort

```
Initialize n = Length of Array

bubblesort(Array, n)

   for i = 0 to n-1

      for j = 0 to n-1

         if Array[j] > Array[j+1]
            swap(Array[j], Array[j+1])
         end if
      end for
  end for
```

Selection Sort

```
Initialize n = Length of Array

selectionsort (Array, n)

   for i = 0 to n-1

     i_min = i

      for j = i+1 to n-1

         if Array[j] < Array[i_min]
            i_min = j
         end if
       Swap(Array[j], Array[i_min])
     end for
```

```
  end for
```

Insertion Sort

```
Initialize n = Length of Array

insertionsort(Array, n)

    for i = 1 to n-1

        value = Array[i]
        position = i
        while (position > 0 and Array[position-1] > value)

            Array[position] = Array[position - 1]
            position = position - 1
        end while
        Array[position] = value;
    end for
```

Merge Sort

```
mergesort(A)

n = length of Array (A)
  if(n<2)
        return
  end if
mid = n/2
left = array of size(mid)
right = array of size(n-mid)

for i = 0 to mid-1
      left[i] = A[i]
end for
for i = mid to n-1
```

```
    right[i-mid] = A[i]
end for
mergesort(left)
mergesort(right)
merge(left, right, A)
```