1. Largest element in array

```
START
  READ A
  SET max = A[0]
  FOR i = 1 to n-1
    IF A[i] > max THEN
      max = A[i]
    ENDIF
  END FOR
  PRINT max
END
```

2. Second largest element

```
START
  READ A
  SET max = -INF, sec = -INF
  FOR each x in A
    IF x > max THEN
      sec = max
      max = x
    ELSE IF x > sec AND x != max THEN
      sec = x
    ENDIF
  END FOR
  PRINT sec
END
```

3. Reverse array

```
START
  READ A
  FOR i = n-1 down to 0
    PRINT A[i]
  END FOR
END
```

4. Remove duplicates from array

```
START
  READ A
  CREATE empty SET S
  FOR each x in A
    ADD x to S
  END FOR
  PRINT S
END
```

5. Bubble sort

```
START
  READ A
  FOR i = 0 to n-2
    FOR j = 0 to n-i-2
      IF A[j] > A[j+1] THEN SWAP A[j], A[j+1]
    END FOR
  END FOR
  PRINT A
END
```

6. Linear search

```
START
  READ A, key
  SET found = FALSE
  FOR each x in A
    IF x == key THEN found = TRUE
  END FOR
  PRINT found
END
```

7. Binary search

```
START
  READ sorted A, key
  SET low = 0, high = n-1
  WHILE low <= high
    mid = (low + high) / 2
    IF A[mid] == key THEN
      PRINT "Found"
      STOP
    ELSE IF key > A[mid] THEN
      low = mid + 1
    ELSE
      high = mid - 1
    ENDIF
  END WHILE
  PRINT "Not Found"
END
```

8. Missing number from 1 to N

```
START
  READ A, N
  expected = N * (N + 1) / 2
  sum = 0
  FOR each x in A
    sum = sum + x
  END FOR
  PRINT expected - sum
END
```

9. Rotate array left by one

```
START
  READ A
  first = A[0]
  FOR i = 0 to n-2
    A[i] = A[i+1]
  END FOR
  A[n-1] = first
  PRINT A
END
```

10. Count even and odd elements

```
START
  READ A
  even = 0, odd = 0
  FOR each x in A
    IF x % 2 == 0 THEN even++
    ELSE odd++
```

```
    ENDIF
  END FOR
  PRINT even, odd
END
```

11. Check palindrome string

```
START
  READ s
  rev = ""
  FOR i = length(s)-1 down to 0
    rev = rev + s[i]
  END FOR
  IF s == rev THEN PRINT "Palindrome"
  ELSE PRINT "Not Palindrome"
  ENDIF
END
```

12. Count vowels and consonants

```
START
  READ s
  v = 0, c = 0
  FOR each ch in s
    IF ch in (a,e,i,o,u,A,E,I,O,U) THEN v++
    ELSE IF ch is alphabet THEN c++
    ENDIF
  END FOR
  PRINT v, c
END
```

13. Reverse string

```
START
  READ s
  rev = ""
  FOR i = length(s)-1 down to 0
    rev = rev + s[i]
  END FOR
  PRINT rev
END
```

14. Check anagram strings

```
START
  READ a, b
  SORT a
  SORT b
  IF a == b THEN PRINT "Anagram"
  ELSE PRINT "Not Anagram"
  ENDIF
END
```

15. First non-repeating character

```
START
  READ s
  FOR each ch in s
    IF index_of(ch) == last_index_of(ch) THEN
      PRINT ch
      STOP
```

```
    ENDIF
  END FOR
END
```

16. Insert node at beginning (Linked List)

```
START
  CREATE node N
  N.next = head
  head = N
END
```

17. Insert node at end

```
START
  CREATE node N
  IF head == NULL THEN head = N
  ELSE
    temp = head
    WHILE temp.next != NULL
      temp = temp.next
    END WHILE
    temp.next = N
  ENDIF
END
```

18. Reverse a linked list

```
START
  prev = NULL
  curr = head
  WHILE curr != NULL
    next = curr.next
    curr.next = prev
    prev = curr
    curr = next
  END WHILE
  head = prev
END
```

19. Middle of linked list

```
START
  slow = head
  fast = head
  WHILE fast != NULL AND fast.next != NULL
    slow = slow.next
    fast = fast.next.next
  END WHILE
  PRINT slow
END
```

20. Detect cycle in linked list

```
START
  slow = head
  fast = head
  WHILE fast != NULL AND fast.next != NULL
    slow = slow.next
    fast = fast.next.next
    IF slow == fast THEN
```

```
      PRINT "Cycle"
      STOP
    ENDIF
  END WHILE
  PRINT "No Cycle"
END
```

21. Implement stack using array

```
START
  push(x): top = top + 1; S[top] = x
  pop(): return S[top]; top = top - 1
END
```

22. Check balanced parentheses using stack

```
START
  CREATE empty stack
  FOR each ch in s
    IF ch == '(' THEN push
    ELSE IF ch == ')' THEN
      IF stack empty THEN PRINT "Not Balanced"; STOP
      ELSE pop
      ENDIF
    ENDIF
  END FOR
  IF stack empty THEN PRINT "Balanced"
  ELSE PRINT "Not Balanced"
END
```

23. Reverse a stack using recursion

```
START
  reverse(stack):
    IF stack empty THEN RETURN
    temp = pop()
    reverse(stack)
    insertBottom(stack, temp)
END
```

24. Implement queue using array

```
START
  enqueue(x): Q[rear] = x; rear++
  dequeue(): x = Q[front]; front++; return x
END
```

25. Implement circular queue

```
START
  enqueue(x): Q[rear] = x; rear = (rear + 1) % N
  dequeue(): x = Q[front]; front = (front + 1) % N
END
```

26. Bubble sort (again)

(See 5)

27. Selection sort

```
START
  FOR i = 0 to n-2
    min = i
```

```
    FOR j = i+1 to n-1
      IF A[j] < A[min] THEN min = j
    END FOR
    SWAP A[i], A[min]
  END FOR
END
```

28. Insertion sort

```
START
  FOR i = 1 to n-1
    key = A[i]
    j = i - 1
    WHILE j >= 0 AND A[j] > key
      A[j+1] = A[j]
      j--
    END WHILE
    A[j+1] = key
  END FOR
END
```

29. Binary search

(See 7)

30. Merge sort (logic)

```
START
  divide array into two halves
  recursively sort left half
  recursively sort right half
  merge both sorted halves
END
```

31. Fibonacci using recursion

```
START
  fib(n):
    IF n <= 1 THEN return n
    ELSE return fib(n-1) + fib(n-2)
END
```

32. Factorial using recursion

```
START
  fact(n):
    IF n == 0 THEN return 1
    ELSE return n * fact(n-1)
END
```

33. Sum of digits using recursion

```
START
  sum(n):
    IF n == 0 THEN return 0
    ELSE return (n % 10) + sum(n / 10)
END
```

34. Inorder traversal of binary tree

```
START
  inorder(node):
    IF node == NULL THEN RETURN
```

```
    inorder(node.left)
    PRINT node
    inorder(node.right)
END
```

35. Preorder traversal

```
START
  preorder(node):
    IF node == NULL THEN RETURN
    PRINT node
    preorder(node.left)
    preorder(node.right)
END
```

36. Postorder traversal

```
START
  postorder(node):
    IF node == NULL THEN RETURN
    postorder(node.left)
    postorder(node.right)
    PRINT node
END
```

37. Insert into BST

```
START
  insert(root, key):
    IF root == NULL THEN create new node
    ELSE IF key < root THEN go left
    ELSE go right
END
```

38. Search key in BST

```
START
  search(root, key):
    IF root == NULL THEN return FALSE
    IF key == root THEN return TRUE
    ELSE IF key < root THEN search left
    ELSE search right
END
```

39. Count frequency using HashMap concept

```
START
  CREATE empty map
  FOR each x in A
    map[x] = map[x] + 1
  END FOR
  PRINT map
END
```

40. Find duplicates using HashSet concept

```
START
  CREATE empty set S
  FOR each x in A
    IF x in S THEN PRINT "Duplicate", x
    ELSE add x to S
    ENDIF
```

```
  END FOR
END
```