B1:

2 log(n) ,4n ,3n+100log(n), nlog(n) , 4nlogn + 2n , n2 + 10n ,n3 ,210 ,2n

B2:

C1:

Result = 1

For i from 1 to n:

Result = Result \*2

End for

Return Result;

Complexity : O(n) = n;

C2:

Function calculate2ToTheN(n):

if n == 0:

return 1

if n is even:

temp = calculate2ToTheN(n / 2)

return temp \* temp

else:

temp = calculate2ToTheN((n - 1) / 2)

return temp \* temp \* 2

Complexity : O(n) = log(n);

B3:

InitializeQueue():

queue = new Array

front = 0

rear = -1

Enqueue(queue, element):

rear = (rear + 1)

queue[rear] = element

Dequeue(queue):

element = queue[front]

front = (front + 1)

return element

Front(queue):

return queue[front]

B4:

Structure Node:

data

next

InitializeQueue():

front = None

tail= None

Enqueue(element):

new\_node = CreateNode(element)

if front is None:

front = new\_node

tail= new\_node

else:

tail.next = new\_node

tail= new\_node

Dequeue():

if front is None:

return "Queue is empty"

else:

element = front.data

front = front.next

if front is None:

tail= None

return element

Front():

if front is None:

return "Queue is empty"

else:

return front.data

B5:

InitializeStack():

stack = new Array

top = -1

Push(element, stack):

top = top + 1

stack[top] = element

Pop(stack):

if top == -1:

return "Stack is empty"

else:

element = stack[top]

top = top - 1

return element

Peek(stack):

if top == -1:

return "Stack is empty"

else:

return stack[top]

B6:

Structure Node:

data

next

InitializeStack():

top = None

Push(element):

new\_node = CreateNode(element)

if top is None:

top = new\_node

else:

new\_node.next = top

top = new\_node

Pop():

if top is None:

return "Stack is empty"

else:

element = top.data

top = top.next

return element

Peek():

if top is None:

return "Stack is empty"

else:

return top.data

IsEmpty():

return top == None