//C-2.1

```
Algorithm findMiddle(S):
  if (S.isEmpty()) return 0 endif
  p:=S.first();
  q:=S.last();
  while(p!=q and p.before()!=q) do
    p!=S.after(p)
    q!=S.after(q)
  endwhile;
  return p1;
```

// O(n) since both pointers meet in the middle, each moving one step per iteration, thus traversing half the list.

//C-2.2

```
Algorithm enqueu(x):
  stack1.push(x)

Algorithm dequeu():
  if(stack2.isEmpty()):
  while stack1 !isEmpty do:
    stack2.push(stack1.pop)
    endwhile;
  return stack2.pop();
```

//enqueue(): O(1) since it just pushes onto stack1.

//dequeue(): O(n) in the worst case when stack2 is empty and all elements need to be transferred from stack1 to stack2.

//C-2.3

```
Algorithm push(x)
   queu.enqueu(x);
   for i=1 to queu.size()-1
      queu.enque(queu.dequeu())
Algorithm pop()
   queu.dequeu();
```

//push(): O(n) due to rotating the queue.

//pop(): O(1) since it simply dequeues.

//A .

```
Algorithm removeDuplicates(S):
   Set seen={}
   p=S.first()

   while p!=null do
      if seen.contains(p) then S.remove(P)
      else seen.add (p)
      p=S.after(p)
```

//For a List: The same algorithm applies, but:

//Time Complexity for Sequence: O(n) (assuming a hash set is used to store duplicates).

//Time Complexity for List: O(n), but operations on positions might vary depending on the list's implementation (e.g., doubly linked list vs. array-based).

```
//B.
algorithm Powersets(n):
   if n == 0:
      return {{}}
   subsets=Powersets(n-1)

   newSubsets={}
   for subset in subsets:
      newSubsets.add(subset + {n})

   return subsets+newSubsets


//Time Complexity: O(2^n), as the number of subsets doubles with each added element.


//R-2.1
Algorithm insertBefore(p,e):
   newNode = Node(e);
   prev=p.before();

   prev.after=newNode
   newNode.before=prev;
   p.before=newNode;
   newNode.after=p;
```

```
Algorithm insertFirst(e):
   newNode = Node(e)
   first = L.first()

   L.header.after = newNode
   newNode.before = L.header
   newNode.after = first
   first.before = newNode

Algorithm insertLast(e):
   newNode = Node(e)
   last = L.last()

   last.after = newNode
   newNode.before = last
   newNode.after = L.trailer
   L.trailer.before = newNode
```

Time Complexity: O(1) for each of these operations in a doubly linked list.

//A.

```
Algorithm isBalanced(arr):
   Stack s = new Stack()

   for each char in arr:
```

```
    if char is '(', '[', or '{':
        s.push(char)
    else if char is ')', ']', or '}':
        if s.isEmpty():
            return false
        top = s.pop()
        if (char == ')' and top != '(') or
           (char == ']' and top != '[') or
           (char == '}' and top != '{'):
            return false

  return s.isEmpty()
```

Time Complexity: O(n)