What is the time complexity of the below code snippets?

1.

```
for i in range(n):
   k = 2 + 2
for j in range(3):
   k = 2 + 2
```

2.

```
for i in range(n):
   for j in range(n):
      k = 2 + 2
```

3.

```
for i in range(n):
   k = 2 + 2
for j in range(m):
   k = 2 + 2
```

4.

```
i = n
while i > 0:
   k = 2 + 2
   i = i // 2
```

5.

```
for i in range(n):
   for j in range(n):
      for k in range(n):
         k = 2 + 2
```

6.

```
i = n
while i > 0:
   k = 2 + 2
   i = i – 1
```

7.

```
i = n
while i > 0:
   k = 2 + 2
   i = i – 2
```

8.

```
i = 1
while i < n:
   k = 2 + 2
   i = i * 2
```

9.

```
i = 1
while i < 4:
   myList[i] = i
```

10.

```
for i in range(100098765):
   k = 2 + 2
```

11.

```
for i in range(100098765):
   for j in range(n)
       k = 2 + 2
```

What is the time complexity of these methods in this queue class (from the PS book)

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()
```

What is the memory complexity for the following code snippets (assuming there are n words in file ”text.txt”, and that the longest line contains m words)

1.

```
with open('text.txt') as f:
     text = f.read()
     for word in text.split():
        print (word)
```

2.

```
with open('text.txt') as f:
     for line in f:
       for word in line.split():
          print (word)
```

## Key

What is the time complexity of the below code snippets?

1.

```
for i in range(n):
   k = 2 + 2
for j in range(3):
   k = 2 + 2
```

O(n)

2.

```
for i in range(n):
   for j in range(n):
      k = 2 + 2
```

$O(n^2)$

3.

```
for i in range(n):
   k = 2 + 2
for j in range(m):
   k = 2 + 2
```

O(n+m)  or p = max(m,n), then O(p)

4.

```
i = n
while i > 0:
   k = 2 + 2
   i = i // 2
```

O(log n)

5.

```
for i in range(n):
   for j in range(n):
      for k in range(n):
         k = 2 + 2
```

$O(n^3)$

6.

```
i = n
while i > 0:
   k = 2 + 2
   i = i – 1
```

O(n)

7.

```
i = n
while i > 0:
   k = 2 + 2
   i = i – 2
```

O(n)

8.

```
i = 1
while i < n:
   k = 2 + 2
   i = i * 2
```

O(log n)

9.

```
i = 1                                   O(1)
while i < 4:
   myList[i] = i
```

10.

```
for i in range(100098765):              O(1)
   k = 2 + 2
```

12.

```
for i in range(100098765):              O(n)
   for j in range(n)
       k = 2 + 2
```

What is the time complexity of each method in this queue class (from the PS book)

```
class Queue:
    def __init__(self):
        self.items = []                             O(1)

    def isEmpty(self):                              O(1)
        return self.items == []

    def enqueue(self, item):                        O(n)
        self.items.insert(0,item)

    def dequeue(self):                              O(1)
        return self.items.pop()
```

Note that this simple way of implementing queue is not optimal. If we want to implemnt queue with a list, it is better to use moving start and end indeces, and never actually removing values, jsut keeping track of the active indeces. Then we can have O(1) also for enqueue.

What is the memory complexity for the following code snippets (assuming there are n words in file ”text.txt”, and that the longest line contains m words)

1.

```
with open('text.txt') as f:                 memory complexity: O(n)
     text = f.read()
     or word in text.split():
        print (word)
```

2.

```
with open('text.txt') as f:                 memory complexity: O(m)
     for line in f:
       for word in line.split():
        print (word)
```

Note that the time complexity is O(n) in both cases, since we always loop through each word