Big O notation describes how algorithmic run time increases with input size. Determine the algorithmic time complexity in Big O notation of the following functions where `values` is **a sorted list** of length `n`

```
def func0(values):
    for i in range(10):
        print(values[-1])

def func1(values):
    n = len(values)
    if n % 2 == 0:
        return (values[n//2-1] + values[n//2])/2
    else:
        return(values[n//2])

def func2(values, myValue):
    total = 0
    for item in values:
        if item == myValue:
            total += 1
    return total

def func3(values):
    for i in range(10):
        for j in range(100):
            print(values[-1])

def func4(values):
    for i in range(len(values)):
        for j in range(0, len(values), 2):
            print(values[i], values[j])
```

These next ones are a bit different – we don't have a list of elements, but a similar idea

```
def func5(n, m):
    for i in range(n):
        for j in range(m):
            print("Hello!")



def func6(n, m):
    for i in range(n):
        for j in range(m//2):
            for k in range(10):
                print("Hello!")
```