Exercise 1

Slowest to fastest: 9 – 6 - 1 – 5 – 4 – 2 – 8 – 3 – 7

Exercise 2

1. 6* 10 ^7 2. 7,746 3. 391 4. 10 5. 3,950,158
6. 8,649,296 7. 20 8. 153,262 9. 1.196 10. 12.92

Exercise 3

A

1. g(n)=O(f(n)) 2. g(n)=O(f(n)) and O(g(n))= f(n) 3. g(n)=O(f(n)) 4. g(n)=O(f(n))
5. g(n)=O(f(n)) and O(g(n))= f(n) 6. g(n)=O(f(n)) 7. f(n)=O(g(n))

B

Prove: g(n)=O(f(n)):

∵ $\sqrt{n} = 2\ln(n)$ $n \approx 1.3$

∴ $n > 1.3$ $2\ln(n) > \sqrt{n}$

∵ $c = 1$ $N = 2$ $\sqrt{n} \leqslant c^* 2\ln(n)$ $(n > N)$

∴ g(n)=O(f(n))

In [3]:
```
# exercise 4
n = int(input()) # c1 = 1
def harmonic_number(n): # c2 = 1

    H_n = sum(1/k for k in range(1, n+1)) # c3 = n
    return H_n # c4 = 1

print(harmonic_number(n)) # c5 = 1
# T(n) = 4*n = O(n)
```

```
6
2.4499999999999997
```

In [10]:
```
# exercise 5
def factorial(n):
    if n == 0:  # c1: 1
        return 1
    else:
        fact = 1  # c2: 1
        for i in range(1, n + 1):  # c3: n
            fact *= i  # c4: n
        return fact  # c5: 1

try:
    n = int(input("Enter a non-negative integer: "))  # c6: 1
    print(factorial(n))  # c7: 1
except ValueError:
    print("Please enter a valid integer.")  # c8: 1
# T(n) = 6+2n =O(n)
```

```
3
6
```

In [16]:
```
# exercise 6
def bubble_sort(arr):
    n = len(arr) # c1: 1
    for i in range(n):  # c2: n
        for j in range(0, n-i-1):  # c3: n(n-1)/2
            if arr[j] > arr[j+1]:  # c4: n(n-1)/2
                arr[j], arr[j+1] = arr[j+1], arr[j]  # c5: n(n-1)/2
    return arr  # c6: 1
# ∴O (n^2)
```

In [18]:
```
test_array = [64, 34, 1, 12, 221, 11, 90]
sorted_array = bubble_sort(test_array)
print(sorted_array)
```

```
[1, 11, 12, 34, 64, 90, 221]
```

In [ ]:
```
# exercise 7
# Algorithm 2:

s = 0 #c1: 1
i = 0 #c2: 1
while i < n do #c3:n+1
    s = s + 1 #c4:n
    i = i + 1 #c5:n

# ∴ O (n)
```

In [ ]:
```
# Algorithm 3:

s = 0 #c1:1
i = 0 #c2:1
while i < n do #c3: n+1
    j = 0 #c4 : n
    while j < n do # c5: n^2
        s = s + 1 #c6: n^2
        j = j + 1 #c7: n^2
    i = i + 1 #c8: n
#∴ O(n^2)
```

In [ ]:
```
# Algorithm 3:

s = 0 #c1: 1
i = 0 #c2: 1
while i < n do #c3: n+1
    j = 0 #c4: n
    while j < n do #c5: n^2
        s = s + 1 #c6: n^2
        j = j + 1 #c7: n^2
    i = i + 1 #c8: n

# ∴ O (n^2)
```

In [ ]:
```
# Algorithm 4:

s = 0         # c1: 1
i = 0         # c2: 1
while i < n do # c3: n+1
    j = 0         # c4: n
    while j < n * n do # c5: n*n*n = n^3
        s = s + 1 # c6: n^3
        j = j + 1 # c7: n^3
    i = i + 1     # c8: n
# ∴ O (n^3)
```

In [ ]:
```
# Algorithm 5:

s = 0 #c1: 1
i = 0 #c2: 1
while i < n do # c3: n+1
    j = 0 #c4: n
    while j < i do #c5: n(n-1)/2
        s = s + 1 #c6: n(n-1)/2
    i = i + 1 #c7: n

# ∴ O (n^2)
```

In [ ]:
```
# Algorithm 6:

s = 0         # c1: 1
i = 0         # c2: 1
while i < n do # c3: n+1
    j = 0         # c4: n
    while j < i do # c5: n(n-1)/2
        k = 0          # c6: n(n-1)/2
        while k < j do # c7: n(n-1)(n+1)/6
            s = s + 1 # c8: n(n-1)(n+1)/6
            k = k + 1 # c9: n(n-1)(n+1)/6
        j = j + 1     # c10: n(n-1)/2
    i = i + 1         # c11: n
# ∴ O (n^3)
```