# 1 Time complexity review

Find the (exact) number of times *print()* function executes for the following loops using n = 10 and n = 20. Next, find or estimate tight asymptotic bound of the following loops. If you can't find tight asymptotic bound find lower and upper asymptotic bound. All undefined variables used are of type integer.

**Algorithm 1**

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         Console.print(i)
4:     end for
5:     return 0
6: end function
```

**Algorithm 2**

```
1: function FUN(int n)
2:     for (int i = 1; i < n/2; i+=5) do
3:         Console.print(i)
4:     end for
5:     return n
6: end function
```

**Algorithm 3**

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         Console.print(i)
4:     end for
5:     for (int i = n; i > 0; i– –) do
6:         Console.print(i)
7:     end for
8:     return 0
9: end function
```

**Algorithm 4**

```
1: function FUN(int n)
2:     for (int i = n; i > 1; i/=2) do
3:         Console.print(i)
4:     end for
5:     return 0
6: end function
```

**Algorithm 5**

```
1: function FUN(int n)
2:     for (int i = 0; i < n/2; i++) do
3:         for (int j = 0; j < n/2; j++) do
4:             Console.print(i+j)
5:         end for
6:     end for
7:     return 0
8: end function
```

**Algorithm 6**

```
1: function FUN(int n)
2:     int m = 2000
3:     for (int i = 0; i < n/2; i++) do
4:         for (int j = 0; j < m*1000; j++) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

**Algorithm 7**

```
1: function FUN(int n)
2:     int m = n/10000
3:     for (int i = 0; i < n/2; i++) do
4:         for (int j = 0; j < m; j+=20) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

**Algorithm 8**

```
1: function FUN(int n)
2:     for (int i = 0; i < n; i++) do
3:         for (int j = i; j < n; j++) do
4:             Console.print(n)
5:         end for
6:     end for
7:     return 0
8: end function
```

**Algorithm 9**

```
1: function FUN(int n)
2:     int m = n*n;
3:     for (int i = n/2; i > 1; i/=3) do
4:         for (int j = 0; j < m; j++) do
5:             Console.print(i+j)
6:         end for
7:     end for
8:     return 0
9: end function
```

**Algorithm 10**

```
1: function FUN(int n)
2:     int m = 2147483647; //MAX_INTEGER
3:     for (int i = 0; i < m; i++) do
4:         for (int j = m; j < m; j/=2) do
5:             Console.print(n)
6:         end for
7:     end for
8:     return 0
9: end function
```

**Algorithm 11**

```
1: function FUN(int n)
2:     while n > 1 do
3:         Console.print(n)
4:         n = n / 2
5:     end while
6:     return 0
7: end function
```

**Algorithm 12**

```
1: function FUN(int n)
2:     m = 0
3:     while m * m < n + 100 do
4:         Console.print(m)
5:         m++
6:     end while
7:     return 0
8: end function
```

**Algorithm 13**

```
1: function FUN(int n)
2:     while n > 0 do
3:         for (int i = n; i > 0; i/=3) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

**Algorithm 14**

```
1: function FUN(int n)
2:     while n > 0 do
3:         for (int i = n; i > 1; i = sqrt(i)) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

**Algorithm 15**

```
1: function FUN(int n)
2:     m = n * n
3:     while m > 2 do
4:         Console.print(n)
5:         m = m / 2
6:     end while
7:     return 0
8: end function
```

**Algorithm 16**

```
1: function FUN(int n, int m)
2:     while n > 0 do
3:         for (int i = 0; i < m; i++) do
4:             Console.print(n)
5:         end for
6:         n = n / 2
7:     end while
8:     return 0
9: end function
```

**Algorithm 17** Random() returns a number between 0 and 1

```
1: function FUN(int n)
2:     int m = 0
3:     for (int i = 0; i < n; i++) do
4:         m += i
5:     end for
6:     if Random() > 0.5 then
7:         return 0
8:     end if
9:     for (int i = 0; i < m * n; i++) do
10:        Console.print("?")
11:    end for
12:    return 0
13: end function
```

**Algorithm 18**

```
1: function FUN(int n, int m)
2:     if n > m then
3:         return 0
4:     end if
5:     for (int i = 0; i < n; i++) do
6:         for (int j = i; j < m; j++) do
7:             Console.print(" :)")
8:         end for
9:     end for
10:    return 0
11: end function
```

# 2 Streaming median

Below are two streaming median algorithms. The streaming median algorithm keeps track of the median even if we add numbers to the list. Find the upper asymptotic bound of both algorithms.

Structure List consists of two functions. Function **add**(int n) adds a new element $n$ in sorted order to the structure in $\Theta(n)$. Function **get**(int i) returns $i$-th element in constant time.

Structure Heap has four functions. **Insert()** inserts new element in structure in $\Theta(log(n))$. Function **size()** returns the number of elements in heap in constant time. Function **peek()** returns minimal (or maximal) element in the structure in constant time. Functions **extractMin()** and **extractMax()** return and delete minimal/maximal element respectively in $\Theta(log(n))$ time.

**Algorithm 19**

```
Require: external List sortedNumbers
1: function findMedian(int newNumber)
2:     sortedNumbers.add(newNumber)
3:     int medianIndex = length(sortedNumbers)/2
4:     return sortedNumbers.get(medianIndex)
5: end function
```

**Algorithm 20**

```
Require: external Heap minHeap, maxHeap
1: function findMedian(int newNumber)
2:     if newNumber < maxHeap.peek() then
3:         maxHeap.insert(newNumber)
4:     else
5:         minHeap.insert(newNumber)
6:     end if
7:     if minHeap.size()+1 > maxHeap.size() then
8:         int root = minHeap.extractMin()
9:         maxHeap.insert(root)
10:    else if maxHeap.size()+1 > minHeap.size() then
11:        int root = maxHeap.extractMax()
12:        maxHeap.insert(root)
13:    end if
14:    if minHeap.size() >= maxHeap.size() then
15:        return minHeap.peek()
16:    else
17:        return maxHeap.peek()
18:    end if
19: end function
```

## 3 Sorting functions

Sort the following functions in ranks. From lowest rank to highest rank depending on asymptotic growth.

- $n$
- $nlog(n)$
- $2^{log_2(n)}$
- $2^n$
- $3^n$
- $2^{n+7}$
- $log(n^2)$
- $3\sqrt{n}$
- $n!$
- $n^2$
- $\binom{n}{2}$
- $2^{4096}$
- $n^3$
- $log(log(n))$
- $(n-1)!$
- $7n^3+5n^2-2n+13$
- $n \cdot 2^n$
- $log\left(\sqrt{log(n)}\right)$
- $n \cdot log(n^n)$