**Triangle Using Heap Sort**

**pseudocode**

**my\_len(A)**

**//Array A is passed**

**count ← 0**

**for each element in A**

**count ← count + 1**

**return count**

**MaxHeap(arr, n, i )**

**//Array arr of size n and root node of index i is passed**

largest ← i

left ←2 \* i + 1

right ← 2 \* i + 2

if left < n and arr[largest] < arr[left]:

largest ← left

if right < n and arr[largest] < arr[right]:

largest ← right

if largest !← i:

swap arr[i] with arr[largest]

MaxHeap(arr, n, largest)

**Build\_Max\_Heap(arr)**

**//Array arr is passed**

**n ← my\_len(arr)**

for i from (n/2) - 1 to 0 step -1:

**MaxHeap (arr, n, i)**

**Algorithm HeapSort (A)**

**//Array A is passed**

**Build\_Max\_Heap (A)**

n ← my\_len(A)

for i from n - 1 to 0 step -1:

swap arr[i] with arr[0]

**MaxHeap(arr, i, 0)**

**Algorithm isTriangular(A)**

**HeapSort(A)**

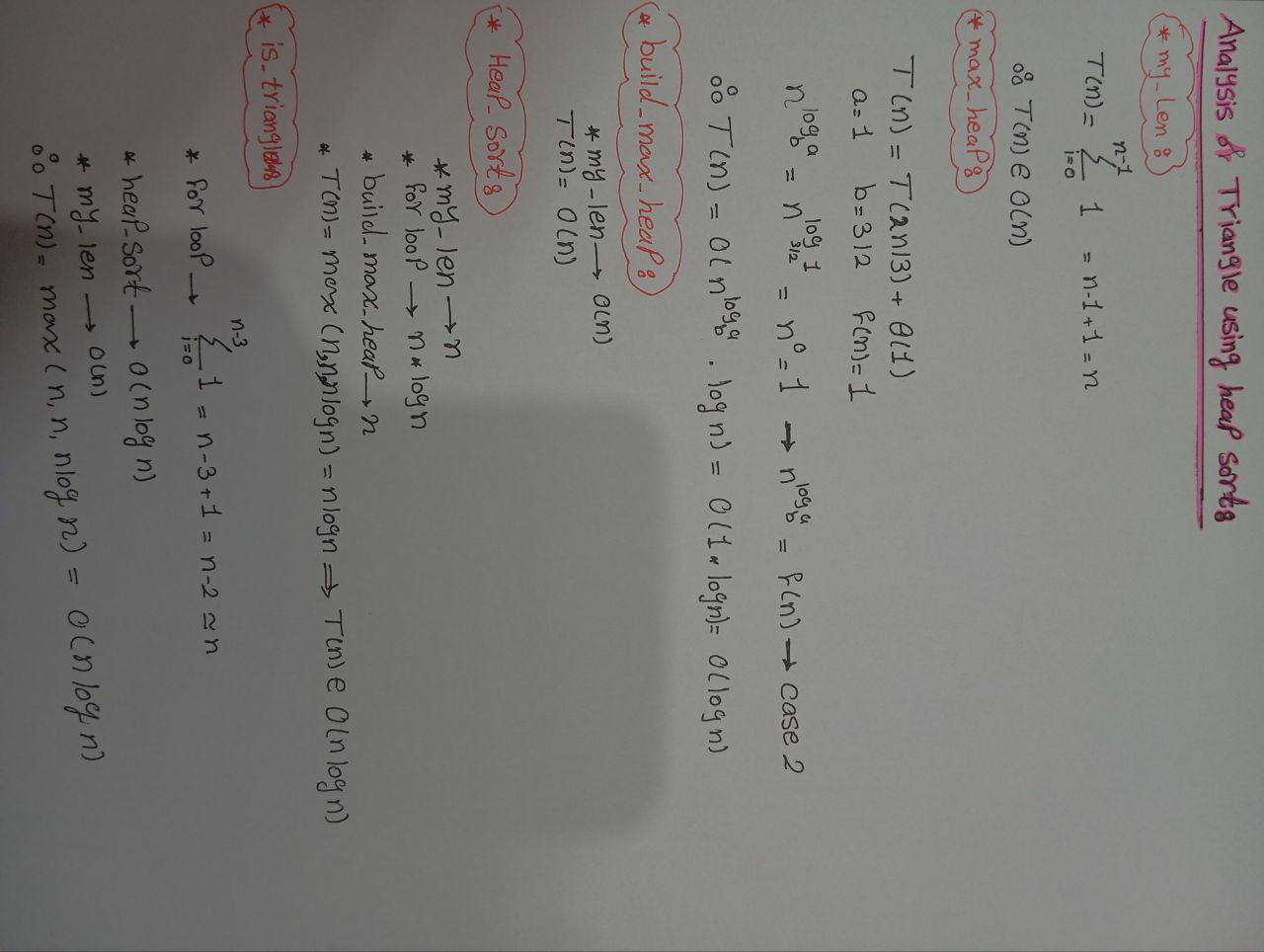
for i from 0 to length of nums – 3:

if nums[i] + nums[i + 1] > nums[i + 2] and nums[i + 1] + nums[i + 2] > nums[i] and nums[i] + nums[i + 2] > nums[i + 1]:

return 1

return 0

**Analysis**



**Triangle Using loops**

**pseudocode**

**my\_len(A)**

//Array A is passed

count ← 0

for each element in A

count ← count + 1

return count

**Algorithm is\_triangular(A)**

//Array A is passed

n ← my\_len(A)

for i ← 0 to n-1

for j ← i+1 to n-1

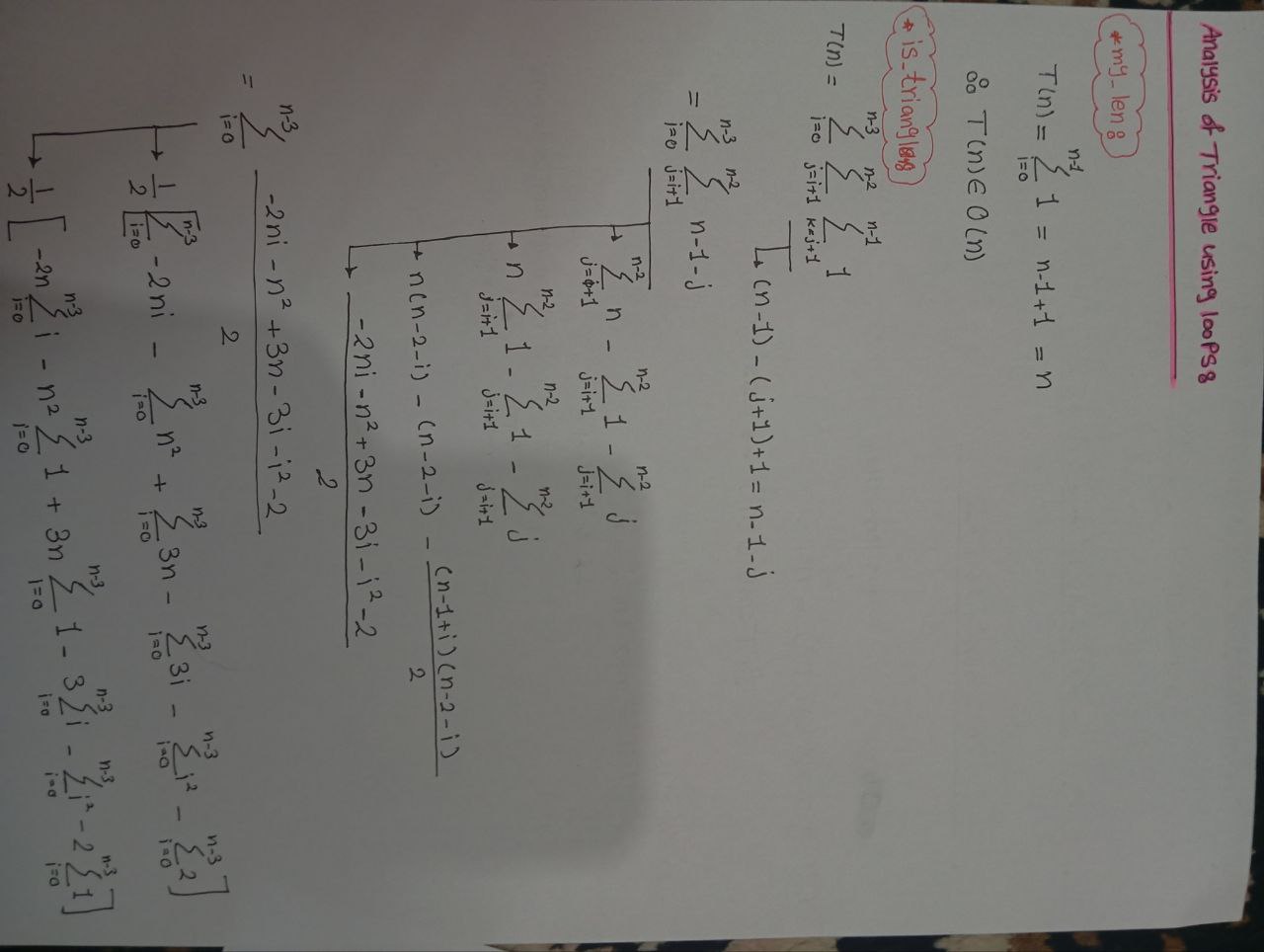
for k ← j+1 to n-1

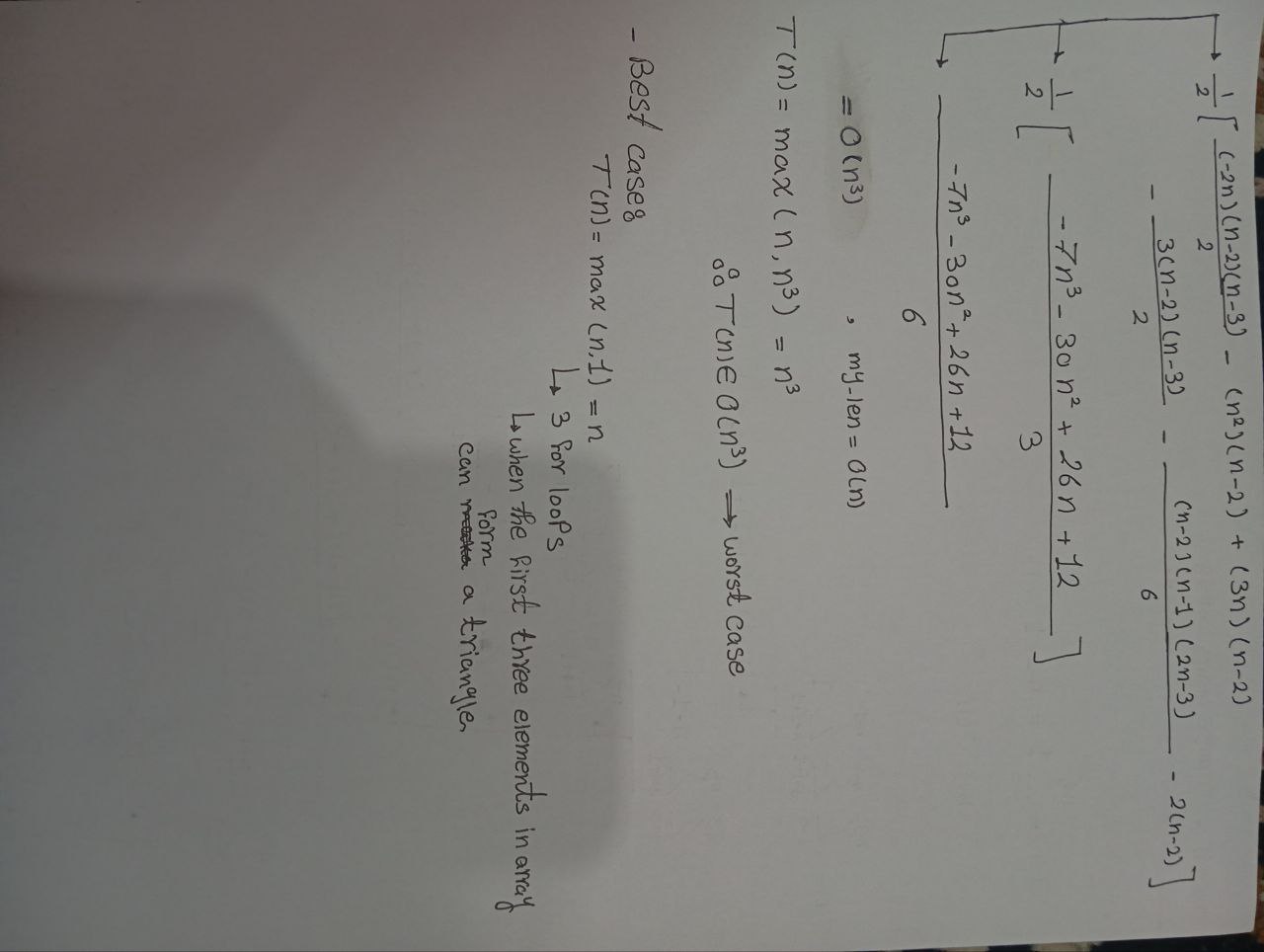
if A[i] + A[j] > A[k] and A[i] + A[k] > A[j] and A[j] + A[k] > A[i]

return 1

return 0

**Analysis**





**Triangle Using Insertion Sort**

**Pseudocode**

**my\_len(A)**

//Array A is passed

count ← 0

for each element in A

count ← count + 1

return count

**Insertion\_Sort(A,n)**

//Array A of size n is passed

n ← my\_len(A)

for j ← 1 to n

key ← A[j]

i ← j - 1

while i >= 0 and A[i] > key

do

A[i+1] = A[i]

i ← i-1

A[i+1] = key

return A

**Algorithm is\_triangular(A)**

//Array A of size n is passed

Insertion\_Sort(A)

n ← my\_len(A)

for i ← to n-2

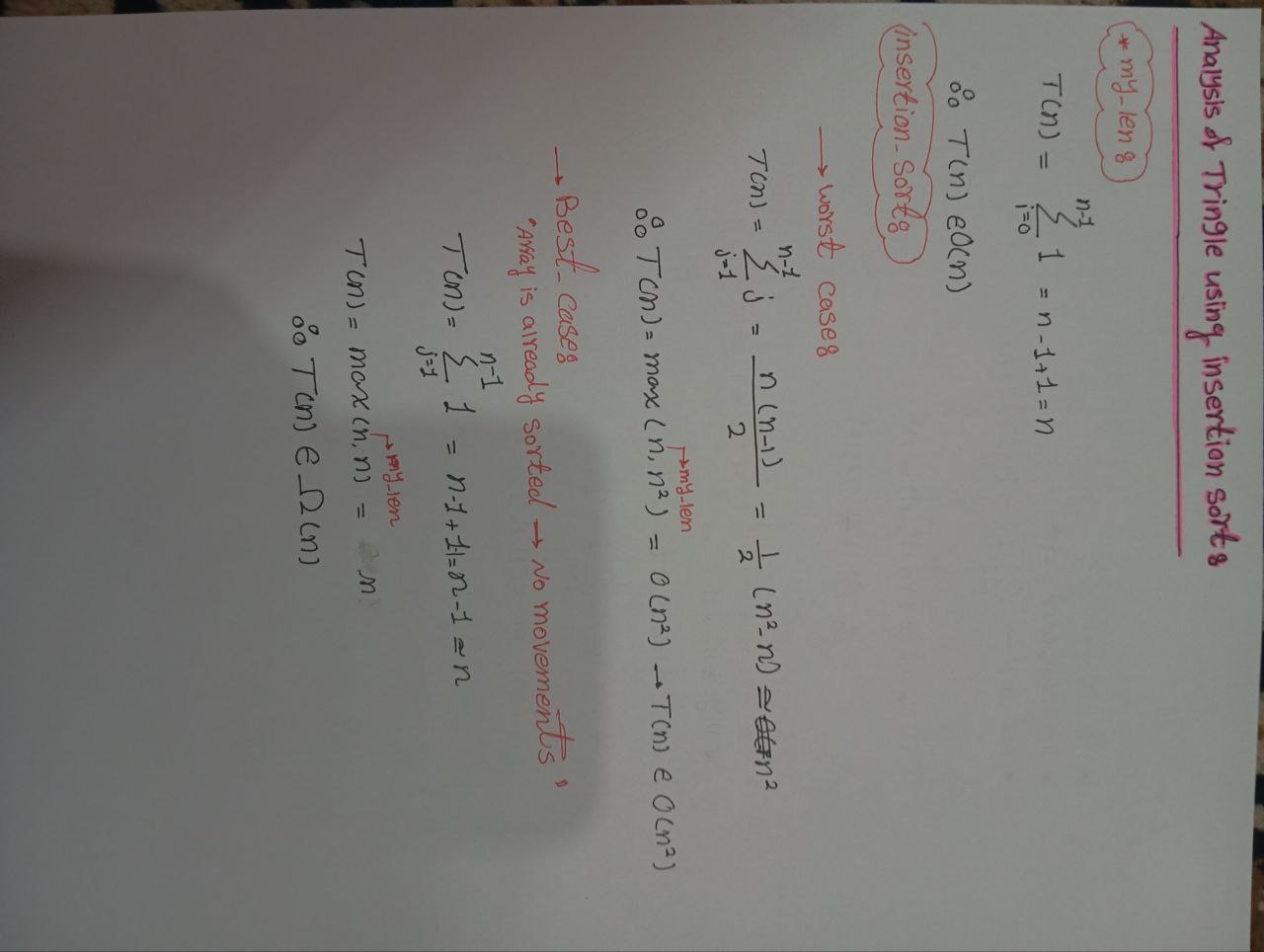
if A[i] + A[i + 1] > A[i + 2]

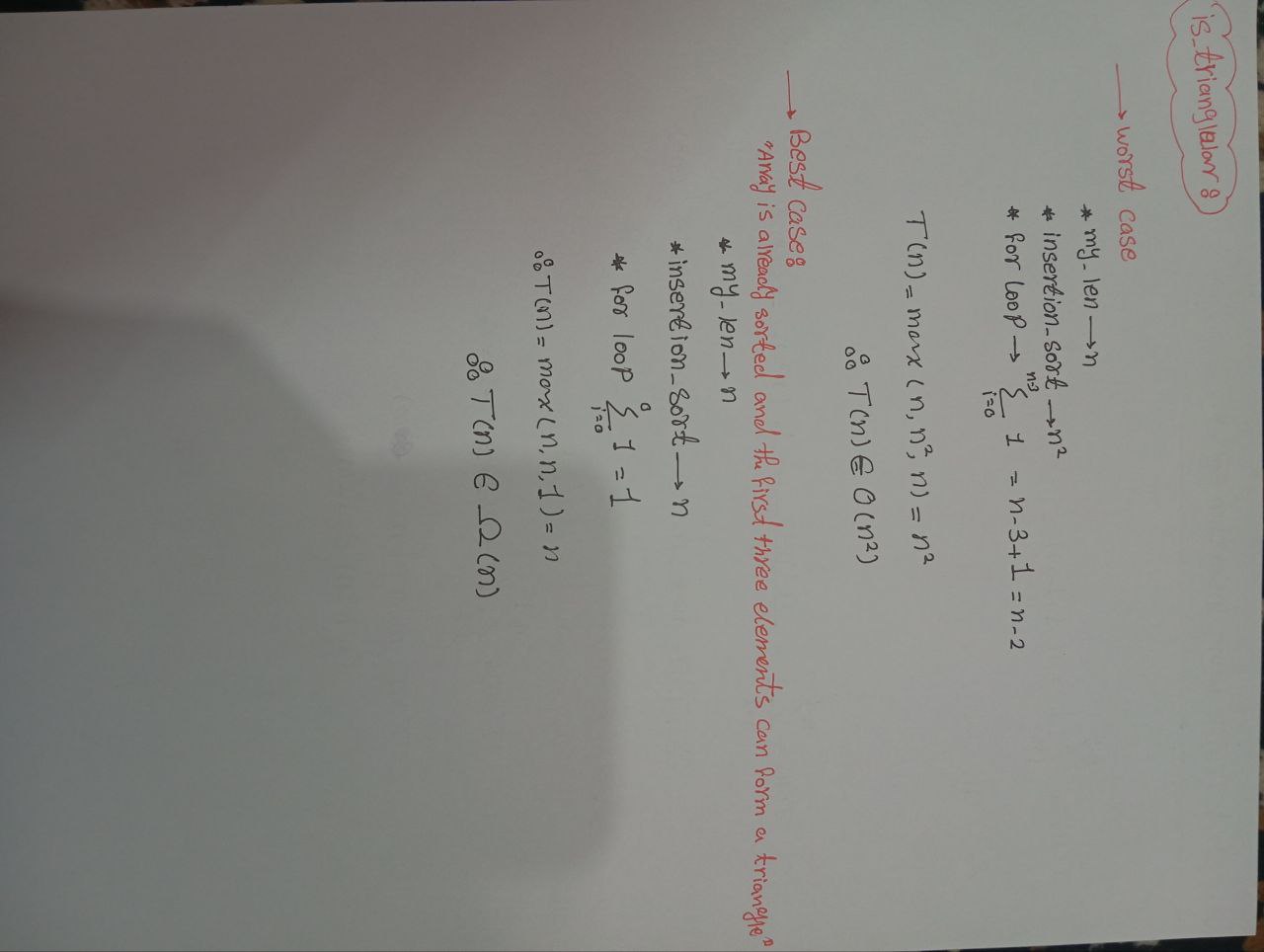
return 1

else

return 0

**Analysis**





**Comparison**

A screenshot of a computer

Description automatically generated

* Space complexity is the same for the 3 algorithms.
* Best case Time complexity in 1st & 2nd algorithms are less than the 3rd algorithm.
* Worst case time complexity in the 3rd algorithm is less than the 1st & 2nd algorithm.

Therefore, according to the Worst-Case time complexity the 3rd algorithm is the best algorithm.