**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**

A notebook with writing on it

Description automatically generated

**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

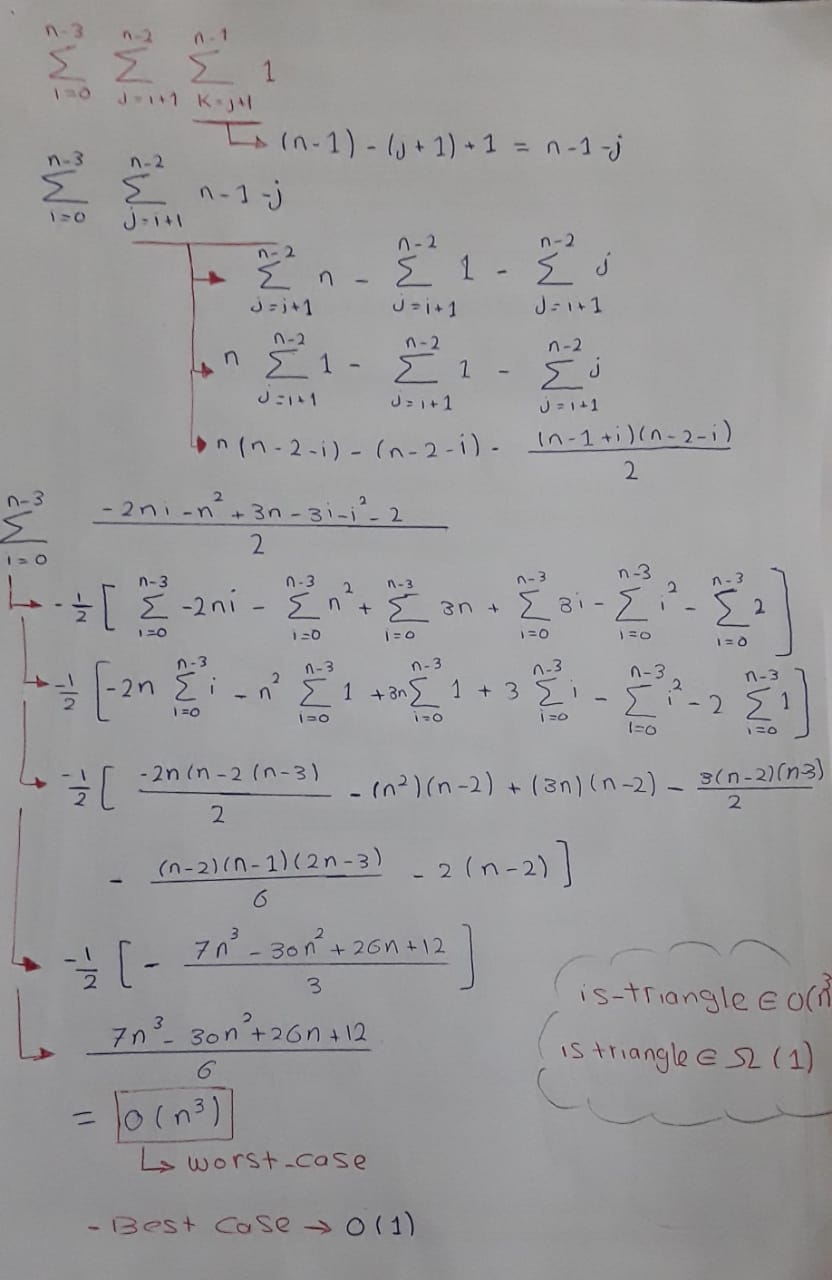
for k ← j+1 to n-1

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**



**Comparison**

|  |  |  |
| --- | --- | --- |
| P.O.C | Loops | Heap Sort |
| Best Case | O(1) | O(nlogn) |
| Worst Case | O(n3) | O(nlogn) |

According to Best Case Loops are better.

According to Worst Case Heap Sort is better.