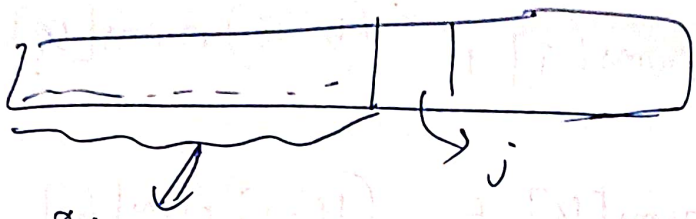
This can be done using nested for loop $O(N^2)$ complexity



I have answers for these positions ans[i]

$ans[j] = max(ans[i] \& (j-i)*nums[j])$ $\forall$ i from [i to j-1]

Is there any way to calculate this faster than O(N)?

[ans[i], (j-i)] → we cannot maintain this as the sort order. As nums[j] is differ for all indexes.

Jump i → to → K

option 1 → i → j, j → K

nums[j](j-i) + nums[K](K-j)

option 2 → i to K direct

(K-i) nums[K]

forming

$(K-j)*nums[K] + (j-i)*nums[K]$

P.T.O

Compare both options.

$i \to j \to k \to (j-i)\,nums[j] + (k-j)\,nums[k]$

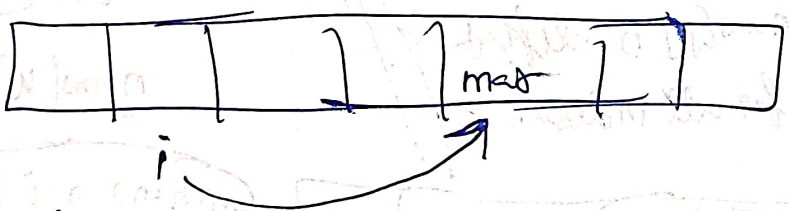
$i \to k \to (k-i)\,nums[k] + (k-j)\,nums[k]$.

well case 1 $nums[k] > nums[j]$.

Its better to jump directly to k

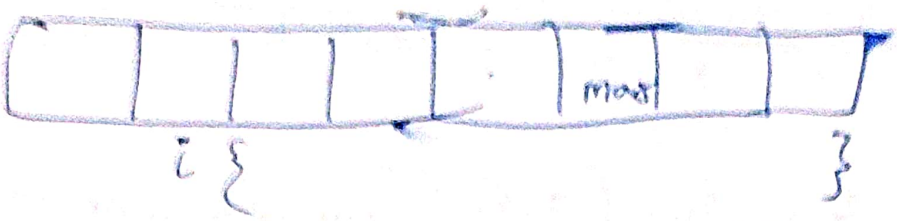
Case 2 $nums[j] > nums[k]$

Its better to jump ~~direct~~ to j first and then k. So you pick the benifit of nums[j] over nums[k] and don't miss out on it



In brief, if you are at position i, its better to jump to a position. (p). where nums[p] is the max on the right hand side of i.

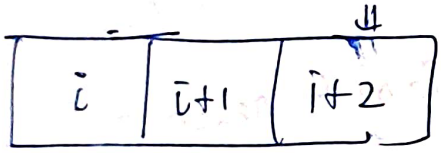
So ~~Ref~~ lets consider how you can code this.



Maybe use a priority queue string elements in the suffix, or a tree set → of (nums[i], i) descending order to get max

~~But the~~ O(logN). So it optimisation.

Next further optimisation..

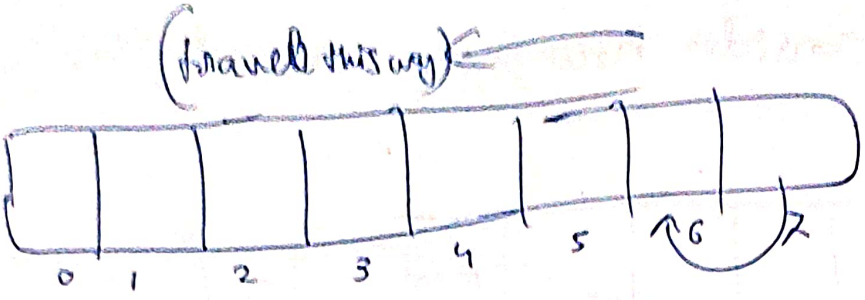


Lets say i+2 is the last cell, and you have to jump from i → i+1
↘ i+2

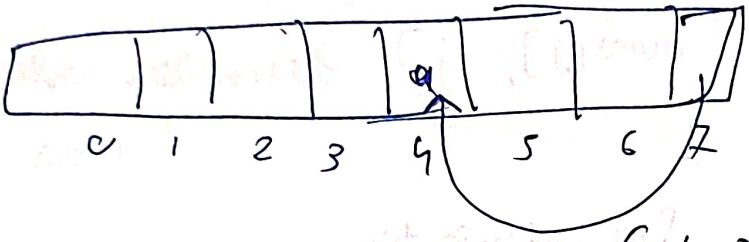
You will jump to the cell i+2 first if nums[i+2] > nums[i+1]
else jump to nums[i+1] then nums[i+2]

So why dont you travel from last cell to the front,
You need to keep track of max on the suffix right?
So travel from (⟵) Right to left and ~~keep~~ for
each cell you travell ~~next~~ left ~~max~~ add it to your score

P.T.O.



to jump from (7 → 6) obviously nums[7] will be score



Jump from (7 to 4)

if nums[7] is maximum in that range

the score(4) = (7-4) nums[7]

= 3 · nums[7].

So just keep a max variable → (score to reach 0).

sum = 0. max = nums[7]

for (i = len-2; i >= 0; i--) {

sum += max

max = Math.max(max, nums[i])

}

so at (i=0)

we will get the max sum.