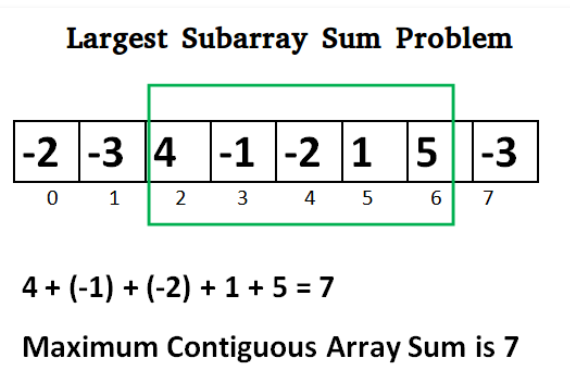
**Kadane’s algorithm**

This algorithm works for a set of positive numbers.

It is used to find the largest sum contiguous sub array from an array.



**Input:** nums = [-2,1,-3,4,-1,2,1,-5,4]  
**Output:** 6  
**Explanation:** [4,-1,2,1] has the largest sum = 6.

Pseudo code

Initialise

Max\_so\_far = 0;

Max\_till\_now = 0;

For(int i=0; i < a.length; i++){

Max\_till\_now = Max\_till\_now + a[i];

If(Max\_till\_now < 0) {

Max\_till\_now = 0;

}

If(Max\_so\_far < Max\_till\_now) {

Max\_so\_far = Max\_till\_now;

}

}

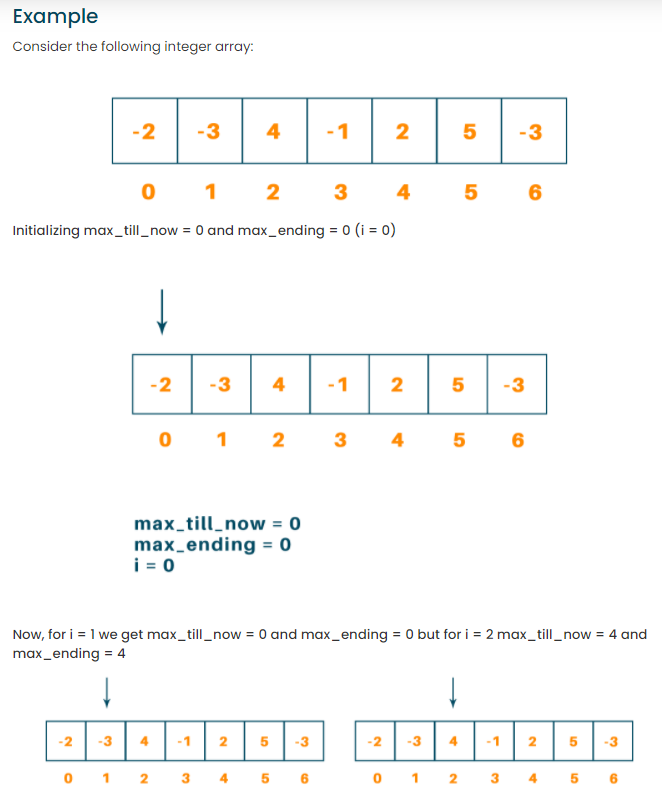
Time complexity = O(n)

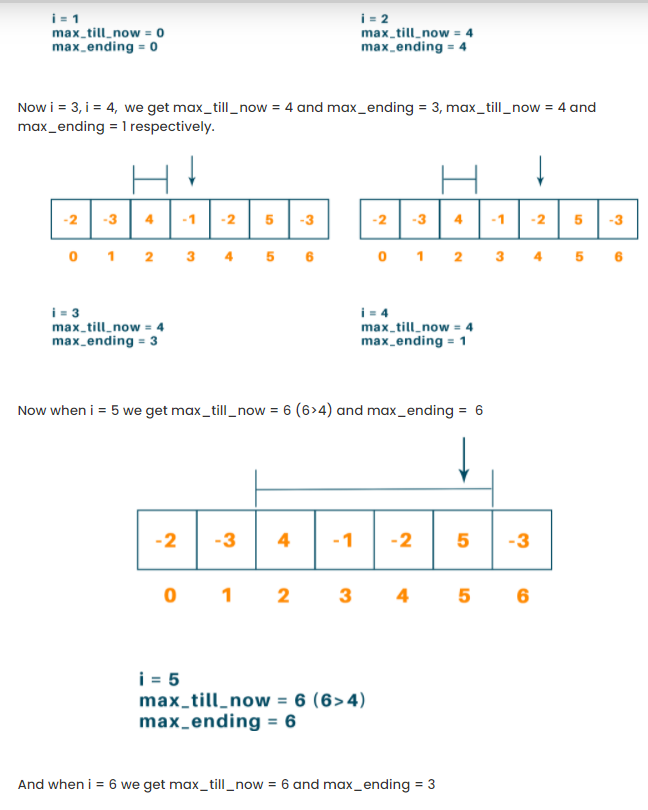
Use case / test case for Kadanes Algorithm

There are many applications of kadane’s algorithm and some of them are as mentioned below:

* Finding maximum subarray sum for a given array of integer
* Used as an image processing algorithm
* It can be used to solve the problems like “Station Travel in Order” and “Hotels Along the Coast”
* It is used for business analysis

The simple idea of Kadane’s algorithm is to look for all positive contiguous segments of the array (max\_till\_now is used for this). And keep track of maximum sum contiguous segment among all positive segments (max\_so\_far is used for this). Each time we get a positive-sum compare it with max\_so\_far and update max\_so\_far if it is greater than max\_till\_now



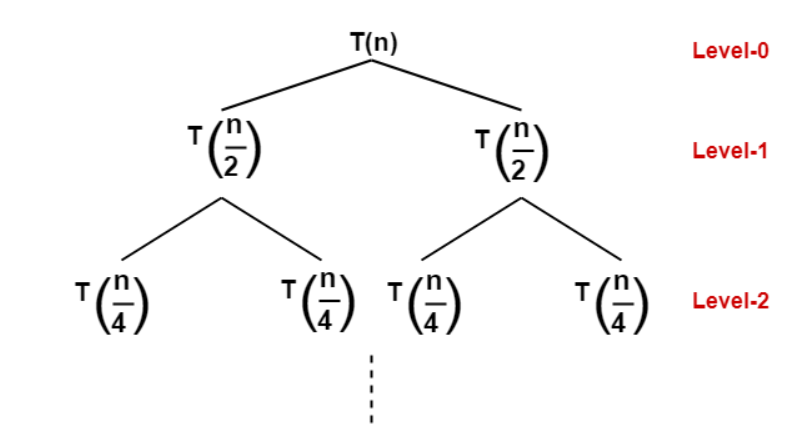
  
  
This is a short explanation for the working of Kandane’s algorithm.

**BINARY SEARCH**

It is only applicable for a sorted array.

It follows a recirsive way to search for a number in an array. This process allows it to have a time complexity of OlogN.

Time complexity of recursive function



***Cost at each level***

The cost of dividing and combining the solution of size N/2^k is N/2^k and the number of nodes at each level is 2^k.

So, the cost at each level is (N/2^k)\*(N^k) = N.

Size of problem at level-0 : N/2^0

Size of problem at level-1 : N/2^1

Size of problem at level-2 : N/2^2

Size of problem at level-3 : N/2^3

.

.

.

.

Size of problem at level-k : N/2^k

Let us suppose the last level be k. Then

N/2^k = 1

N = 2^k

Taking log both sides

**log**(N) = log(2^k)

log(N) = k\*log(2)

k = log2(N)

The number of levels in the recursion tree is **log2(N).**

In case of binary search tree

Best time complexity is O(1)

Worst time complexity is O(nlogn).

Example of binary tree

10, 14, 19, 26, 27, 31, 33, 35, 42, 44

Find if 31 exists in the given array.

Pseudo code

Procedure binary\_search

A ← sorted array

n ← size of array

x ← value to be searched

Set lowerBound = 1

Set upperBound = n

while x not found

if upperBound < lowerBound

EXIT: x does not exists.

set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

if A[midPoint] < x

set lowerBound = midPoint + 1

if A[midPoint] > x

set upperBound = midPoint - 1

if A[midPoint] = x

EXIT: x found at location midPoint

end while

end procedure

Pseudo code analysis

A -> 10, 14, 19, 26, 27, 31, 33, 35, 42, 44

N -> 10

X -> 31

lowerBound -> 0

upperBound -> 10

Step 1 –

While(31 not found)

I -> 0

10 < 1 ---- false

midpoint = 0 + (9-0)/2 = 4

if(26 < 31) --- true

lowerBound = 4 + 1 = 5  
 upperBound = 10

Step 2

While(31 not found)

10 < 5 ---- false

midpoint = 5 + (9-5)/2 = 7

if(35 < 31) --- false

if(35 > 31) – true

upperBound = 6

lowerBound = 5

Step 3

While(31 not found)

6 < 5 ---- false

midpoint = 5 + (6-5)/2 = 5

if(33 < 33) --- false

if(33 > 34) – false

if(33 = 33) – true

Number exists in the array.

Binary search usage –

The way in which we view dictionary is based on this algorithm.

Many people use binary searches **from childhood** without being aware of it. For example, when you search for words in a dictionary, you don't review all the words; you just check one word in the middle and thus narrow down the set of remaining words to check.