

MAXIMUM SUB- ARRAY SUM

UNDERSTANDING THE PROBLEM

You are given an Array[] with n elements. You need to find the Maximum sum of Subarray.

Among all Subarrays of that array. A subarray of array A[] of length n is a contiguous segment from A[i] through A[j] where $0 \leq i \leq j \leq n$

- ➡ if the array contains all non-negative numbers the maximum subarray is the entire array
- ➡ Several different sub. array may have the same maximum
- ➡ fun fact: this problem asked in Amazon, Facebook Microsoft

EXAMPLES:

Example 1:

Input: A[] = (-5, 8, 9, -6, 10, -15, 3)

Output: 21, the subarray {8, 9, -6, 10} has the maximum sum among all subarrays

Example 2:

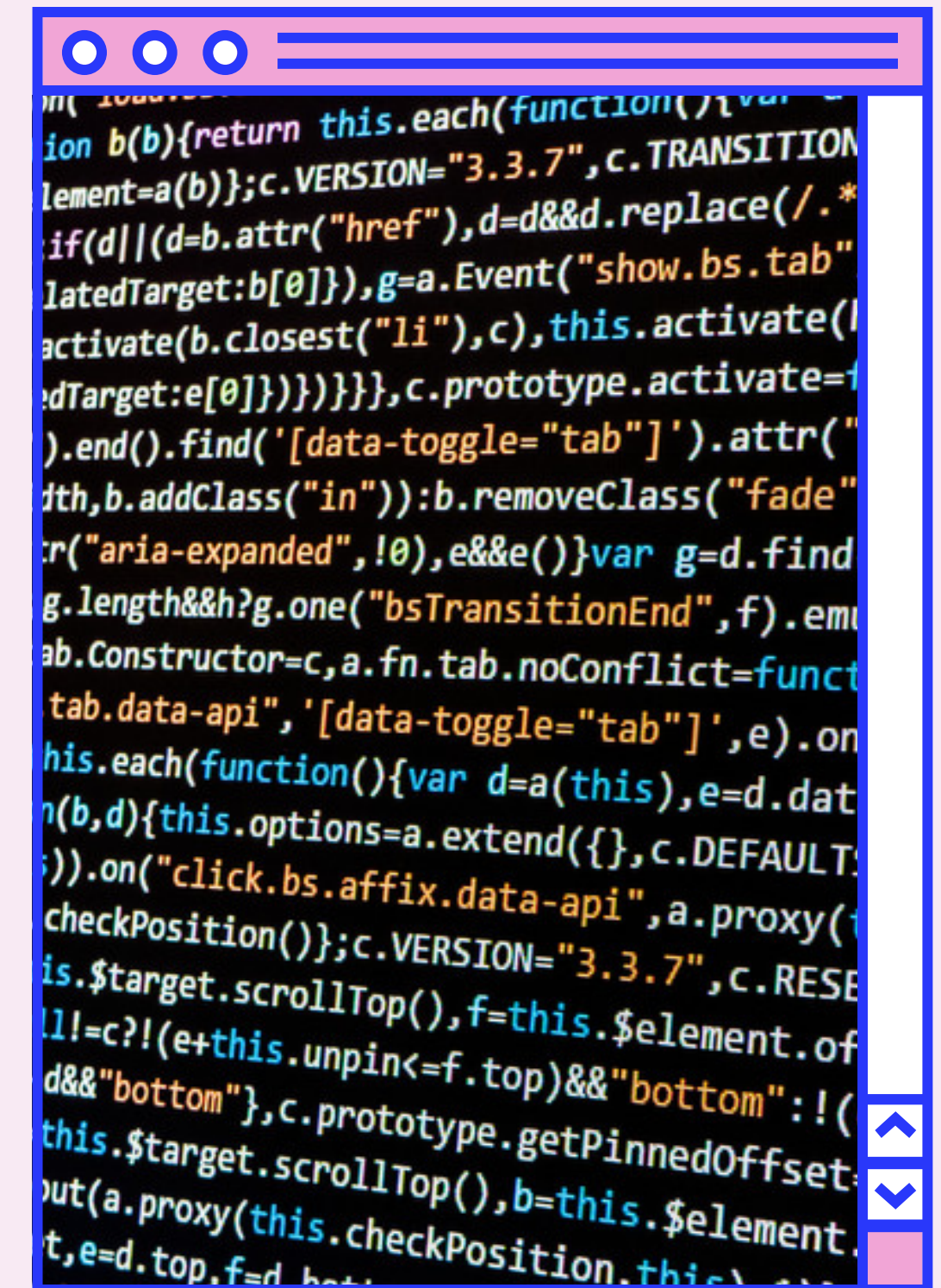
Input: A[] = {-4, -7, -1, 5, -27}


Output: 4, the subarray {-1, 5} has the maximum sum

Example 3:

Input: X[] = [-4, 5, 7, -6, 10, -15, 3], Output: 16

Explanation: The subarray [5, 7, -6, 10] has the maximum sum






```
int max_sum = 0

for (i = 0 to n-1){
  for ( j = i to n-1) {
    int sum = 0
    for ( k = i to j) {
      Sum = sum + Array[k]

    if ( sum > max_sum)
      max_sum = sum
    }
  }
}

return max_sum
}
```



BRUTE FORCE APPROACH I

algorithm that uses 3 nested loops, the first for loop to determine the starting point of the sub-array, the second for the for loop to determine the end point of the sub-array, and the last for loop to compute the sum between the starting point and ending point and compare the outputs to get the largest value.

The problem with this algorithm is that it cannot deal with an array of large size, a direct or positive relationship between the size of the array and the time it will take to show the result (an increase in the size of an array causes an increase in execution time).

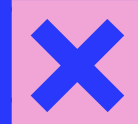


TIME COMPLEXITY

<code>int max_sum = 0</code>	$O(1)$
<code>for (i = 0 to n-1)</code>	$O(n)$
<code>for (j = i to n-1)</code>	$O(n)$
<code>int sum = 0</code>	$O(1)$
<code>for (k = i to j)</code>	$O(n)$
<code>Sum = sum + Array[k]</code>	$O(1)$
<code>if (sum > max_sum)</code>	$O(1)$
<code>max_sum = sum</code>	$O(1)$
<code>return max_sum</code>	$O(1)$
worst case complexity of the code	$O(n^3)$

Brute force approach II

So The idea is to start at all positions in the array and calculate running sums. The outer loop picks the beginning element, the inner loop finds the maximum possible sum with the first element picked by the outer loop and compares this maximum with



```
brute_force11 (arr, n)

max sum = 0

for (j= 0 to n-1)

sum=0

for(i=j to n-1)

sum= sum+ arr[j]

if (sum > max_sum)

max sum = sum

return max sum
```



TIME COMPLEXITY

<pre>max_sum = 0</pre>	$O(1)$
<pre>for i in range(0, len(arr)):</pre>	$O(n)$
<pre> sum = 0</pre>	$O(1)$
<pre> for j in range(i, len(arr)):</pre>	$O(n)$
<pre> sum += arr[j]</pre>	$O(1)$
<pre> max_sum = max(sum, max_sum)</pre>	$O(1)$
<pre>return max_sum</pre>	$O(1)$
worst case complexity of the code	$O(n^2)$

DYNAMIC PROGRAMMING APPROACH USING AN AUXILIARY ARRAY

Auxiliary

data structure is a fancy way and logical replication of data from one or more columns of a table.

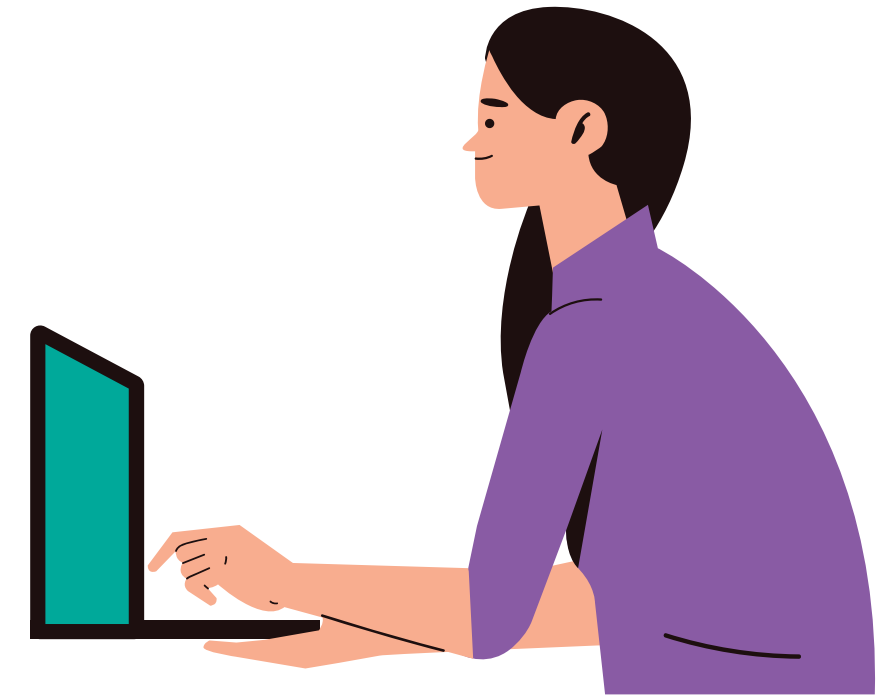
Auxiliary

data structures contain copies of, references to, or data computed from base table data, deleting data from an auxiliary data structure does not result in the loss of base table data. They are therefore derived data.

Auxiliary array (aka, prefix sum) : is a simple yet powerful technique that allows for the fast calculation of sums of elements in given contiguous segments of array, is a very vital tool in competitive programming. thus reduces the time complexity of our program

Applications:

- Evaluation of Arithmetic Expressions
- Sorting data
- Reverse a Data




```

MAXSUBARRAYSUM( A, N)
MAX_ENDING[N]
MAX_ENDING[0]= A[0]
FOR(I-1 TO N-1 ){
  IF (A[I] + MAX_ENDING[I-1] >0)
    MAX_ENDING[I]= A[I] + MAX_ENDING[I-1]
  ELSE
    MAX_ENDING[I]= A[I]}
ANS = 0
FOR(I = 0 TO N-1 )
  ANS = MAX(ANS, MAX_ENDINGL[I])

```

```

,r.resetText||n.data("resetText",n[i]()),n[i](r[e]||this
attr(t,t):n.removeClass(t).removeAttr(t)},0)},t.prototy
radio"]');e&&e.find(".active").removeClass("active"),th
on(n){return this.each(function(){var r=e(this),i=r.dat
n=="toggle"?i.toggle():n&&i.setState(n)}}},e.fn.button.
n.noConflict=function(){return e.fn.button=n,this},e(do
(t){var n=e(t.target);n.hasClass("btn")||(n=n.closest("
=function(t,n){this.$element=e(t),this.$indicators=this
over"&&this.$element.on("mouseenter",e.proxy(this.paus
ion(t){return t||(this.paused=!1),this.interval&&clearI
interval=setInterval(e.proxy(this.next,this),this.option
nt.find(".item.active"),this.$items=this.$active.parent
this.getActiveIndex(),r=this;if(t>this.$items.length-1|
r.to(t)):n==t?this.pause().cycle():this.slide(t>n?"nex
his.$element.find(".next, .prev").length&&e.support.tra
cle(!0)),clearInterval(this.interval),this.interval=nul
},prev:function(){if(this.sliding)return;return this.sl
ve"),i=n||r[t](),s=this.interval,o=t=="next"?left:"ri
i=i.length?i:this.$element.find(".item")[u](),f=e.Event
his.$indicators.length&&(this.$indicators.find(".active
dicators.children()[a.getActiveIndex()]));t&&t.addClass(
s.$element.trigger(f);if(f.isDefaultPrevented())return;
(e.support.transition.end,function(){i.removeClass([t,o
")),a.sliding=!1,setTimeout(function(){a.$element.trig
turn;r.removeClass("active"),i.addClass("active"),this.
ar n=e.fn.carousel;e.fn.carousel=function(n){return thi
carousel.defaults,typeof n=="object"&&n),o=typeof n=="s
to(n):o?i[o]():s.interval&&i.pause().cycle()}}},e.fn.ca
structor=t,e.fn.carousel.noConflict=function(){return e
],[data-slide-to]",function(t){var n=e(this),r,i=e(n.a
=e.extend({},i.data(),n.data()),o;i.carousel(s),(o=n.at
Default()))}(window.jQuery),!function(e){"use strict";v
se.defaults,n),this.options.parent&&(this.$parent=e(thi
tor:t,dimension:function(){var e=this.$element.hasClass
his.transitioning||this.$element.hasClass("in"))return;
is.$parent.find("> .accordion-group > .in");if(r&&r.len
("hide"),i||r.data("collapse",null)}this.$element[t](0)
his.$element[t](this.$element[0][n])),hide:function(){v
nension(),this.reset(this.$element[t]()),this.transitio
ion(e){var t=this.dimension();return this.$element.rem
ull?"addClass":"removeClass"]("collapse"),this},transit
positioning=0,i.$element.trigger(r):this.$element.trigg

```



TIME COMPLEXITY

<code>max_ending[n]</code>	$O(1)$
<code>max_ending[0] = A[0]</code>	$O(1)$
<code>for (i=1 to n-1)</code>	$O(n)$
<code>if (A[i] + max_ending[i-1] > 0)</code>	$O(1)$
<code>max_ending[i] = A[i] + max_ending[i-1]</code>	$O(1)$
<code>max_ending[i] = A[i]</code>	$O(1)$
<code>ans=0</code>	$O(1)$
<code>for(i=0 to n-1)</code>	$O(n)$
<code>ans = max(ans, max_ending[i])</code>	$O(1)$
worst case complexity of the code	$O(n)$

KADANE'S ALGORITHM

an Efficient Approach that attempt to improve the previous algorithms by merge the advantages of the Dynamic Programming using auxiliary array and the brute Force approach and Dynamic Programming approach Using an auxiliary array

```
maxSubarraySum(A, n)
max_so_far = 0, max_ending_here = 0
for (i=1 to n)
    max_ending_here += A[i]
    if( max_ending_here < 0)
        max_ending_here = 0
return max_so_far
```



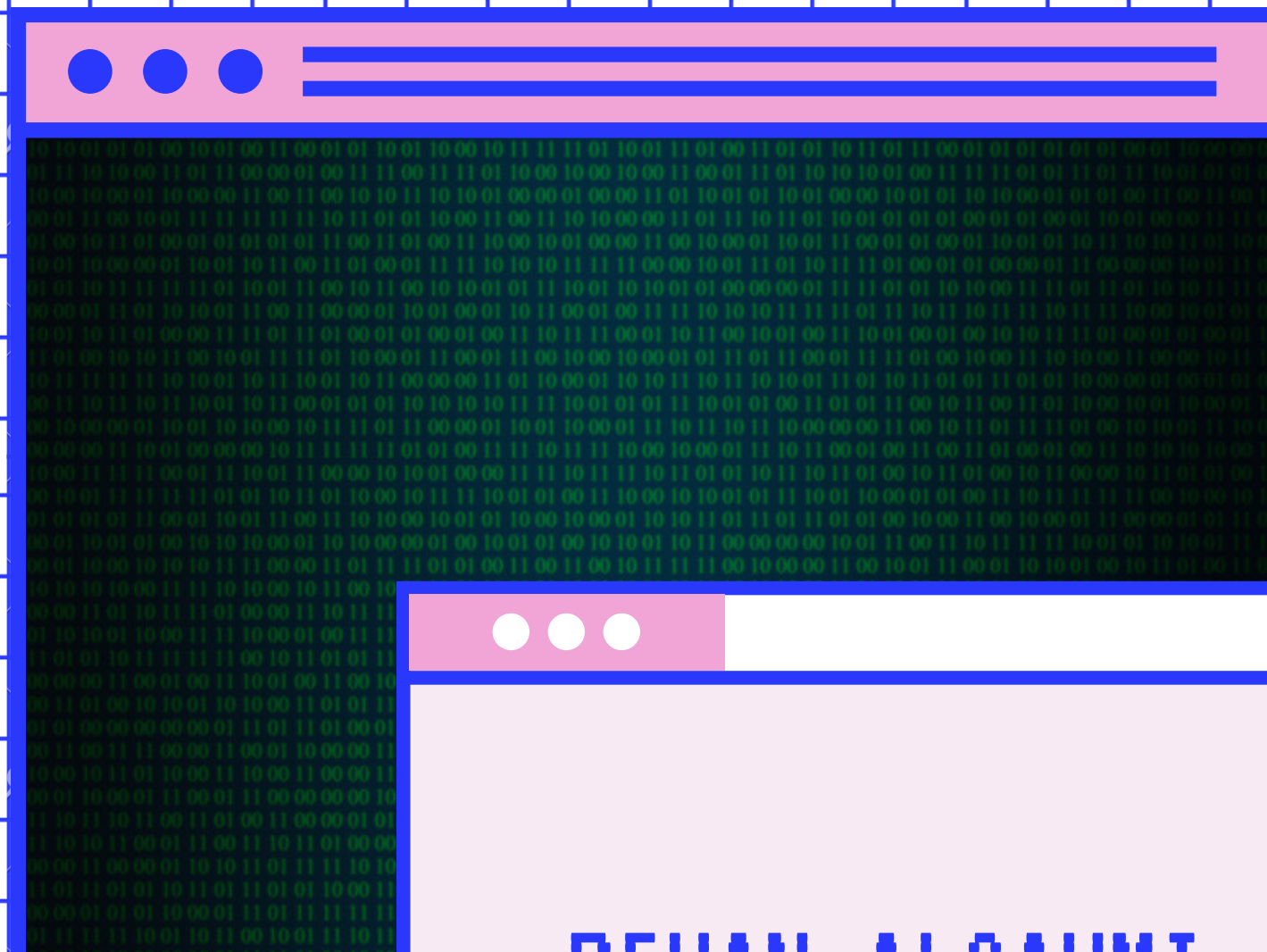


TIME COMPLEXITY

<code>max_so_far = 0</code>	$O(1)$
<code>max_ending_here = 0</code>	$O(1)$
<code>for (i=0 to n-1)</code>	$O(n)$
<code>max_ending_here += A[i]</code>	$O(1)$
<code>if(max_ending_here < 0)</code>	$O(1)$
<code>max_ending_here= 0</code>	$O(1)$
<code>return max_so_far</code>	$O(1)$
worst case complexity of the code	$O(n)$

ALGORITHM	TIME COMPLEXITY	SPACE COMPLEXITY
BRUTE FORCE APPROACH I	$O(n^3)$	$O(1)$
BRUTE FORCE APPROACH II	$O(n^2)$	$O(1)$
DYNAMIC PROGRAMMING USING AUXILIARY ARRAY	$O(n)$	$O(n)$
KADANE	$O(n)$	$O(1)$

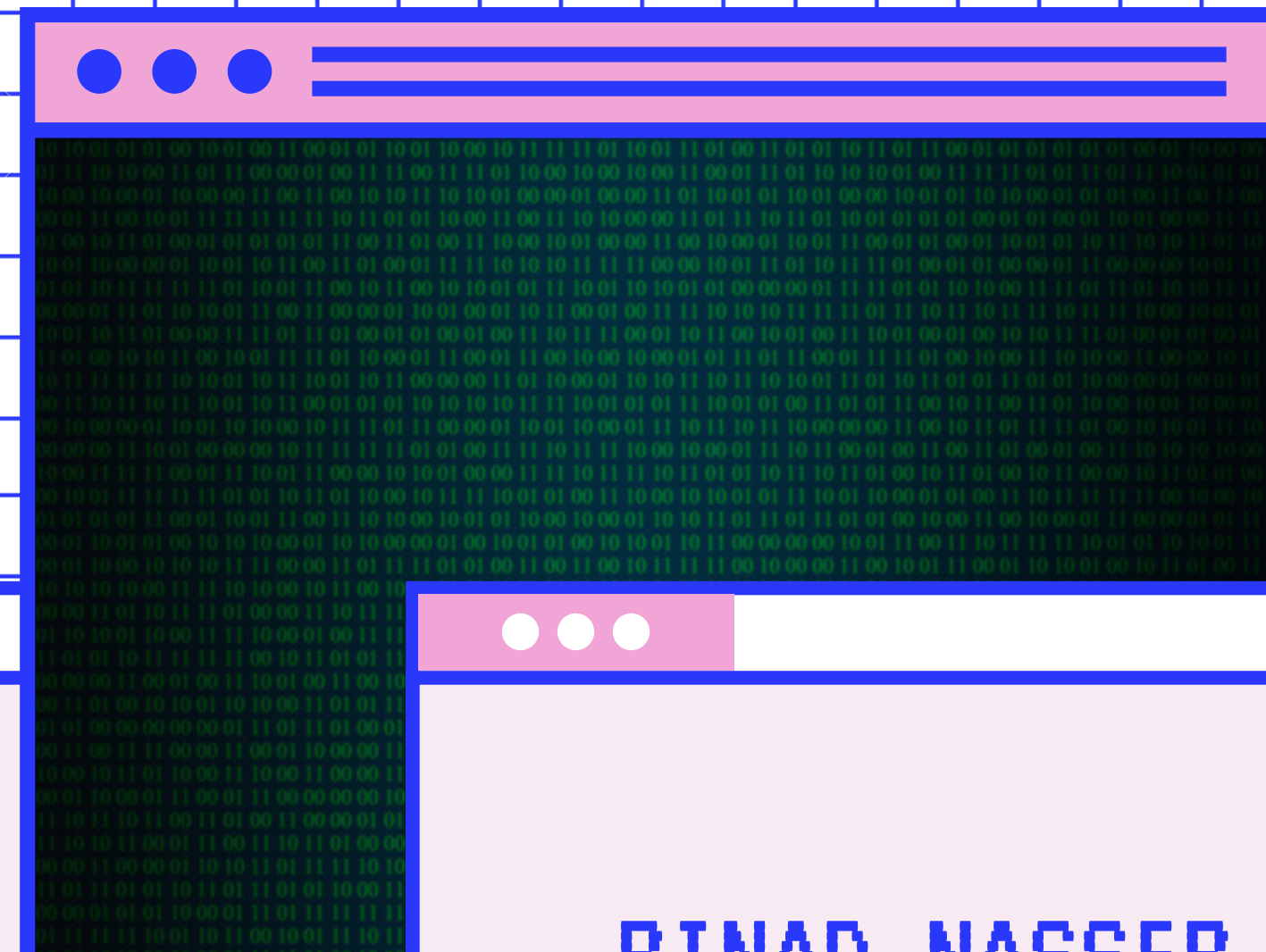
Comparison of Solutions



● ● ●

REVAN ALQAHMI

NORAH ALSALEM



● ● ●

RINAD NASSER

RANIYA MASSOUD