Largest Rectangular Area in a Histogram - GeeksforGeeks

Source: https://www.geeksforgeeks.org/largest-rectangle-under-histogram/

Largest Rectangular Area in a Histogram Last Updated : 15 Sep, 2025 Given an array arr[] representing a histogram, where each element denotes the height of a bar and every bar has a uniform width of 1 unit, find the largest rectangular area that can be formed within the histogram. The rectangle must be formed using contiguous bars. Example: Input: arr[] = [60, 20, 50, 40, 10, 50, 60] Output: 100 Explanation : We get the maximum area by picking bars highlighted above in green (50, and 60). The area is computed (smallest height) * (no. of the picked bars) = 50 * 2 = 100. Input: arr[] = [3, 5, 1, 7, 5, 9] Output: 15 Explanation : We get the maximum are by picking bars 7, 5 and 9. The area is computed (smallest height) * (no. of the picked bars) = 5 * 3 = 15. Try it on GfG Practice Table of Content [Naive Approach] By Finding Max Area of Rectangles all Heights - O(n^2) Time and O(1) Space [Expected Approach] Using Stack - O(n) Time and O(n) Space [Optimized Approach] Using Single Stack - O(n) Time and O(n) Space [Alternate Approach] Using Divide and Conquer - O(n log(n)) Time and O(n) Space [Naive Approach] By Finding Max Area of Rectangles all Heights - O(n 2) Time and O(1) Space The idea is to fix each bar as the height of the rectangle and expand towards the left and right while the bars are at least as tall as the current bar. For every valid step, we keep adding the current bar's height to the area. By doing this for all bars and keeping track of the maximum value. C++ #include <iostream> #include <vector> using namespace std ; int getMaxArea (vector < int > & arr){ int res = 0 , n = arr . size (); // Consider every bar one by one for (int i = 0 ; i < n ; i ++){ int curr = arr [i]; // Traverse left while we have a greater height bar for (int j = i -1 ; j >= 0 && arr [j] >= arr [i]; j --) curr += arr [i]; // Traverse right while we have a greater height bar for (int j = i + 1 ; j < n && arr [j] >= arr [i]; j ++) curr += arr [i]; res = max (res , curr); } return res ; } int main () { vector < int > arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; cout << getMaxArea (arr); return 0 ; } C #include <stdio.h> int getMaxArea (int arr [], int n) { int res = 0 ; // Consider every bar one by one for (int i = 0 ; i < n ; i ++) { int curr = arr [i]; // Traverse left while we have a greater height bar for (int j = i - 1 ; j >= 0 && arr [j] >= arr [i]; j --) { curr += arr [i]; } // Traverse right while we have a greater height bar for (int j = i + 1 ; j < n && arr [j] >= arr [i]; j ++) { curr += arr [i]; } if (curr > res) { res = curr ; } } return res ; } int main () { int arr [] = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , getMaxArea (arr , n)); return 0 ; } Java class GfG { static int getMaxArea (int [] arr) { int res = 0 , n = arr . length ; for (int i = 0 ; i < n ; i ++) { int curr = arr [i] ; // Traverse left while we have a greater height bar for (int j = i - 1 ; j >= 0 && arr [j] >= arr [i] ; j --) curr += arr [i] ; // Traverse right while we have a greater height bar for (int j = i + 1 ; j < n && arr [j] >= arr [i] ; j ++) curr += arr [i] ; res = Math . max (res , curr); } return res ; } public static void main (String [] args) { int [] arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; System . out . println (getMaxArea (arr)); } } Python def getMaxArea (arr): res = 0 n = len (arr) for i in range (n): curr = arr [i] # Traverse left while we have a greater height bar j = i - 1 while j >= 0 and arr [j] >= arr [i]: curr += arr [i] j -= 1 # Traverse right while we have a greater height bar j = i + 1 while j < n and arr [j] >= arr [i]: curr += arr [i] j += 1 res = max (res , curr) return res if __name__ == "__main__" : arr = [60 , 20 , 50 , 40 , 10 , 50 , 60] print (getMaxArea (arr)) C# using System ; class GfG { static int getMaxArea (int [] arr) { int n = arr . Length ; int res = 0 ; // Consider every bar one by one for (int i = 0 ; i < n ; i ++) { int curr = arr [i]; // Traverse left while we have a greater height bar int j = i - 1 ; while (j >= 0 && arr [j] >= arr [i]) { curr += arr [i]; j -- ; } // Traverse right while we have a greater height bar j = i + 1 ; while (j < n && arr [j] >=

arr [i]) { curr += arr [i]; j ++ ; } res = Math . Max (res , curr); } return res ; } static void Main (string [] args) { int [] arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; Console . WriteLine (getMaxArea (arr)); } } JavaScript function getMaxArea (arr) { let n = arr . length ; let res = 0 ; // Consider every bar one by one for (let i = 0 ; i < n ; i ++) { let curr = arr [i]; // Traverse left while we have a greater height bar let j = i - 1 ; while (j >= 0 && arr [j] >= arr [i]) { curr += arr [i]; j -- ; } // Traverse right while we have a greater height bar j = i + 1 ; while (j < n && arr [j] >= arr [i]) { curr += arr [i]; j ++ ; } res = Math . max (res , curr); } return res ; } // Driver code let arr = [60 , 20 , 50 , 40 , 10 , 50 , 60]; console . log (getMaxArea (arr)); Output 100 [Expected Approach] Using Stack - O(n) Time and O(n) Space The idea is that Instead of recalculating the left and right boundary for every bar, we can precompute them once. The Previous Smaller Element (PSE) tells us the nearest bar on the left that is smaller than the current bar. The Next Smaller Element (NSE) tells us the nearest bar on the right that is smaller than the current bar. With these two values, we instantly know the maximum width a bar can extend while maintaining itself as the limiting height. Finally, multiplying this width with the current bar’s height gives the largest rectangle for that bar, and taking the maximum across all bars gives the answer. C++ #include <iostream> #include <stack> #include <vector> using namespace std ; // Function to find next smaller for every element vector < int > nextSmaller (vector < int >& arr) { int n = arr . size (); vector < int > nextS (n , n); stack < int > st ; for (int i = 0 ; i < n ; ++ i) { while (! st . empty () && arr [i] < arr [st . top ()]) { // Setting the index of the next smaller element // for the top of the stack nextS [st . top ()] = i ; st . pop (); } st . push (i); } return nextS ; } // Function to find previous smaller for every element vector < int > prevSmaller (vector < int >& arr) { int n = arr . size (); vector < int > prevS (n , -1); stack < int > st ; for (int i = 0 ; i < n ; ++ i) { while (! st . empty () && arr [i] < arr [st . top ()]) { // Setting the index of the previous smaller element // for the top of the stack st . pop (); } if (! st . empty ()) { prevS [i] = st . top (); } st . push (i); } return prevS ; } // Function to calculate the maximum rectangular // area in the Histogram int getMaxArea (vector < int >& arr) { vector < int > prevS = prevSmaller (arr); vector < int > nextS = nextSmaller (arr); int maxArea = 0 ; // Calculate the area for each Histogram bar for (int i = 0 ; i < arr . size (); ++ i) { int width = nextS [i] - prevS [i] - 1 ; int area = arr [i] * width ; maxArea = max (maxArea , area); } return maxArea ; } int main () { vector < int > arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; cout << getMaxArea (arr) << endl ; return 0 ; } C #include <stdio.h> #include <stdlib.h> // Stack structure struct Stack { int top ; int capacity ; int * items ; }; // Function to create an empty stack with dynamic memory allocation struct Stack * createStack (int capacity) { struct Stack * stack = (struct Stack *) malloc (sizeof (struct Stack)); stack -> capacity = capacity ; stack -> top = -1 ; stack -> items = (int *) malloc (stack -> capacity * sizeof (int)); return stack ; } // Function to check if the stack is empty int isEmpty (struct Stack * stack) { return stack -> top == -1 ; } // Function to push an element onto the stack void push (struct Stack * stack , int value) { if (stack -> top == stack -> capacity - 1) { printf ("Stack overflow \n "); return ; } stack -> items [++ (stack -> top)] = value ; } // Function to pop an element from the stack int pop (struct Stack * stack) { if (isEmpty (stack)) { printf ("Stack underflow \n "); return -1 ; } return stack -> items [(stack -> top) --]; } // Function to get the top element of the stack int peek (struct Stack * stack) { if (! isEmpty (stack)) { return stack -> items [stack -> top]; } return -1 ; } // Function to find the next smaller element for every element void nextSmaller (int arr [], int n , int nextS []) { struct Stack * stack = createStack (n); for (int i = 0 ; i < n ; i ++) { nextS [i] = n ; } for (int i = 0 ; i < n ; i ++) { while (! isEmpty (stack) && arr [i] < arr [peek (stack)]) { nextS [peek (stack)] = i ; pop (stack); } push (stack , i); } } // Function to find the previous smaller element for every element void prevSmaller (int arr [], int n , int prevS []) { struct Stack * stack = createStack (n); for (int i = 0 ; i < n ; i ++) { prevS [i] = -1 ; } for (int i = 0 ; i < n ; i ++) { while (! isEmpty (stack) && arr [i] < arr [peek (stack)]) { pop (stack); } if (! isEmpty (stack)) { prevS [i] = peek (stack); } push (stack , i); } } // Function to calculate the maximum rectangular // area in the Histogram int getMaxArea (int arr [], int n) { int * prevS = (int *) malloc (n * sizeof (int)); int * nextS = (int *) malloc (n * sizeof (int)); int maxArea = 0 ; // Find previous and next smaller elements prevSmaller (arr , n , prevS); nextSmaller (arr , n , nextS); // Calculate the area for each arrogram bar for (int i = 0 ; i < n ; i ++) { int width = nextS [i] - prevS [i] - 1 ; int area = arr [i] * width ; if (area > maxArea) { maxArea = area ; } } return maxArea ; } // Driver code int main () { int arr [] = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , getMaxArea (arr , n)); return 0 ; } Java import java.util.Stack ; class GfG { // Function to find next smaller for every element static int [] nextSmaller (int [] arr) { int n = arr . length ; int [] nextS = new int [n] ; for (int i = 0 ; i < n ; i ++) { nextS [i] = n ; } Stack < Integer > st = new Stack <> (); for (int i = 0 ; i < n ; i ++) { while (! st . isEmpty () && arr [i] < arr [st . peek ()]) { // Setting the index of the next smaller element // for the top of the stack nextS [st . pop ()] = i ; } st . push (i); } return nextS ; } // Function to find previous smaller for every element static int [] prevSmaller (int []

arr) { int n = arr . length ; int [] prevS = new int [n] ; for (int i = 0 ; i < n ; i ++) { prevS [i] = - 1 ; } Stack < Integer > st = new Stack <> (); for (int i = 0 ; i < n ; i ++) { while (! st . isEmpty () && arr [i] < arr [st . peek ()]) { st . pop (); } if (! st . isEmpty ()) { prevS [i] = st . peek (); } st . push (i); } return prevS ; } // Function to calculate the maximum rectangular // area in the histogram static int getMaxArea (int [] arr) { int [] prevS = prevSmaller (arr); int [] nextS = nextSmaller (arr); int maxArea = 0 ; // Calculate the area for each arrogram bar for (int i = 0 ; i < arr . length ; i ++) { int width = nextS [i] - prevS [i] - 1 ; int area = arr [i] * width ; maxArea = Math . max (maxArea , area); } return maxArea ; } public static void main (String [] args) { int [] arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; System . out . println (getMaxArea (arr)); } } Python # Function to find next smaller for every element def nextSmaller (arr): n = len (arr) nextS = [n] * n st = [] for i in range (n): while st and arr [i] < arr [st [- 1]]: # Setting the index of the next smaller element # for the top of the stack nextS [st . pop ()] = i st . append (i) return nextS # Function to find previous smaller for every element def prevSmaller (arr): n = len (arr) prevS = [- 1] * n st = [] for i in range (n): while st and arr [i] < arr [st [- 1]]: st . pop () if st : prevS [i] = st [- 1] st . append (i) return prevS # Function to calculate the maximum rectangular # area in the Histogram def getMaxArea (arr): prevS = prevSmaller (arr) nextS = nextSmaller (arr) maxArea = 0 # Calculate the area for each arrogram bar for i in range (len (arr)): width = nextS [i] - prevS [i] - 1 area = arr [i] * width maxArea = max (maxArea , area) return maxArea if __name__ == "__main__" : arr = [60 , 20 , 50 , 40 , 10 , 50 , 60] print (getMaxArea (arr)) C# using System ; using System.Collections.Generic ; class GfG { // Function to find next smaller for every element static int [] nextSmaller (int [] arr) { int n = arr . Length ; int [] nextS = new int [n]; for (int i = 0 ; i < n ; i ++) { nextS [i] = n ; } Stack < int > st = new Stack < int > (); for (int i = 0 ; i < n ; i ++) { while (st . Count > 0 && arr [i] < arr [st . Peek ()]) { // Setting the index of the next smaller element // for the top of the stack nextS [st . Pop ()] = i ; } st . Push (i); } return nextS ; } // Function to find previous smaller for every element static int [] prevSmaller (int [] arr) { int n = arr . Length ; int [] prevS = new int [n]; for (int i = 0 ; i < n ; i ++) { prevS [i] = - 1 ; } Stack < int > st = new Stack < int > (); for (int i = 0 ; i < n ; i ++) { while (st . Count > 0 && arr [i] < arr [st . Peek ()]) { st . Pop (); } if (st . Count > 0) { prevS [i] = st . Peek (); } st . Push (i); } return prevS ; } // Function to calculate the maximum rectangular // area in the Histogram static int getMaxArea (int [] arr) { int [] prevS = prevSmaller (arr); int [] nextS = nextSmaller (arr); int maxArea = 0 ; // Calculate the area for each arrogram bar for (int i = 0 ; i < arr . Length ; i ++) { int width = nextS [i] - prevS [i] - 1 ; int area = arr [i] * width ; maxArea = Math . Max (maxArea , area); } return maxArea ; } static void Main () { int [] arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; Console . WriteLine (getMaxArea (arr)); } } JavaScript // Function to find next smaller for every element function nextSmaller (arr){ const n = arr . length ; const nextS = new Array (n). fill (n); const stack = []; for (let i = 0 ; i < n ; i ++) { while (stack . length && arr [i] < arr [stack [stack . length - 1]]) { // Setting the index of the next smaller element // for the top of the stack nextS [stack . pop ()] = i ; } stack . push (i); } return nextS ; } // Function to find previous smaller for every element function prevSmaller (arr) { const n = arr . length ; const prevS = new Array (n). fill (- 1); const stack = []; for (let i = 0 ; i < n ; i ++) { while (stack . length && arr [i] < arr [stack [stack . length - 1]]) { stack . pop (); } if (stack . length) { prevS [i] = stack [stack . length - 1]; } stack . push (i); } return prevS ; } // Function to calculate the maximum rectangular // area in the Histogram function getMaxArea (arr) { const prevS = prevSmaller (arr); const nextS = nextSmaller (arr); let maxArea = 0 ; // Calculate the area for each arrogram bar for (let i = 0 ; i < arr . length ; i ++) { const width = nextS [i] - prevS [i] - 1 ; const area = arr [i] * width ; maxArea = Math . max (maxArea , area); } return maxArea ; } // Driver code const arr = [60 , 20 , 50 , 40 , 10 , 50 , 60]; console . log (getMaxArea (arr)); Output 100 [Optimized Approach] Using Single Stack - O(n) Time and O(n) Space This approach is mainly an optimization over the previous approach. The idea is to maintain a stack of bar indices in increasing height order. When a shorter bar is encountered, it means the bar at the top of the stack cannot extend further to the right. We pop it, and using the current index as the right boundary and the new top of the stack as the left boundary, compute the area with the popped bar's height as the smallest height. Below are the detailed steps of implementation. Start with an empty stack and process bars from left to right. Keep pushing bars into the stack as long as their heights are in non-decreasing order . When you encounter a bar shorter than the bar at the top of the stack: The bar at stack.top() cannot extend further to the right. Pop it from the stack this popped bar's height becomes the smallest height for its rectangle. The current index is the right boundary. The index now at the top of the stack (after popping) gives the previous smaller element, marking the left boundary. Compute width = right_boundary - left_boundary - 1 and update max area.Continue until the stack is empty or the current bar is taller, then push the current index. After processing all bars, pop any remaining bars in the stack and Compute width, area, and update the maximum. C++ #include <iostream> #include <stack>

```
#include <vector> using namespace std ; int getMaxArea ( vector < int >& arr ) { int n = arr . size ();
stack < int > st ; int res = 0 ; int tp , curr ; for ( int i = 0 ; i < n ; i ++ ) { while ( ! st . empty () && arr [ st . top
()] >= arr [ i ]) { // The popped item is to be considered as the // smallest element of the Histogram tp =
st . top (); st . pop (); // For the popped item previous smaller element is // just below it in the stack (or
current stack top) // and next smaller element is i int width = st . empty () ? i : i - st . top () - 1 ; res = max
( res , arr [ tp ] * width ); } st . push ( i ); } // For the remaining items in the stack, next smaller does // not
exist. Previous smaller is the item just below in // stack. while ( ! st . empty ()) { tp = st . top (); st . pop ();
curr = arr [ tp ] * ( st . empty () ? n : n - st . top () - 1 ); res = max ( res , curr ); } return res ; } int main () {
vector < int > arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; cout << getMaxArea ( arr ); return 0 ; } C #include
<stdio.h> #include <stdlib.h> // Stack structure struct Stack { int top ; int capacity ; int * array ; }; //
Function to create a stack struct Stack * createStack ( int capacity ) { struct Stack * stack = ( struct
Stack * ) malloc ( sizeof ( struct Stack )); stack -> capacity = capacity ; stack -> top = -1 ; stack -> array
= ( int * ) malloc ( stack -> capacity * sizeof ( int )); return stack ; } int isEmpty ( struct Stack * stack ) {
return stack -> top == -1 ; } void push ( struct Stack * stack , int item ) { stack -> array [ ++ stack -> top ]
= item ; } int pop ( struct Stack * stack ) { return stack -> array [ stack -> top -- ]; } int peek ( struct Stack
* stack ) { return stack -> array [ stack -> top ]; } // Function to calculate the maximum rectangular area
int getMaxArea ( int arr [], int n ) { struct Stack * st = createStack ( n ); int res = 0 , tp , curr ; // Traverse
all bars of the arrogram for ( int i = 0 ; i < n ; i ++ ) { // Process the stack while the current element // is
smaller than the element corresponding to // the top of the stack while ( ! isEmpty ( st ) && arr [ peek ( st
)] >= arr [ i ]) { tp = pop ( st ); // Calculate width and update result int width = isEmpty ( st ) ? i : i - peek (
st ) - 1 ; res = ( res > arr [ tp ] * width ) ? res : arr [ tp ] * width ; } push ( st , i ); } // Process remaining
elements in the stack while ( ! isEmpty ( st )) { tp = pop ( st ); curr = arr [ tp ] * ( isEmpty ( st ) ? n : n -
peek ( st ) - 1 ); res = ( res > curr ) ? res : curr ; } return res ; } int main () { int arr [] = { 60 , 20 , 50 , 40 ,
10 , 50 , 60 }; int n = sizeof ( arr ) / sizeof ( arr [ 0 ]); printf ( "%d \n " , getMaxArea ( arr , n )); return 0 ; }
Java import java.util.Stack ; class GfG { static int getMaxArea ( int [] arr ) { int n = arr . length ; Stack <
Integer > st = new Stack <> (); int res = 0 , tp , curr ; for ( int i = 0 ; i < n ; i ++ ) { // Process the stack
while the current element // is smaller than the element corresponding to // the top of the stack while ( !
st . isEmpty () && arr [ st . peek () ] >= arr [ i ] ) { // The popped item is to be considered as the //
smallest element of the Histogram tp = st . pop (); // For the popped item, previous smaller element // is
just below it in the stack (or current stack // top) and next smaller element is i int width = st . isEmpty () ?
i : i - st . peek () - 1 ; // Update the result if needed res = Math . max ( res , arr [ tp ] * width ); } st . push (
i ); } // For the remaining items in the stack, next smaller does // not exist. Previous smaller is the item
just below in // the stack. while ( ! st . isEmpty ()) { tp = st . pop (); curr = arr [ tp ] * ( st . isEmpty () ? n : n
- st . peek () - 1 ); res = Math . max ( res , curr ); } return res ; } public static void main ( String [] args ) {
int [] arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; System . out . println ( getMaxArea ( arr )); } } Python def
getMaxArea ( arr ): n = len ( arr ) st = [] res = 0 for i in range ( n ): # Process the stack while the current
element # is smaller than the element corresponding to # the top of the stack while st and arr [ st [ - 1 ]]
>= arr [ i ]: # The popped item is to be considered as the # smallest element of the Histogram tp = st .
pop () # For the popped item, the previous smaller # element is just below it in the stack (or # the
current stack top) and the next smaller # element is i width = i if not st else i - st [ - 1 ] - 1 # Update the
result if needed res = max ( res , arr [ tp ] * width ) st . append ( i ) # For the remaining items in the
stack, next smaller does # not exist. Previous smaller is the item just below in # the stack. while st : tp =
st . pop () width = n if not st else n - st [ - 1 ] - 1 res = max ( res , arr [ tp ] * width ) return res if
__name__ == "__main__" : arr = [ 60 , 20 , 50 , 40 , 10 , 50 , 60 ] print ( getMaxArea ( arr )) C# // C#
program to find the largest rectangular area possible // in a given histogram using System ; using
System.Collections.Generic ; class GfG { // Function to calculate the maximum rectangular area static
int getMaxArea ( int [] arr ) { int n = arr . Length ; Stack < int > st = new Stack < int > (); int res = 0 , tp ,
curr ; // Traverse all bars of the arrogram for ( int i = 0 ; i < n ; i ++ ) { // Process the stack while the
current element // is smaller than the element corresponding to // the top of the stack while ( st . Count >
0 && arr [ st . Peek ()] >= arr [ i ]) { tp = st . Pop (); // Calculate width and update result int width = st .
Count == 0 ? i : i - st . Peek () - 1 ; res = Math . Max ( res , arr [ tp ] * width ); } st . Push ( i ); } // Process
remaining elements in the stack while ( st . Count > 0 ) { tp = st . Pop (); curr = arr [ tp ] * ( st . Count ==
0 ? n : n - st . Peek () - 1 ); res = Math . Max ( res , curr ); } return res ; } public static void Main () { int []
arr = { 60 , 20 , 50 , 40 , 10 , 50 , 60 }; Console . WriteLine ( getMaxArea ( arr )); } } JavaScript function
getMaxArea ( arr ) { let n = arr . length ; let st = []; let res = 0 ; // Traverse all bars of the arrogram for (
let i = 0 ; i < n ; i ++ ) { // Process the stack while the current element // is smaller than the element
corresponding to // the top of the stack while ( st . length && arr [ st [ st . length - 1 ]] >= arr [ i ]) { let tp =
```

st . pop (); // Calculate width and update result let width = st . length === 0 ? i : i - st [st . length - 1] - 1 ; res = Math . max (res , arr [tp] * width); } st . push (i); } // Process remaining elements in the stack while (st . length) { let tp = st . pop (); let curr = arr [tp] * (st . length === 0 ? n : n - st [st . length - 1] - 1); res = Math . max (res , curr); } return res ; } // Driver code let arr = [60 , 20 , 50 , 40 , 10 , 50 , 60]; console . log (getMaxArea (arr)); Output 100 [Alternate Approach] Using Divide and Conquer - O(n log(n)) Time and O(n) Space The idea is to find the minimum value in the given array. Once we have index of the minimum value, the max area is maximum of following three values. Maximum area in left side of minimum value (Not including the min value) Maximum area in right side of minimum value (Not including the min value) Number of bars multiplied by minimum value. Please refer Largest Rectangular Area in a histogram Using Divide and Conquer for detailed implementation. Comment Article Tags: Article Tags: Stack DSA Microsoft Amazon Facebook Snapdeal Paytm MAQ Software Akosha + 5 More