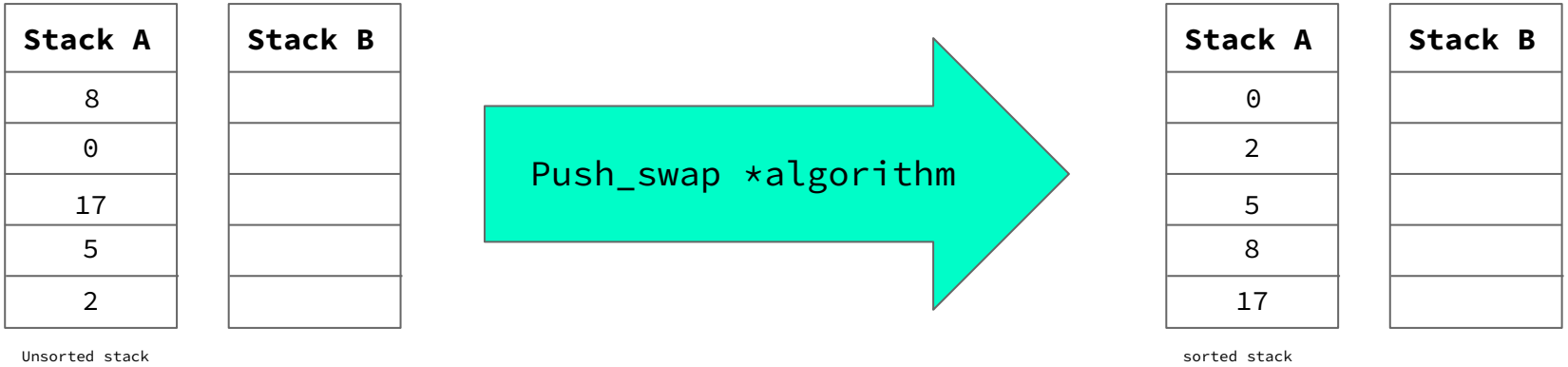
PUSH_SWAP

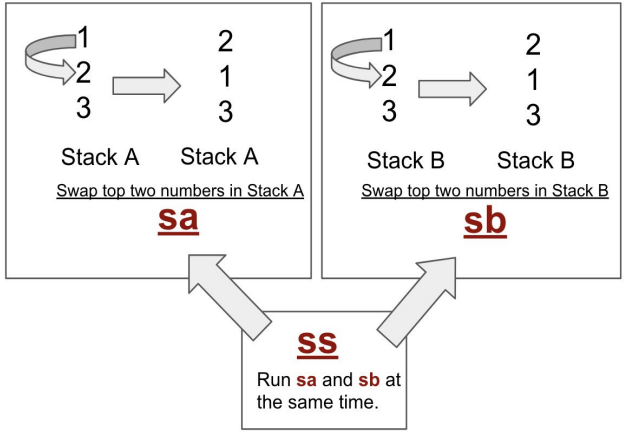


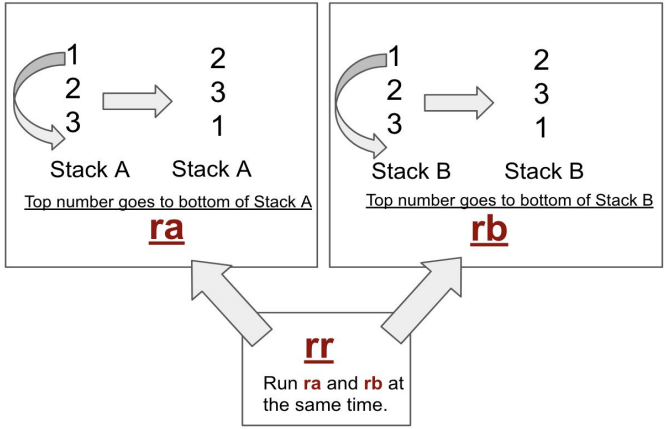
The idea of **Push_Swap** is simple, You have two stacks called **Stack A** and **Stack B**. Stack A is given a random list of unorganized numbers. You must take the random list of numbers in Stack A and sort them so that Stack A is organized from smallest to largest. There are only a few moves you're allowed to used to manipulate the stacks that we're going to call "Actions". The main goal of this project is to organize Stack A in as few **actions** as possible.

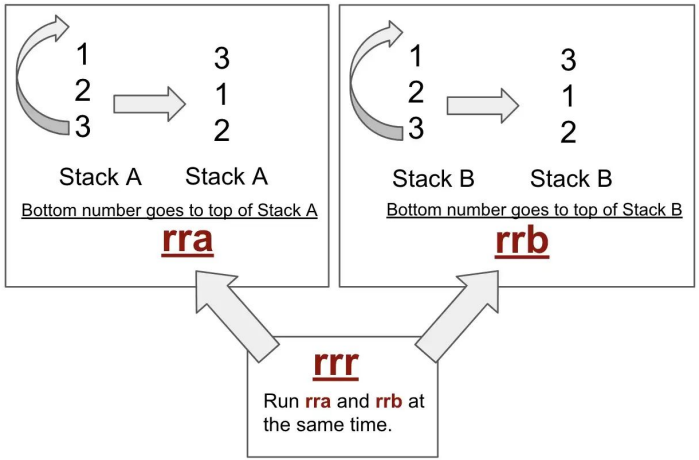
Actions

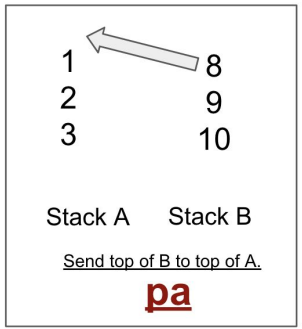
✔The goal is to sort in ascending order numbers into stack a. To do so you have the following operations at your disposal:

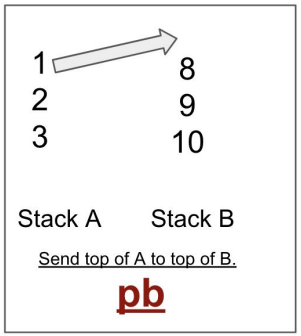
- **sa (swap a)**: Swap the first 2 elements at the top of **stack a**. Do nothing if there is only one or no elements.
- **sb (swap b)**: Swap the first 2 elements at the top of **stack b**. Do nothing if there is only one or no elements.
- **ss** : **sa** and **sb** at the same time.
- **pa (push a)**: Take the first element at the **top of b** and put it at the **top of a**. Do nothing if b is empty.
- **pb (push b)**: Take the first element at the **top of a** and put it at the **top of b**. Do nothing if a is empty.
- **ra (rotate a)**: Top number in **A** goes to bottom of **A**
- **rb (rotate b)**: Top number in **B** goes to bottom of **B**.
- **rr** : **ra** and **rb** at the same time.
- **rra (reverse rotate a)**: Bottom number in **A** goes to top of **A**.
- **rrb (reverse rotate b)**: Bottom number in **B** goes to top of **B**.
- **rrr** : **rra** and **rrb** at the same time.


1
2
3
2
1
3
Stack A
Stack A
Swap top two numbers in Stack A
sa
1
2
3
2
1
3
Stack B
Stack B
Swap top two numbers in Stack B
sb
ss
Run sa and sb at the same time.


1
2
3
2
3
1
Stack A
Stack A
Top number goes to bottom of Stack A
ra
1
2
3
2
3
1
Stack B
Stack B
Top number goes to bottom of Stack B
rb
rr
Run ra and rb at the same time.


1
2
3
3
1
2
Stack A
Stack A
Bottom number goes to top of Stack A
rra
1
2
3
3
1
2
Stack B
Stack B
Bottom number goes to top of Stack B
rrb
rrr
Run rra and rrb at the same time.


1
2
3
8
9
10
Stack A
Stack B
Send top of B to top of A.
pa


1
2
3
8
9
10
Stack A
Stack B
Send top of A to top of B.
pb

Sort five Elements

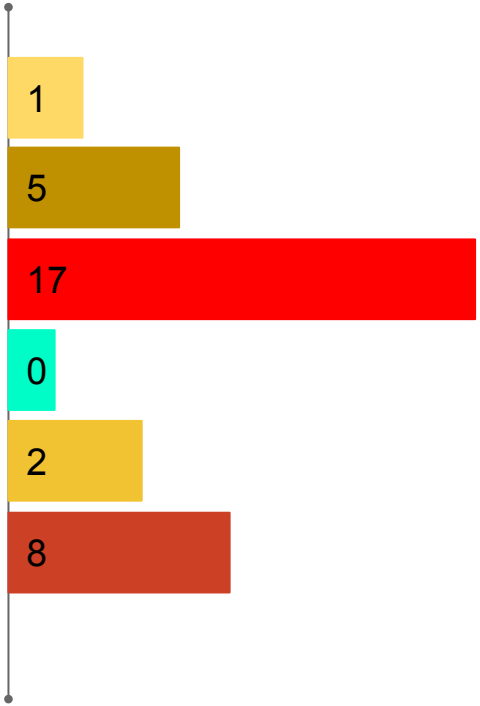
Stack A
1
5
17
0
2
8

Unsorted stack

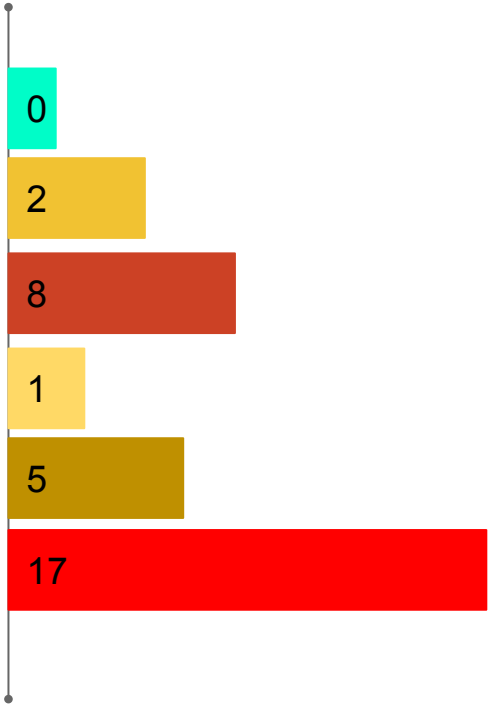
Stack B

The idea is we will search for the minimum value on the stack A and push it to the stack B until still just three elements on stack A, And use the The Hard Coded sorting for three element to sort it and push back all elements from B to Stack A

Sort five Elements

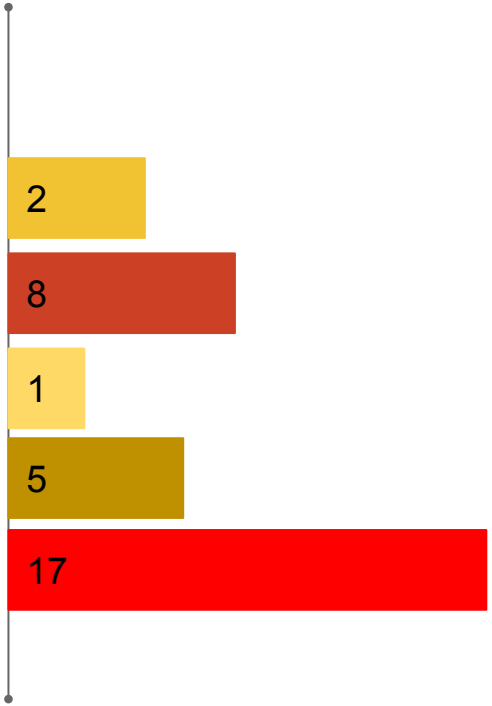


Sort five Elements



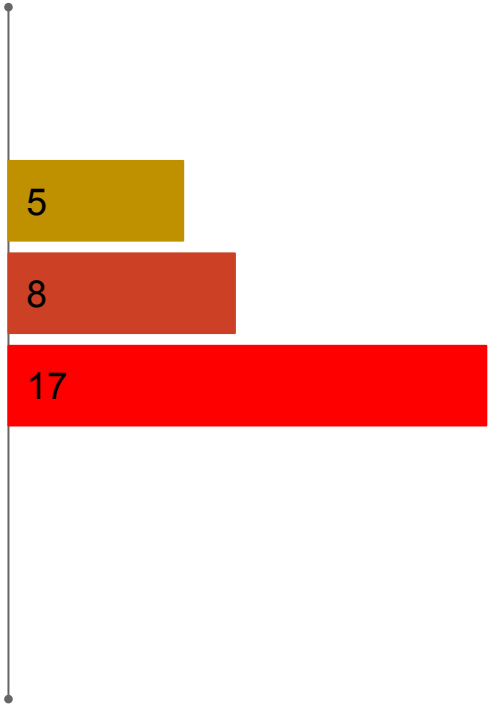
Actions	-rra-rra-rra || So min value at the top of the stack A now let execute pb

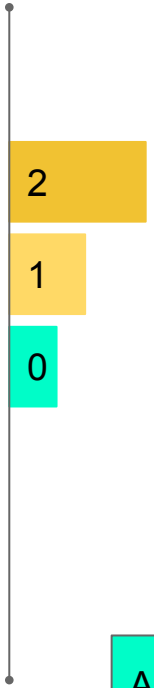
Sort five Elements



Actions	Pb || And now we will repeat the same scenario

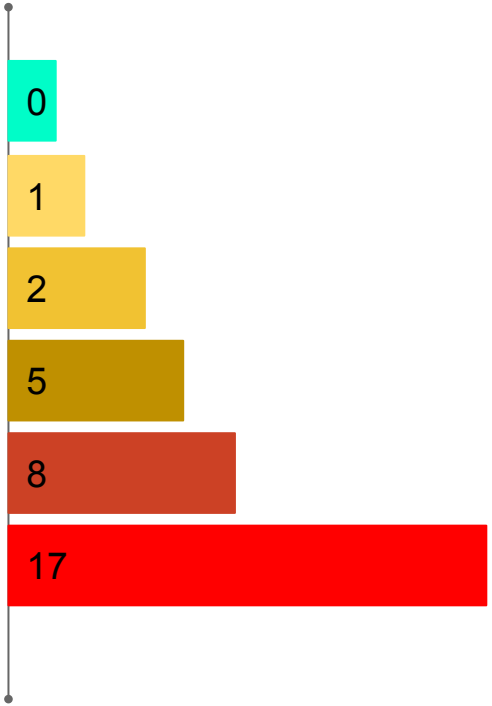
Sort five Elements





Actions	That How look two stack, After that we will push all element on Stack B to A

Sort five Elements



Actions	And as you see stack is sort, really are we finish no of course, if we use this method on 100 elements the number of instructions will be absolutely huge, so what is the solution…

Large sort Algorithm

The idea of this algorithm is first sort the numbers on an array using any sorting algorithm (I used quick sort), and define the div variable or the number of chunks, and define the controllers variables {mid, offset, start, end}
controller:

- Mid : for hold the index of the middle element on the sorted array, (size / 2 - 1)
- Offset : (2*offset) the number of element that will push to Stack B (offset == size / div)
  In the first push the that pushed will be (2*offset + 1) considered the sorted_arr[mid]
- Start : the start of the range
- End : the end of the range
- When we start push the elements between the range [array[start], array[end]] to stack B we should check if this element is less than the sorted_array[mid] if it is, so we must rotate the first element to the bottom of B (rb)

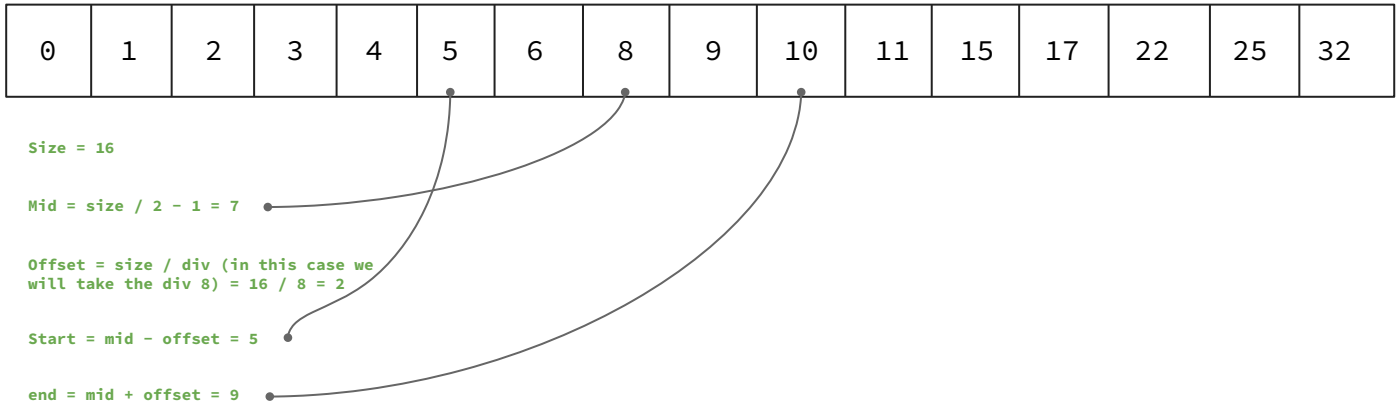
- When we push all the element between the range [array[start], array[end]], we have to update the start and end of range by subtracting the offset from start (start - offset), but before it we must check if it negative value so we have to set start at the index 0, and for end we should add offset to it and also check if we pass the size of array if it true So we will set the value of end at index size - 1 (which is the last element)

Large sort : stage 1

Stack A
1
5
17
0
2
8
3
15
9
6
22
4
32
10
25
11

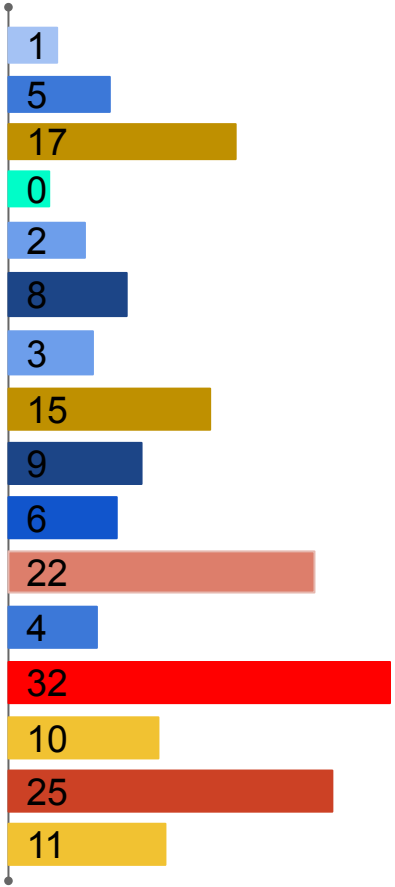
Unsorted stack

Sorted array

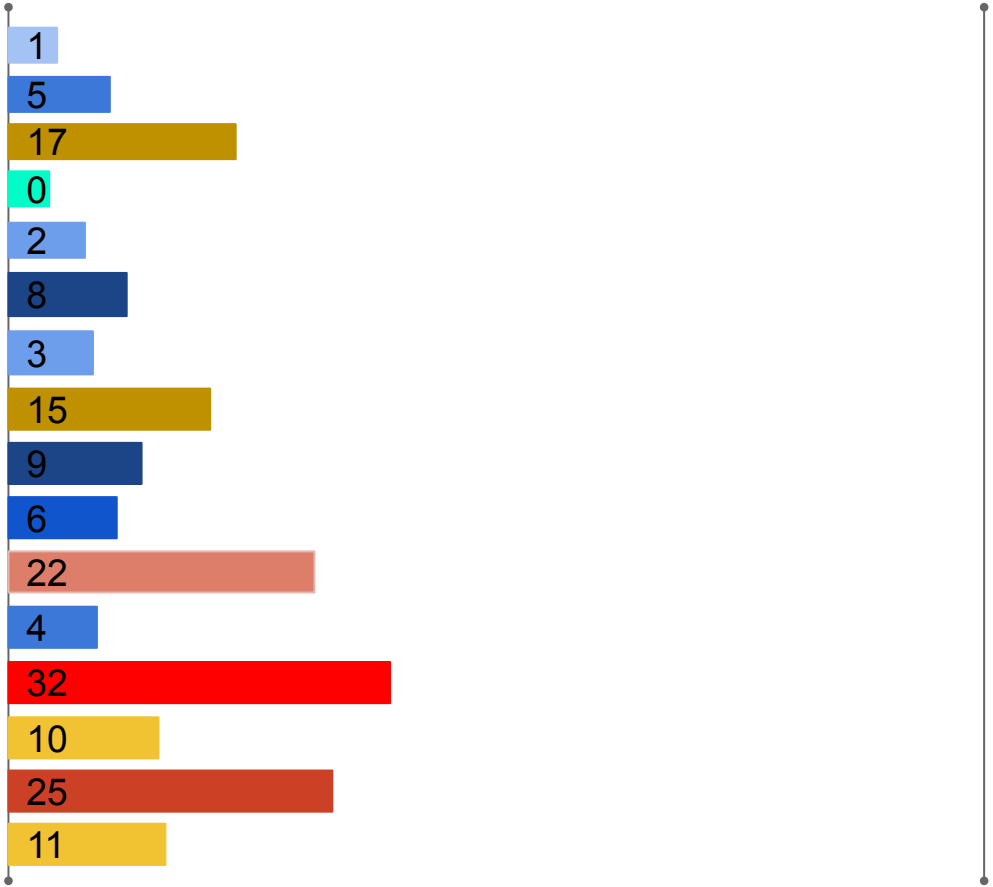


Now we will push all the elements in this range to stack B with check if it less or equal array[mid] (which is 8 here) if it true as i did say we must rotate it to the bottom of B

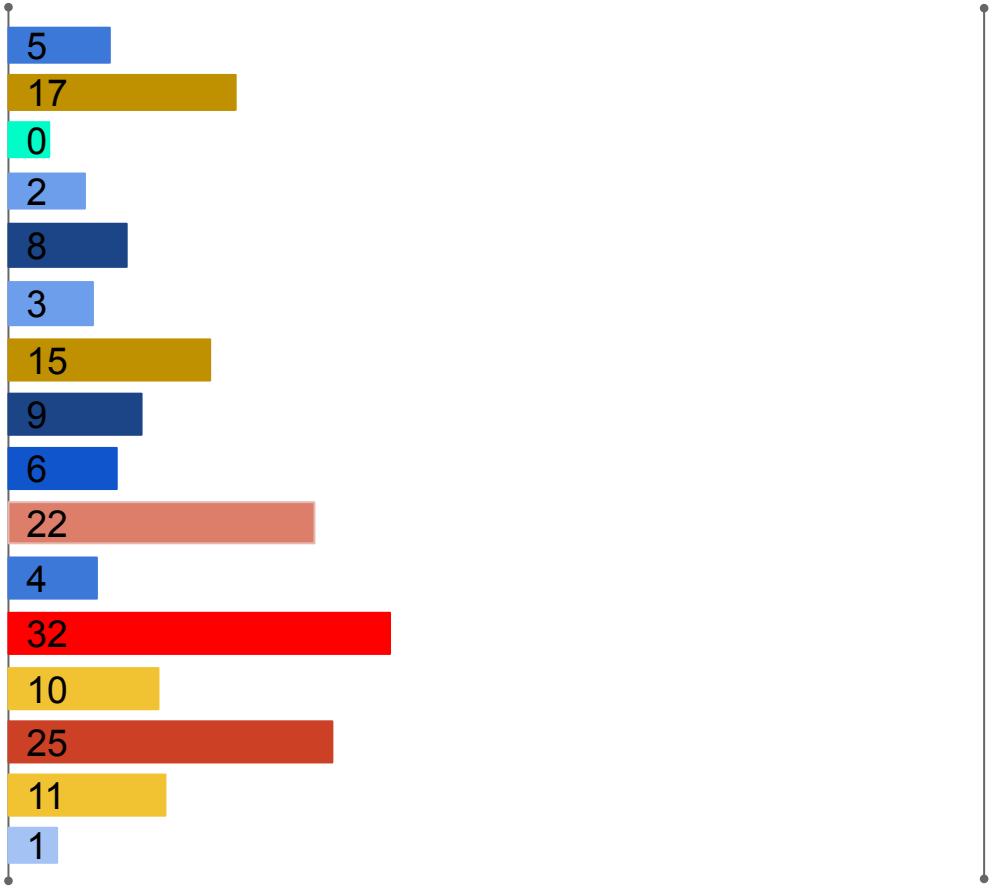
Large sort : stage 1



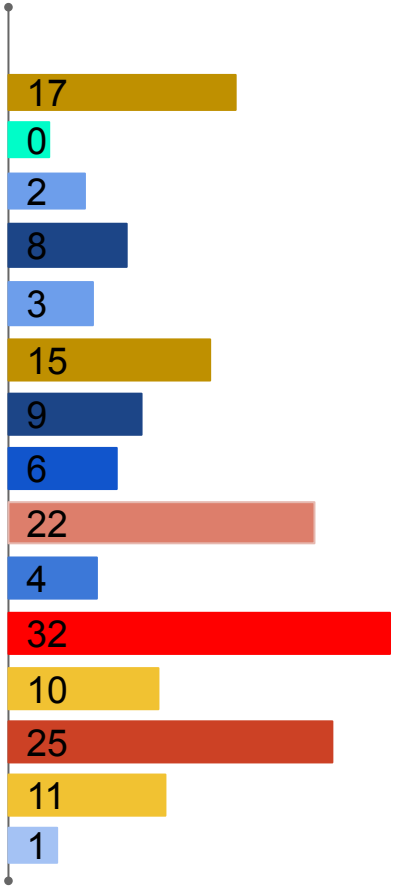
- As we say if the element on the top belong the range we will push it, if not we will define his index and decide which is more efficient (ra or rra) according to index is less than (size / 2) or not, in this case 1 is out of the range [5, 10] so we will search for the first element that on the range which is 5 with index 1 (1 < 18 / 2) so the best action is ra



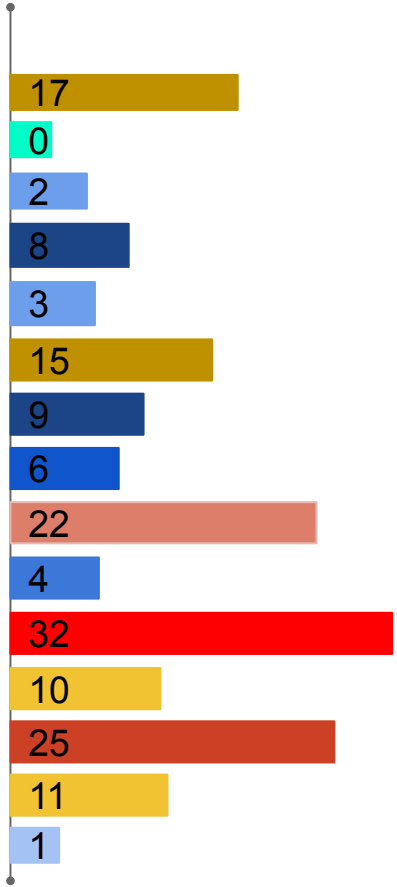
- 5 is on the range [array[5], array[10]] so let push it to stack B pb



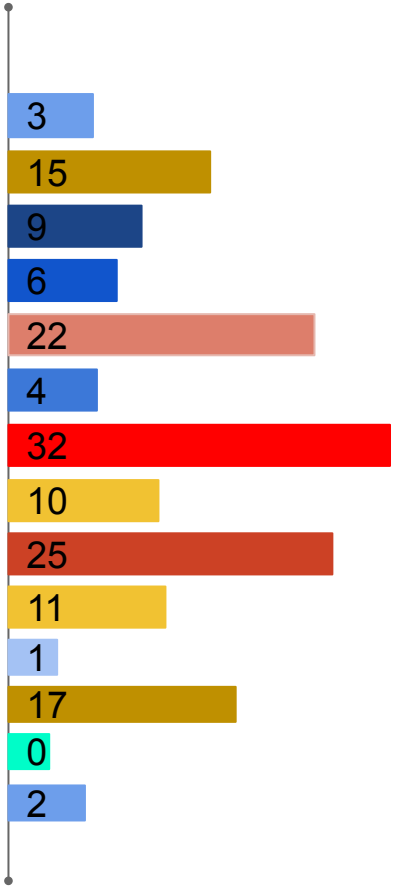
- Even 5 is less than the arr[mid] == 8 We won’t rotate because there is no more element to swap with



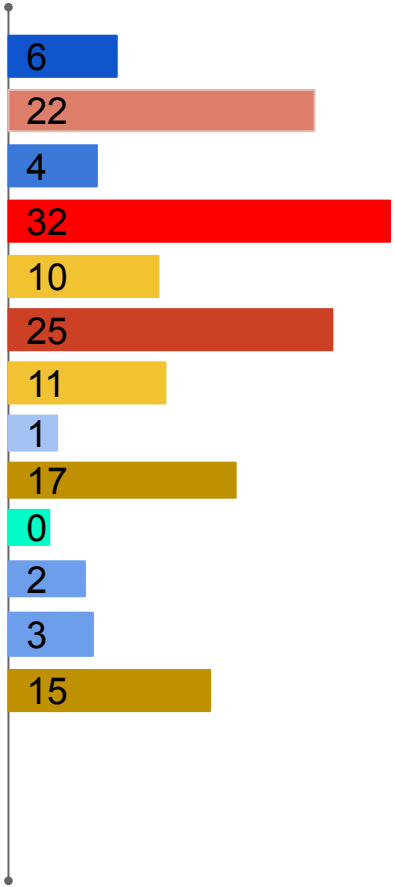
- 17 is out of the range so the first seen element belong the range is 8 with index[3] so let call ra for 3 times
  And push 8 to stack B



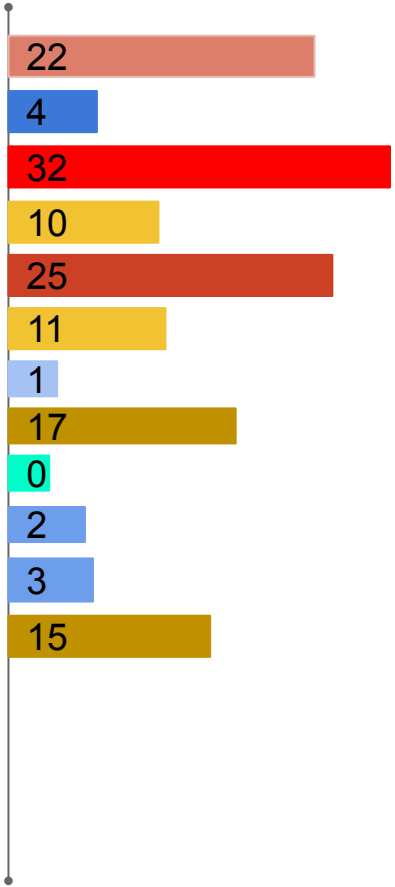
- 8 not less than arr[mid] which is equal, so we didn’t rotate
- 3 is out of the range so the first seen element belong the range is 9 with index[2] so let call ra for 2 times and push it to stack B



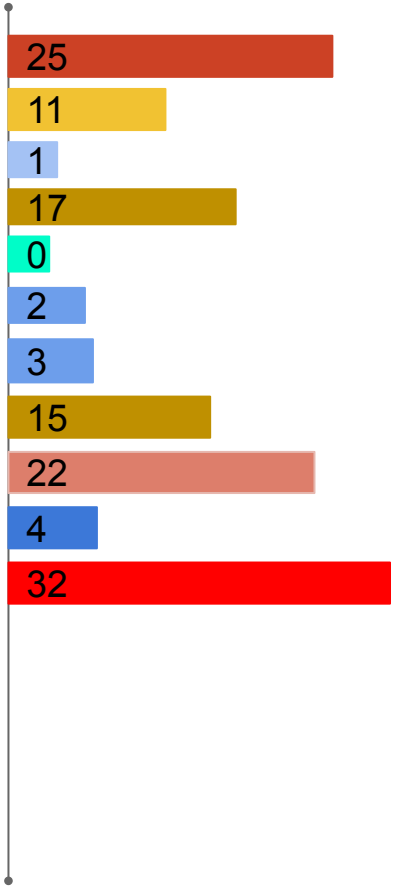
- 9 is greater than arr[mid], so we didn’t rotate it
- 6 is inside the range [5, 10] so will just push it stack B, but wait 6 is less than array[mid] == 8 so we have to rotate it also



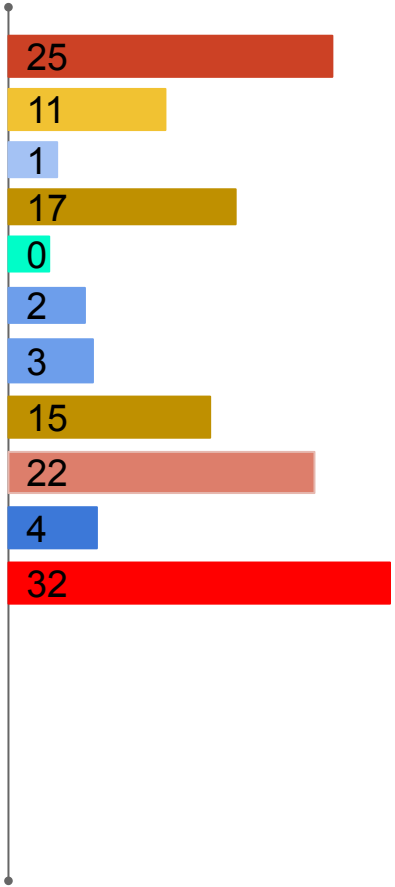
- As we see we push 6 and rotate it to the bottom
- 22 is out the range, the first seen 10 with index[3], let call ra 3 times and push to B



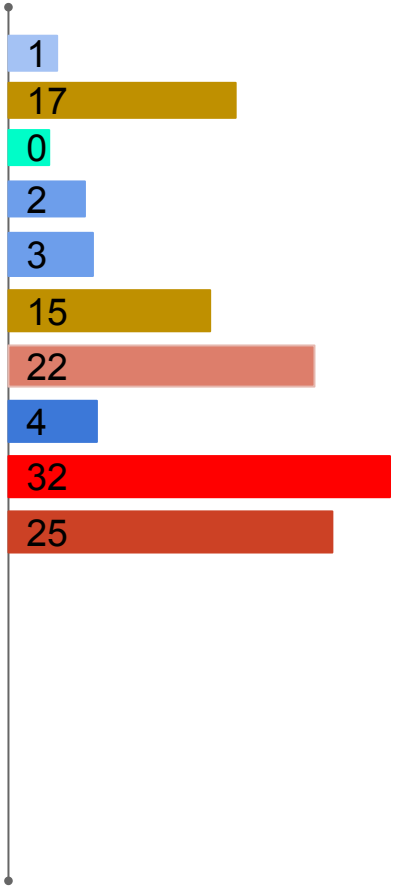
- 25 out of the range, but when try to search for element on the range [5, 10], we won’t find any one
- So know we must update the start and the end as i did say on the first
- start -= offset && end += offset ⇒ start = 3 && end = 11 ⇒ the new range [arr[3], arr[11]] ⇒ [3, 15]



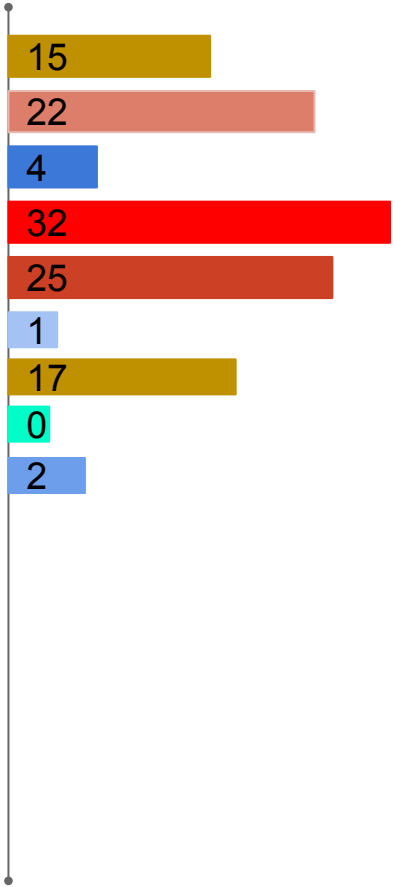
- 25 out of the range [3, 15], so the first seen element on the range is 11 with index[1], so call ra 1 time and push to B

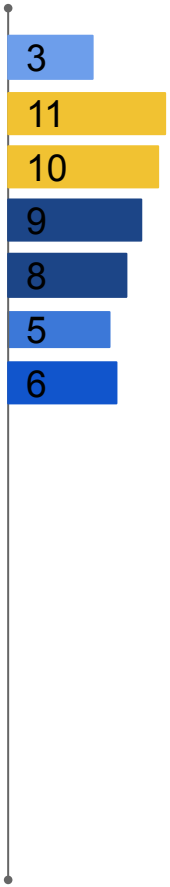


- 11 is greater than 8, so we don’t need to rotate it
- The first seen on the range is 3 with index[4] , so call ra 4 times and push to B (pb)

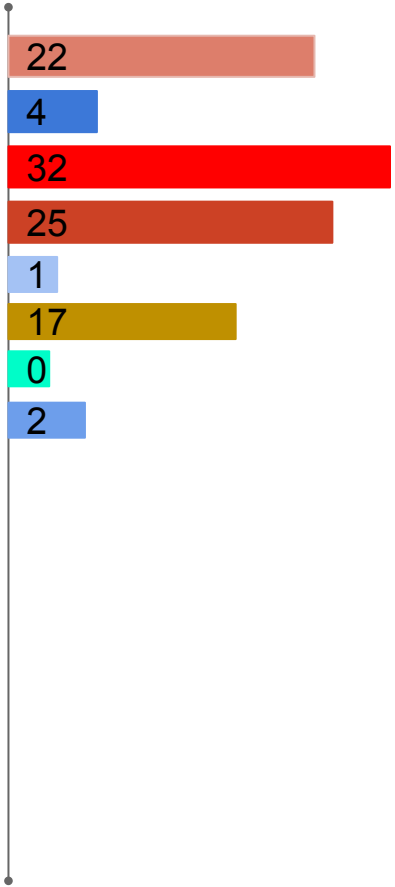


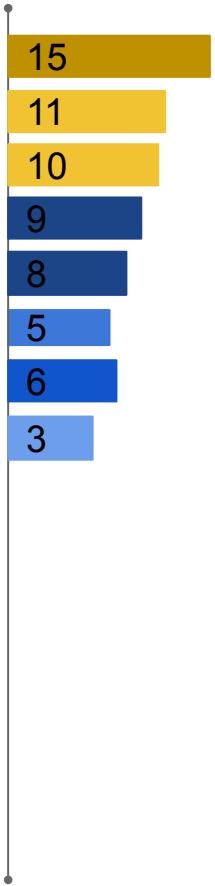
- As you see 3 is less than 8 so rotate it to the bottom



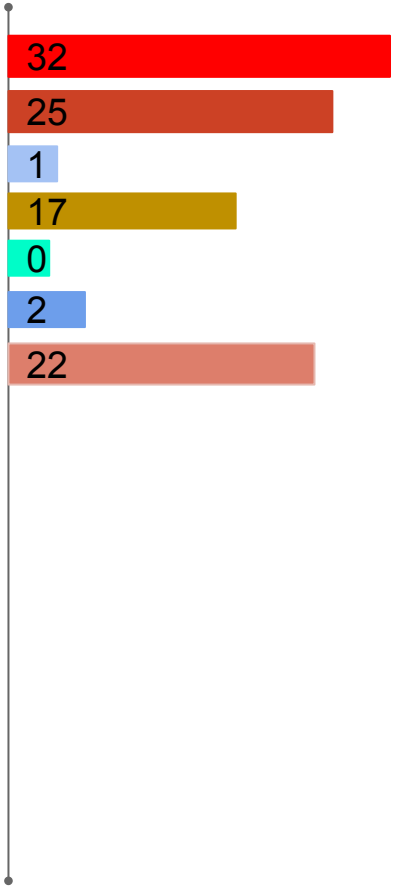


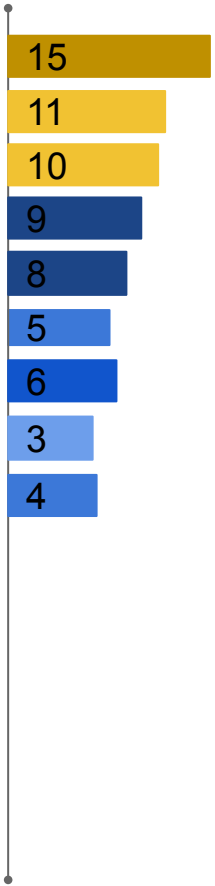
- The first seen element is 4 with index 1 ([3, 15]), so let call ra and pb, and of course 4 < 8 so we will call rb



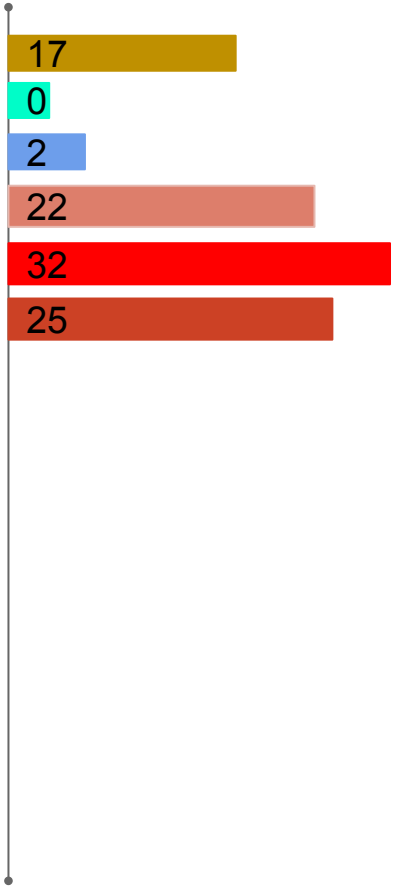


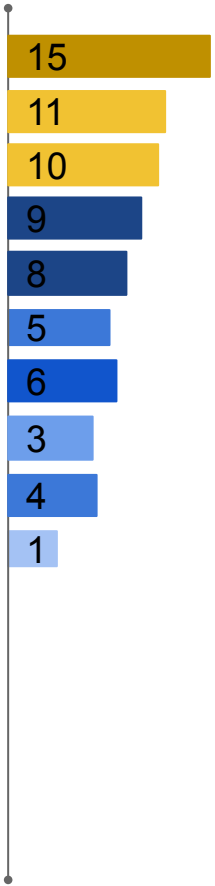
- We won’t find any element on [3, 15], so we will update the controller values (start, and end) start = 1 && end = 14
- New range is (array[1], array[13]) ⇒ [1, 22]
- 1 is the first seen element on the range with index 2, so we will call ra 2 times and push to B (pb), and rotate B (rb)



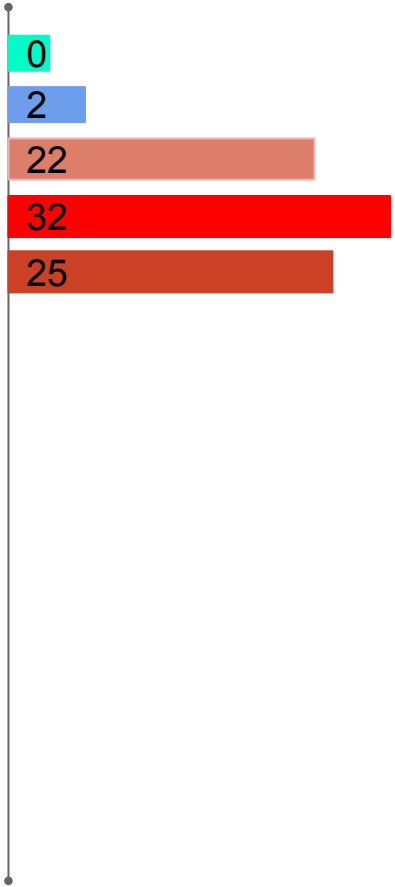


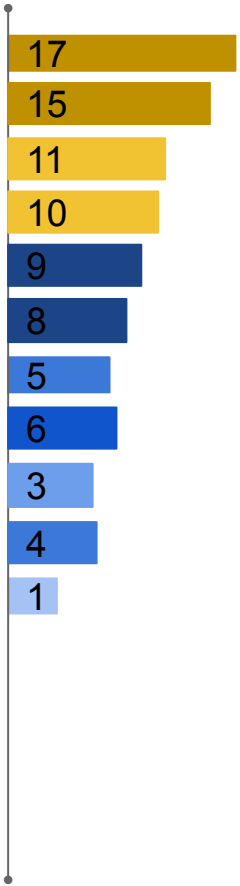
- 17 is on the range [1, 22] push it to B (pb)



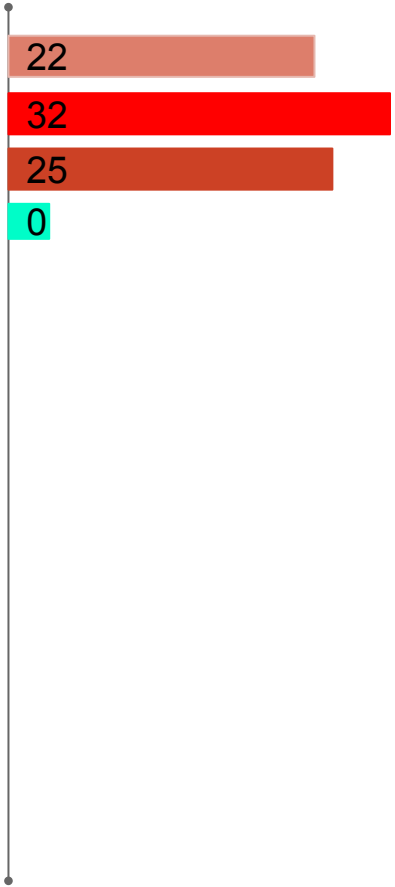


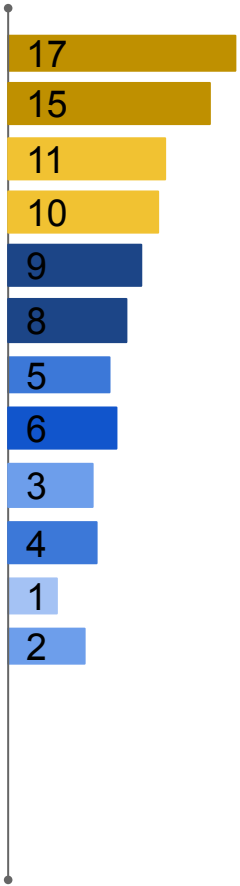
- 2 is on the range [1, 22] with index 1, so we will call ra 1 times and push to B (pb), and rb



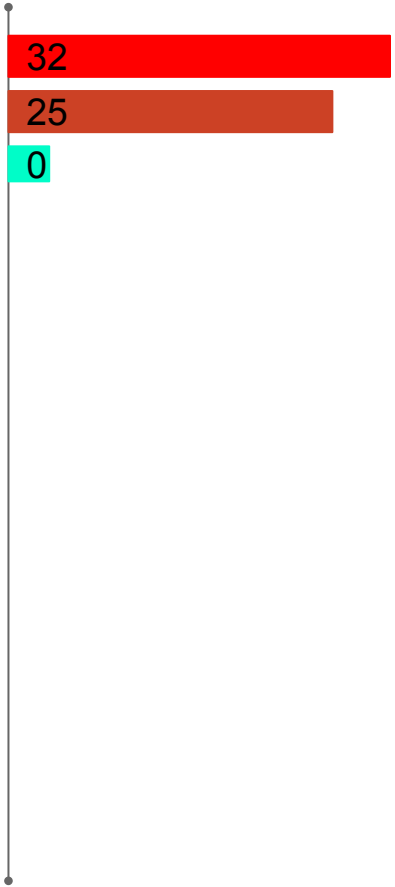


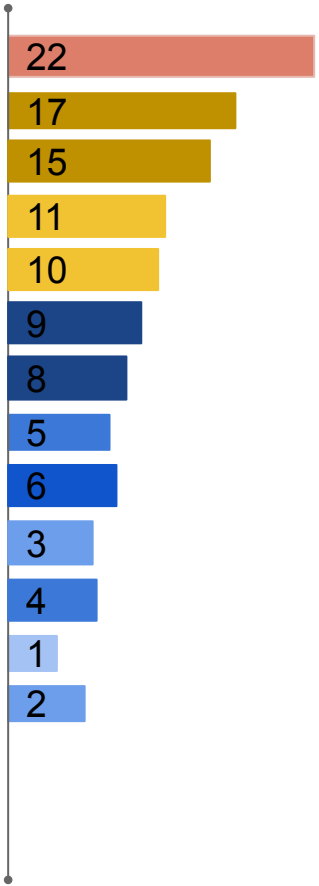
- 22 is on the range [1, 22] with index 0, so we will push to B (pb)



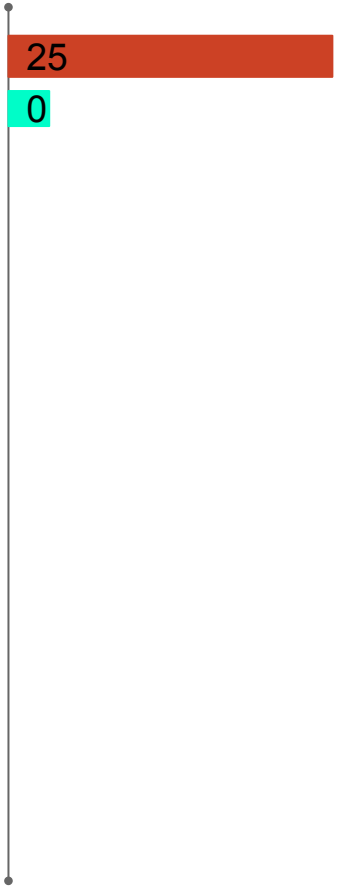


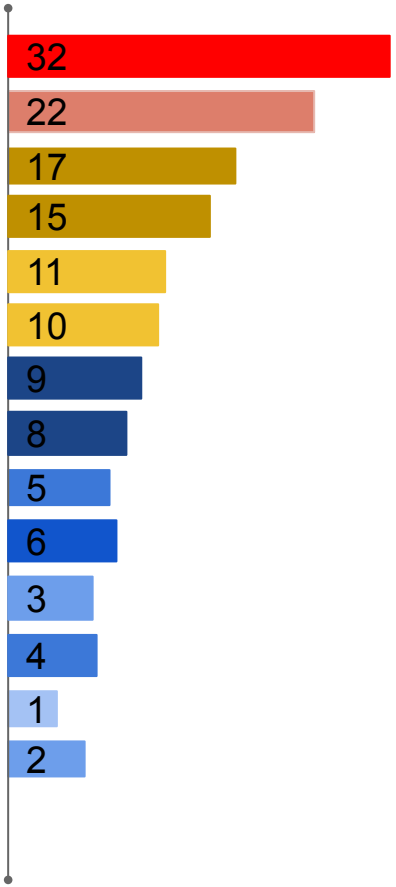
- We won’t find any element on the range, so let update it (start -= offset = -1 less than 0 so, start = 0, end = 15)
- New range (array[0], array[15]) ⇒ [0, 32]
- 32 is on the range so push to B (pb)



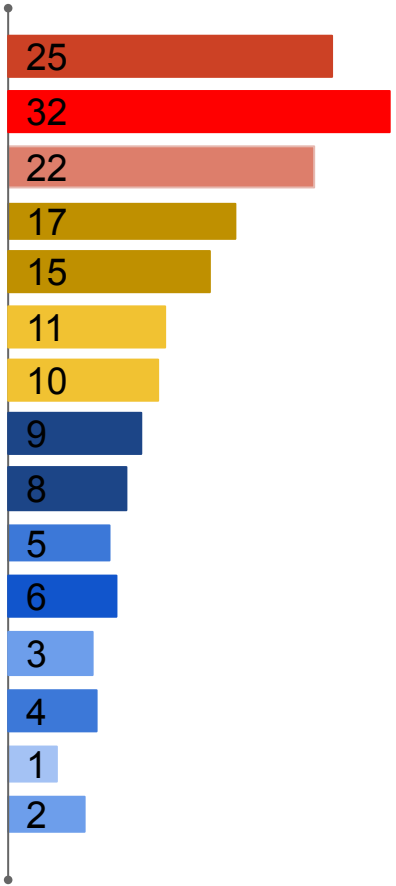


- 25 is on the range [0, 32], push to B (pb)

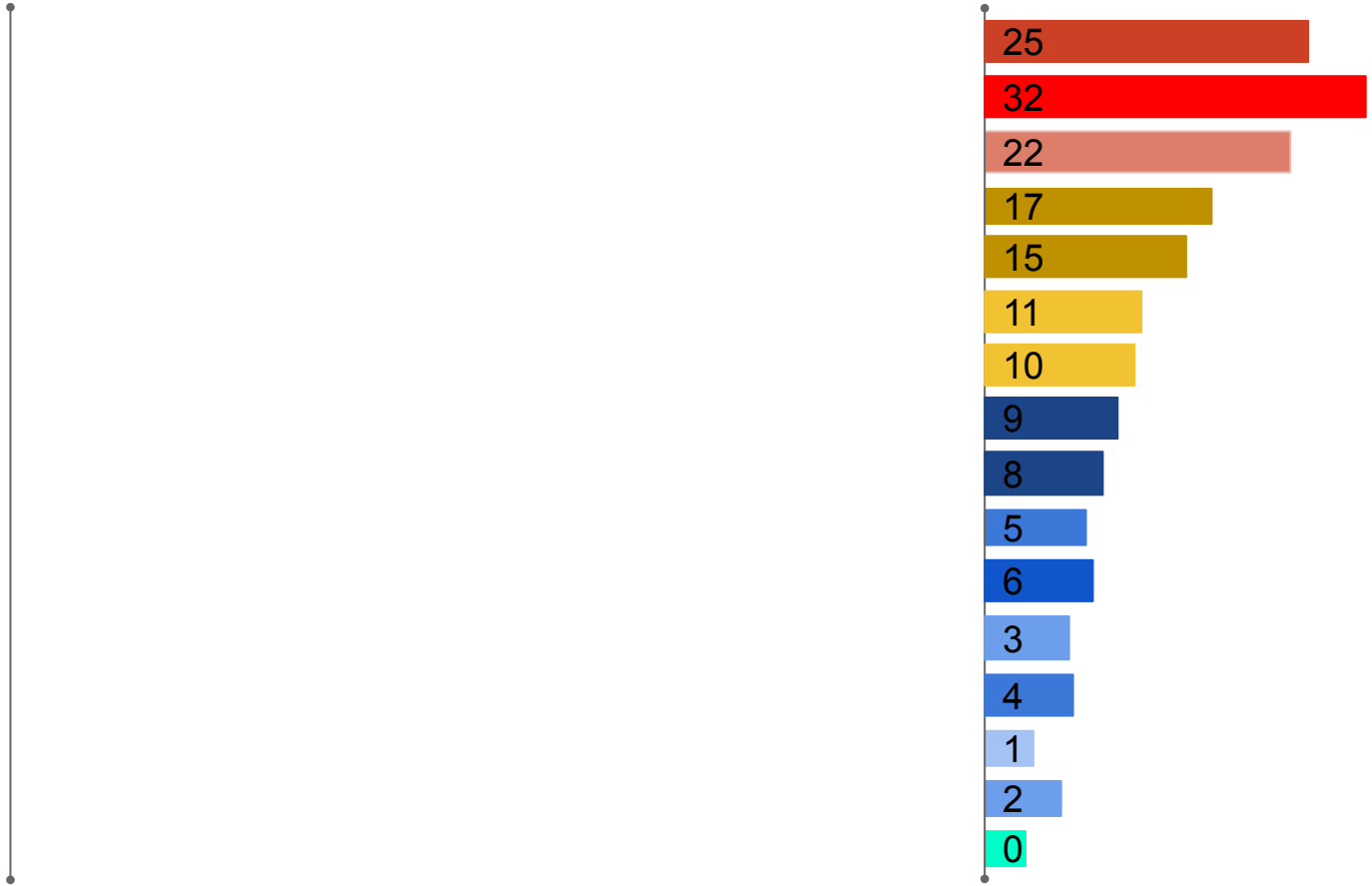




- 0 is on the range [0, 32], push to B (pb) and rotate it (rb)



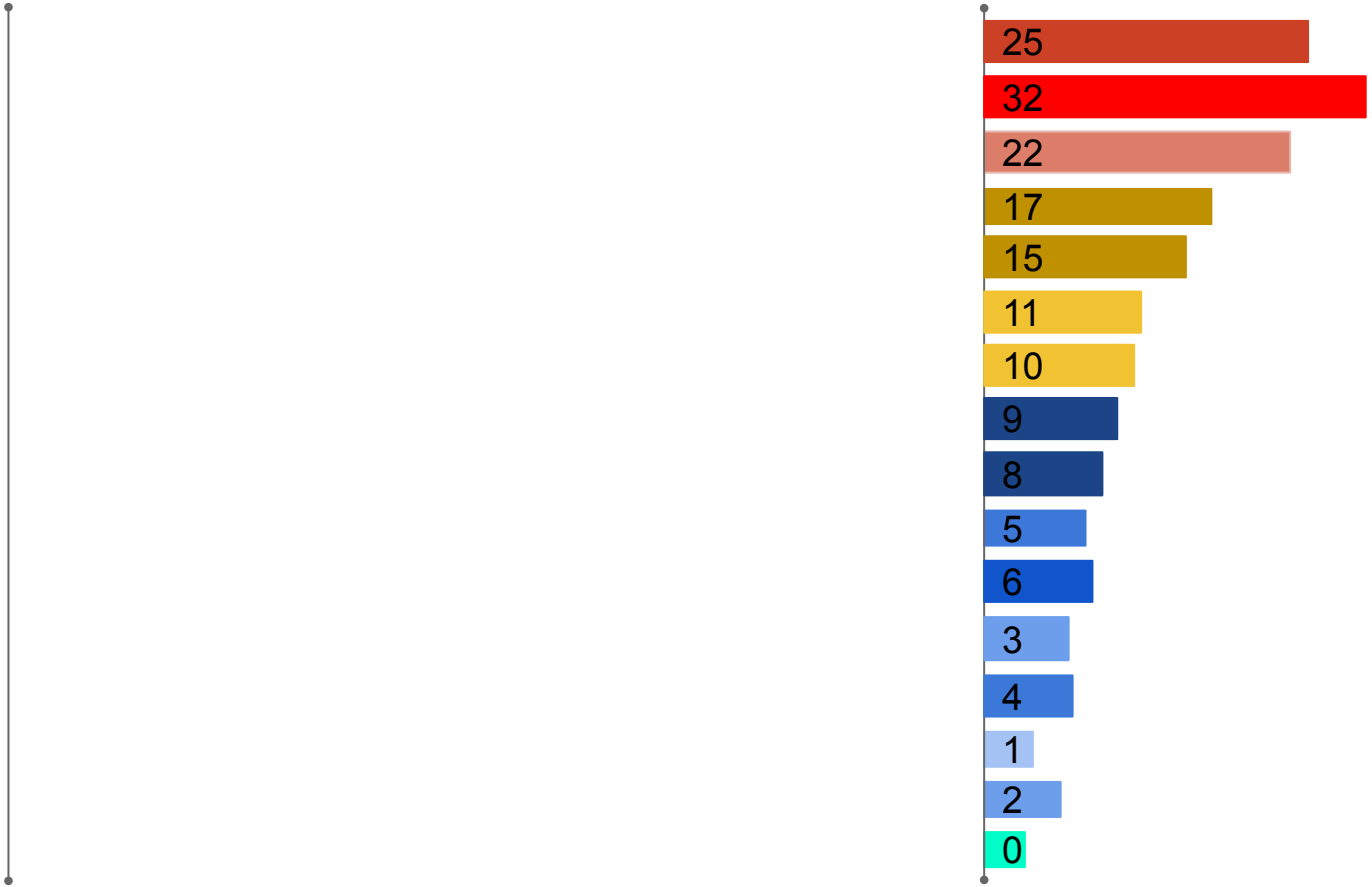
🎉 END OF THE FIRST STAGE 🎉



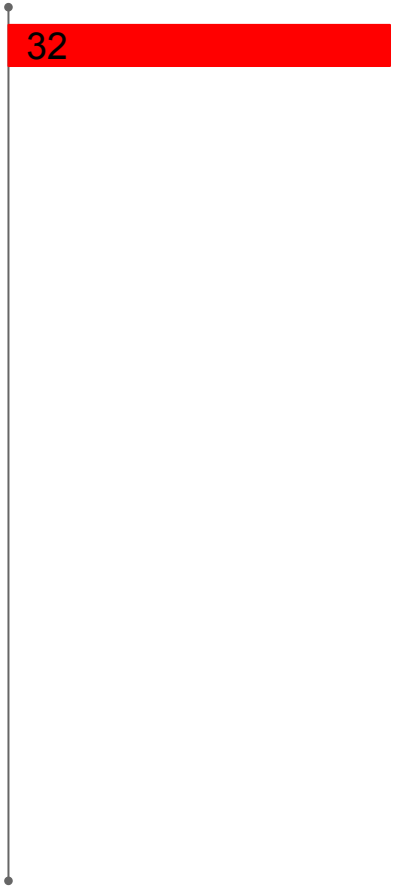
Stage 2: push back all elements to a (sorted)

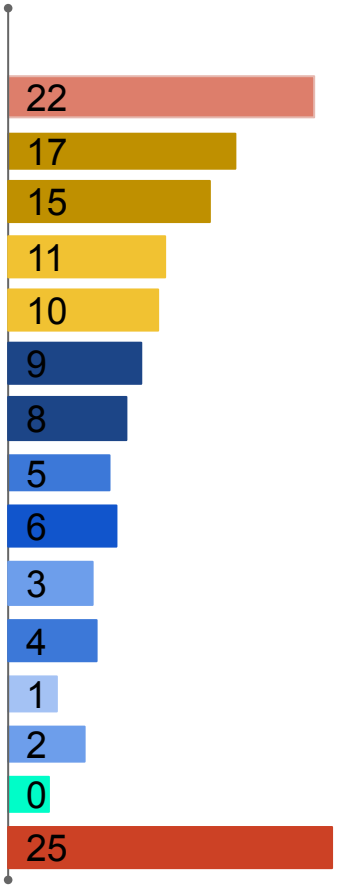
The idea here is to find the max value on the stack B and push it to A, But there is a tricky way to make thing easy, which is usage the bottom of stack A to put an element on it until become the max so just call (rra) to become on the top of A, also we can put the elements that is greater than this current element on bottom of A, So let's discuss that with some chart.

- 32 is the expected max value so we will use ra one time and push it to A

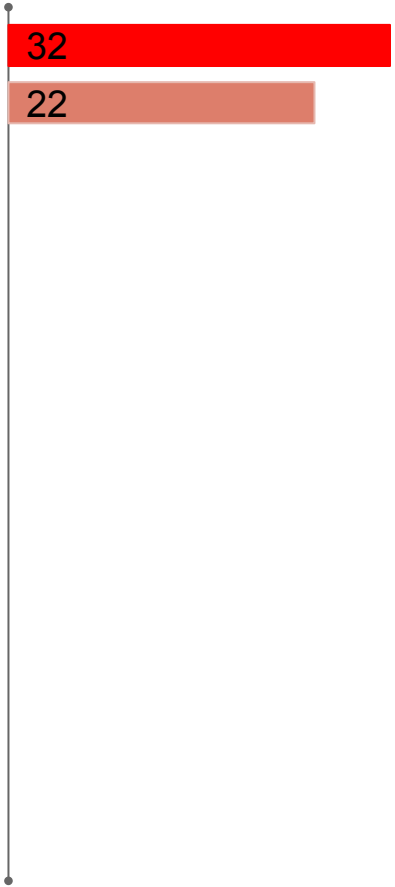


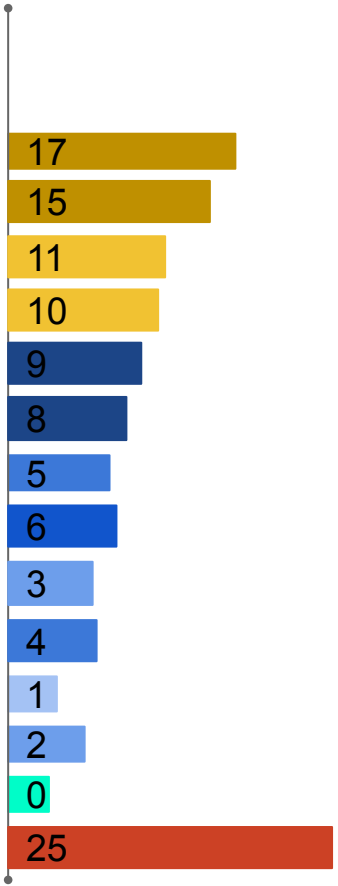
- 32 is the expected max element so let just push it to A



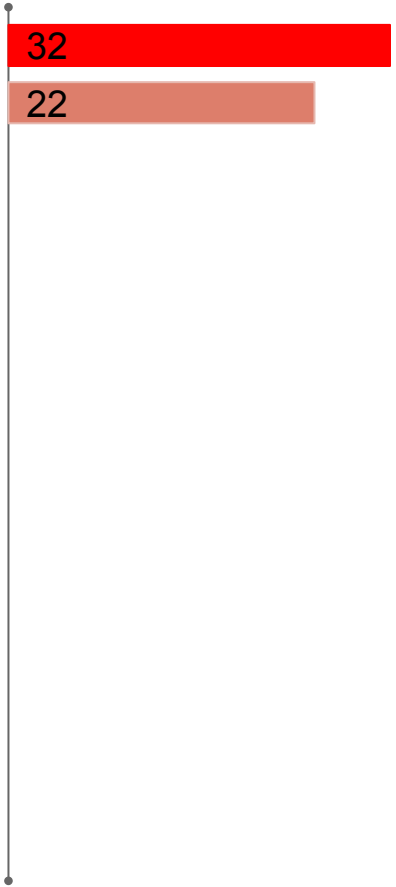


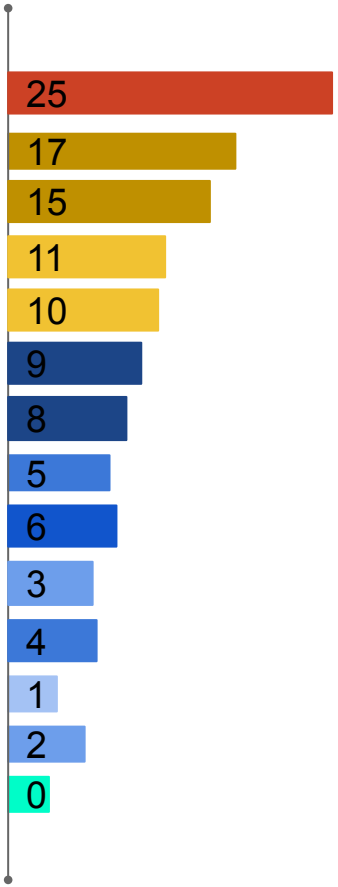
- 32 is the expected max element so let just push it to A



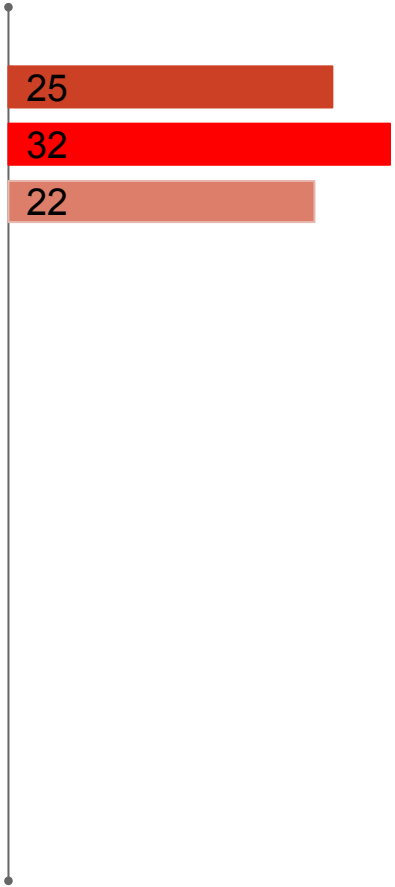


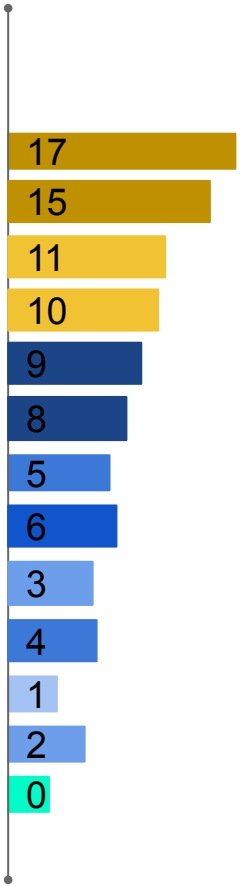
- 32 is the expected max element so let just push it to A



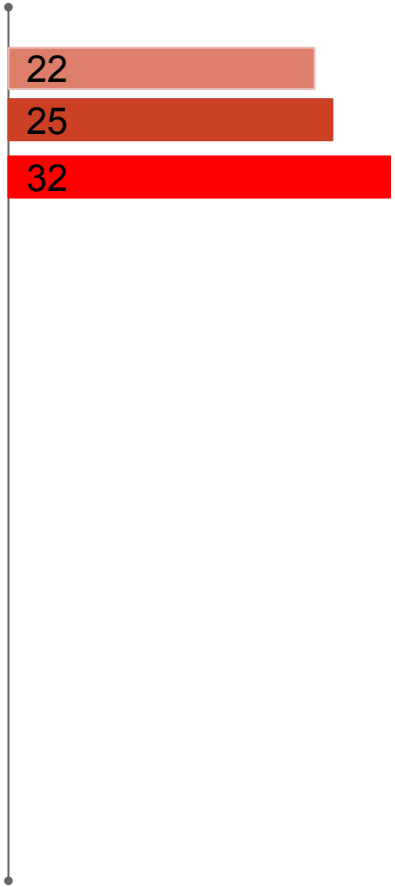


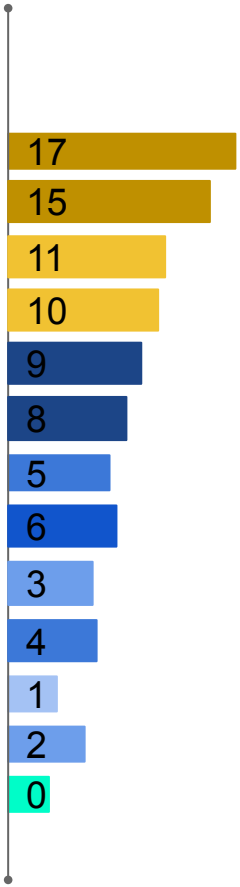
- 32 is the expected max element so let just push it to A



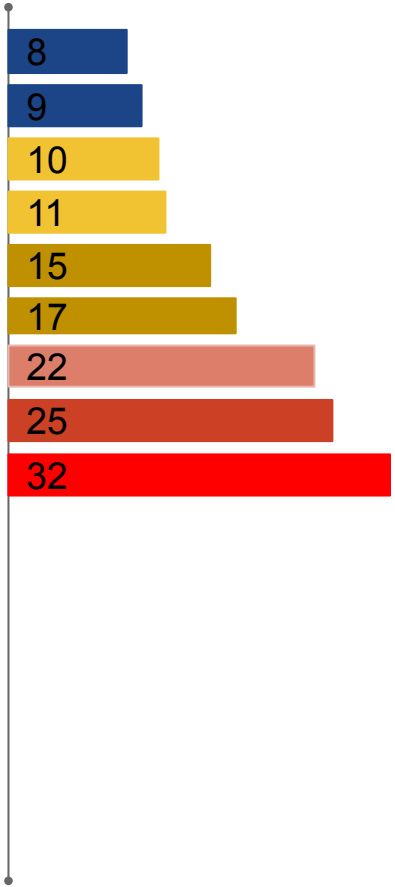


- ra

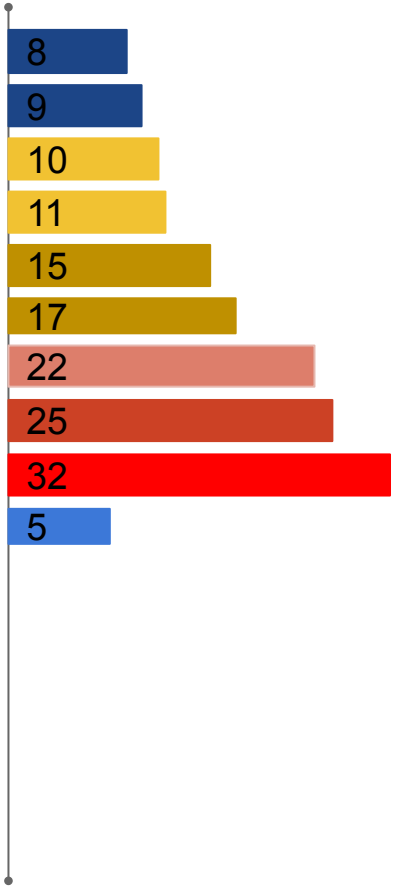




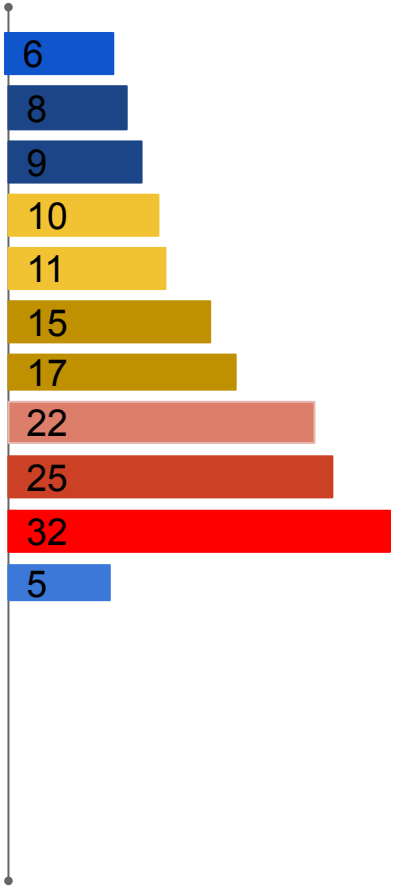
- Even 5 is not the expected max we will push it to A and rotate A, pa ra

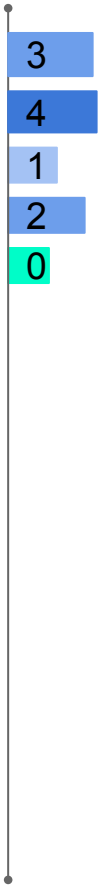


- 6 is the max, push to A

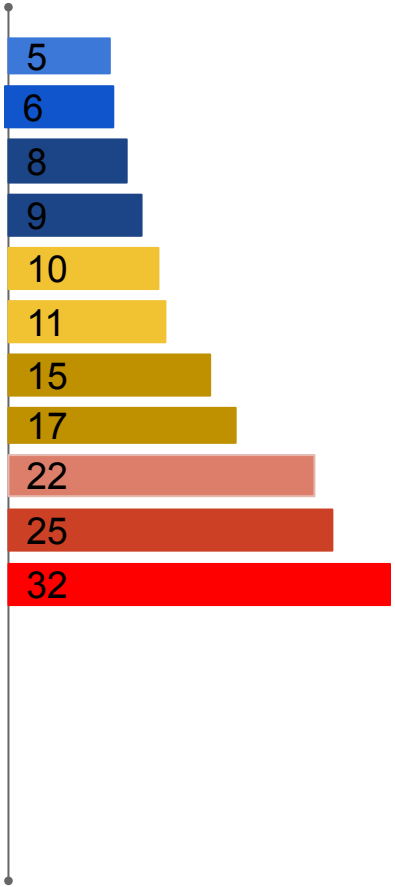


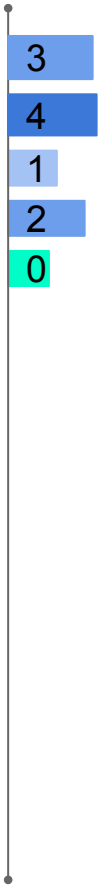
- 5 is the expected max, but we won’t find it on B, So we should just reverse rotate A rra



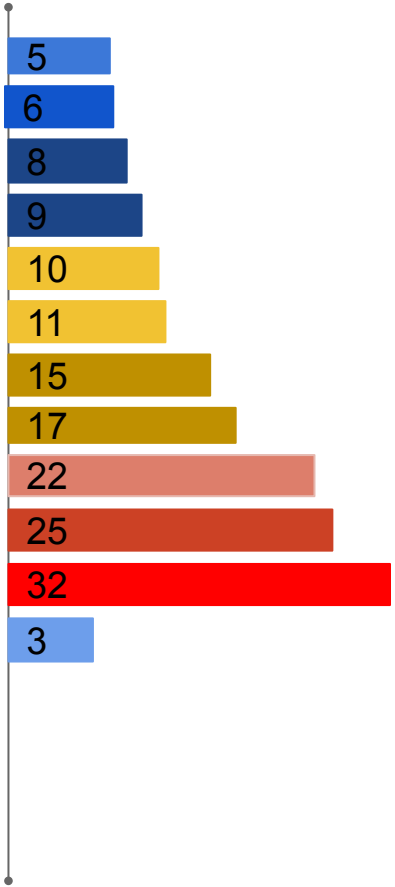


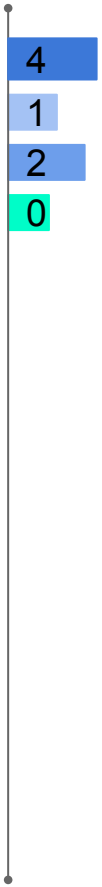
- 3 is not the max but we will push it to A and rotate it



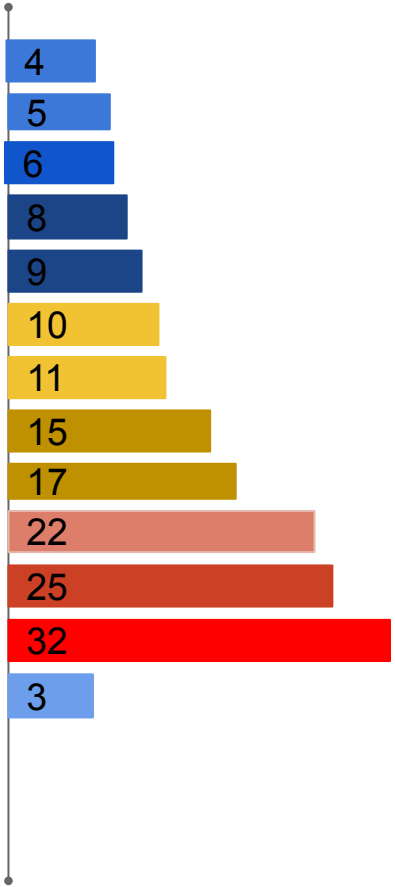


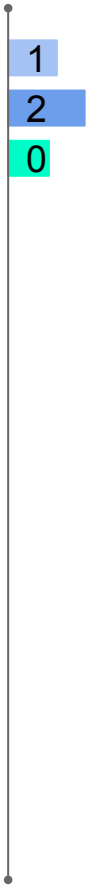
- 4 is the max let push it to A



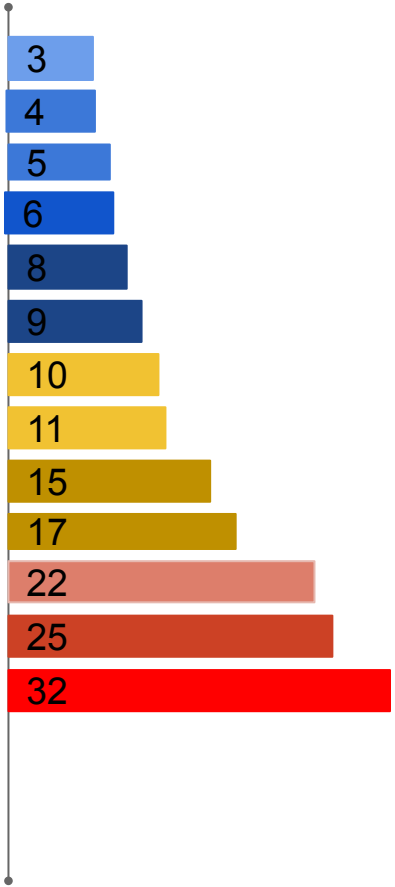


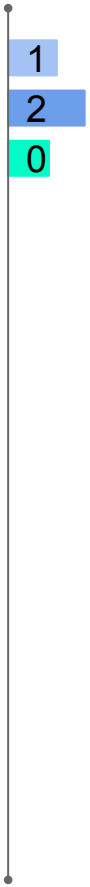
- 3 is the expected max, and it's on A, so rra



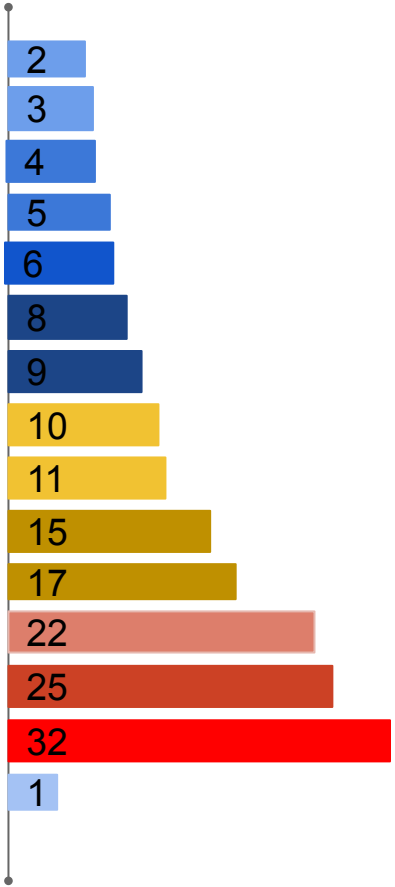


- Push 1 to A, and rotate A

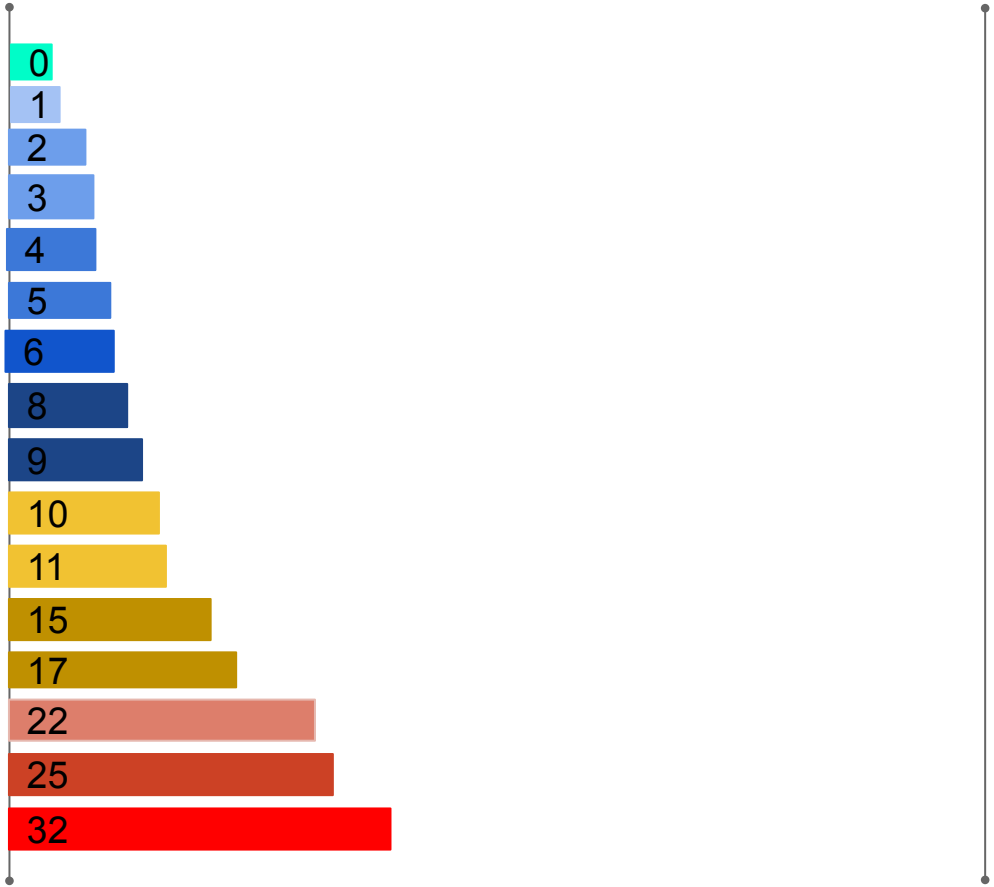




- Reverse rotate A



WE DONE 🎉



BY OUSSAMA KHIAR (OKHIAR)

 ossamakhiar

 okhiar

 ossamakhiar

 ossama.khiar@gmail.com