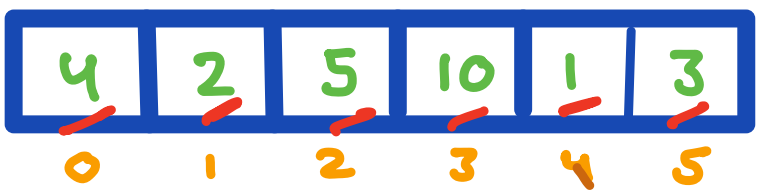
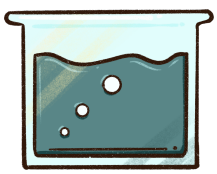
Sorting

Given the following array, sort it.





1 2 3 4 5 10

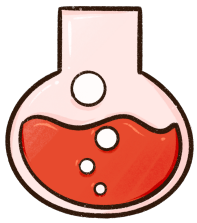
Inbuilt sorting methods in python

- Sorted
- Sort

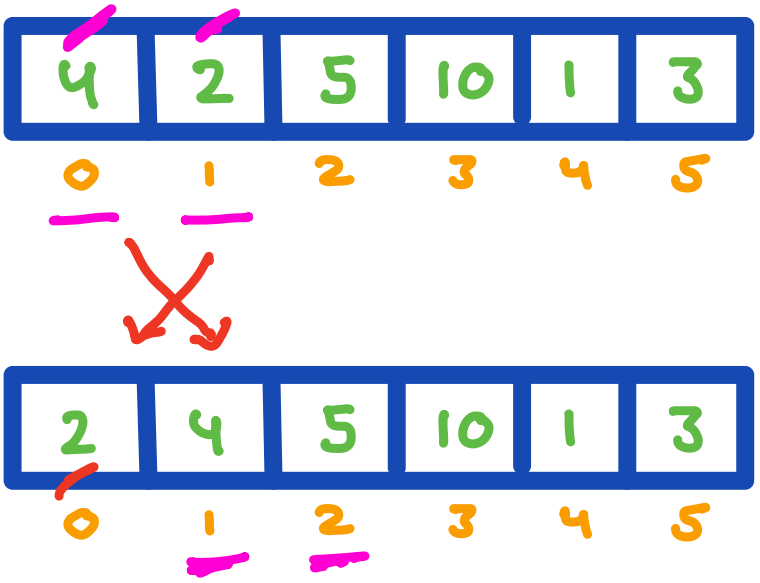
```
a = [4,2,5,1]
a = sorted(a)
```

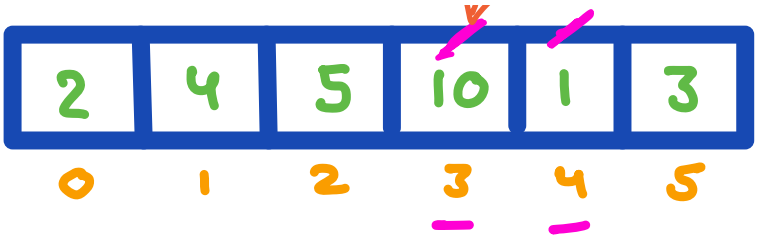
```
a = [4,2,5,1]
a.sort()
```

Bubble Sort

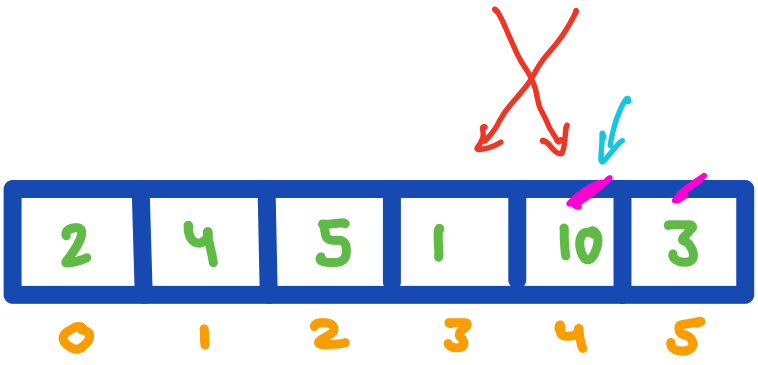


Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.





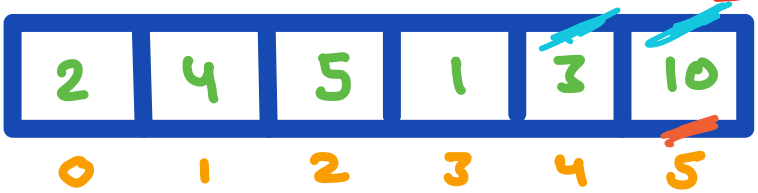
Swap 1 and 10



Swap 3 and 10

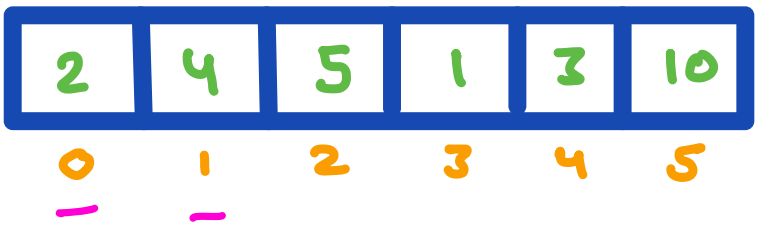
a[j], a[j+1]

n-1

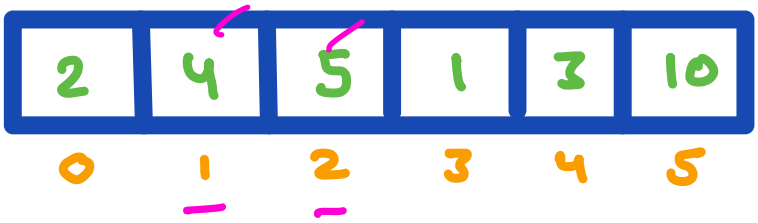


Greatest element at last position

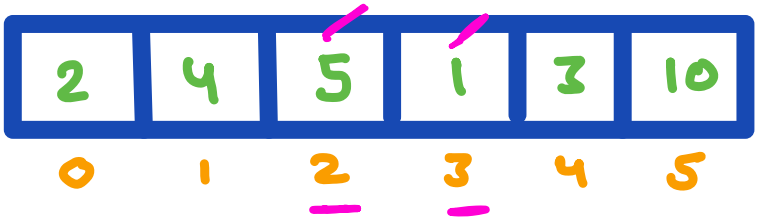
Array after 1st iteration



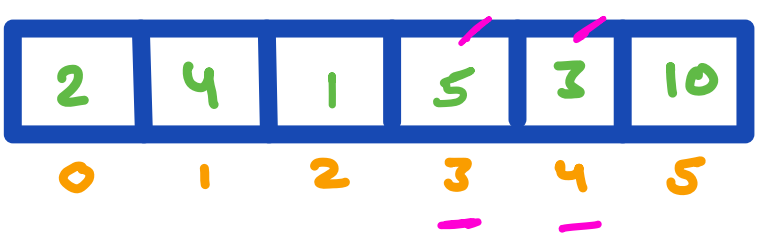
Do nothing



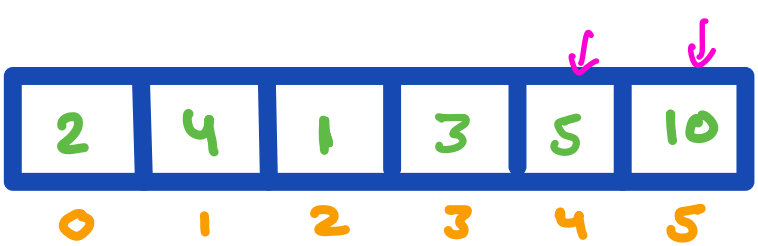
Do nothing



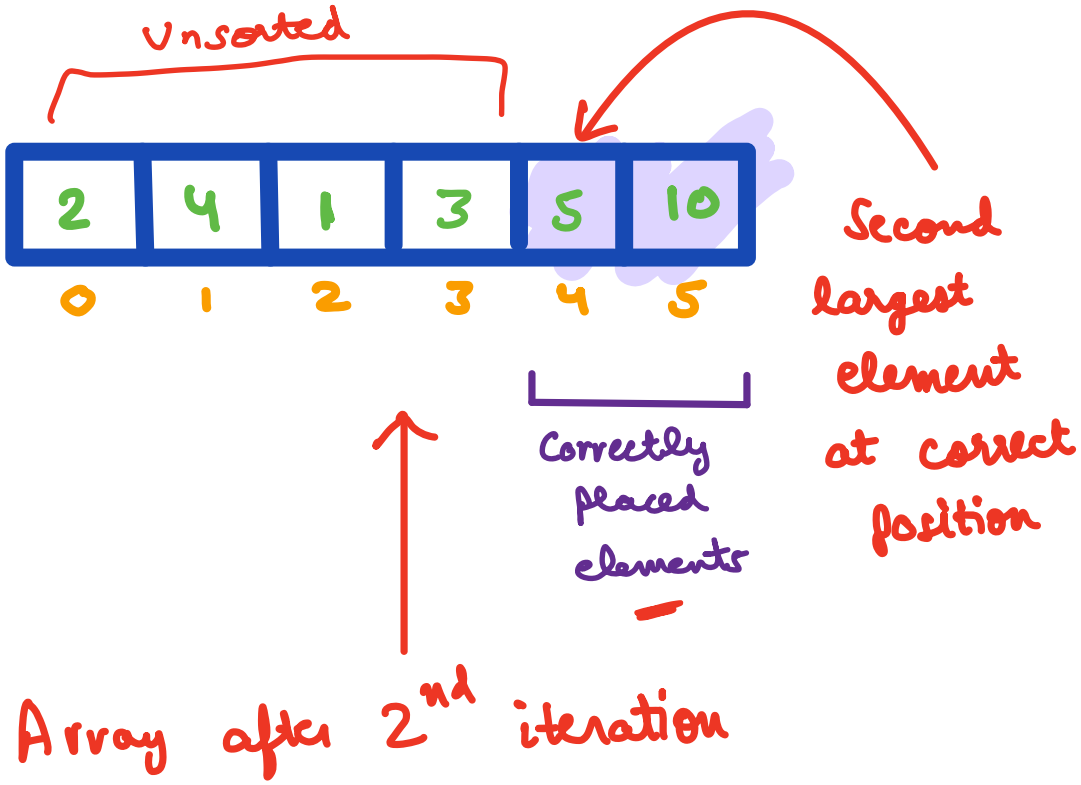
Swap 5 and 1

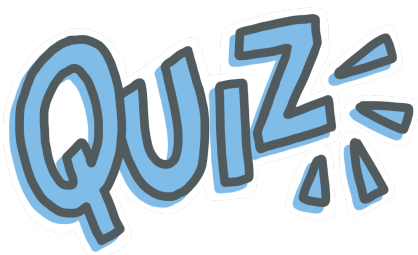


Swap 5 and 3

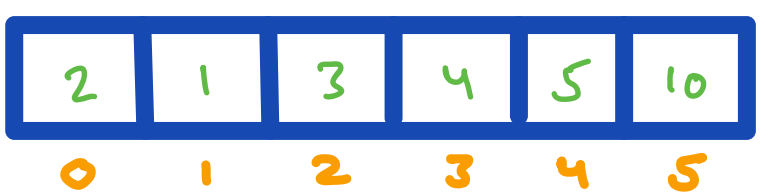


Do nothing





What will the array be after 3rd iteration ?



→ N iterations

→ In each, we place one element correctly

→ In each iteration,

Nested loop [We go about comparing adjacent elements

Time & Space Complexity

Time - $O(N^2)$

N - Size of array / list

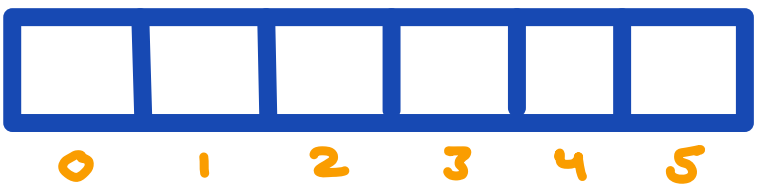
Space - $O(1)$

↓

Extra space

N = 1000

Searching



Binary Search

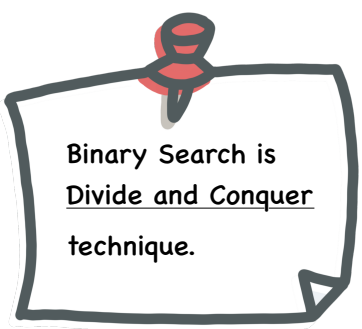
Prerequisite

- Array must be sorted

x = target

Algorithm

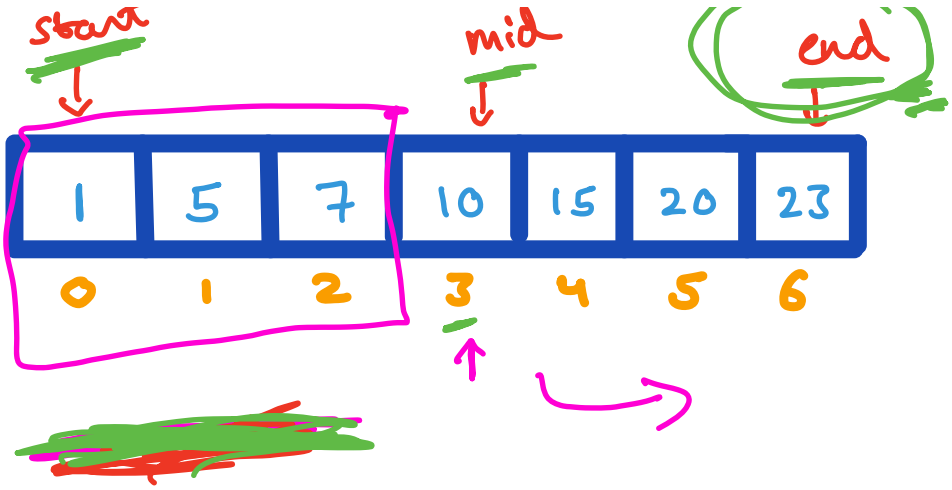
- Find the middle element and compare it with x.
- If middle element is equal to x, end the search. You found the element.
- If middle element is greater than x, search for x in left half.
- If middle element is less than x, search for x in right half.
- Repeat the above steps till you find x, or till you exhaust the array.



Compare mid element with the target.

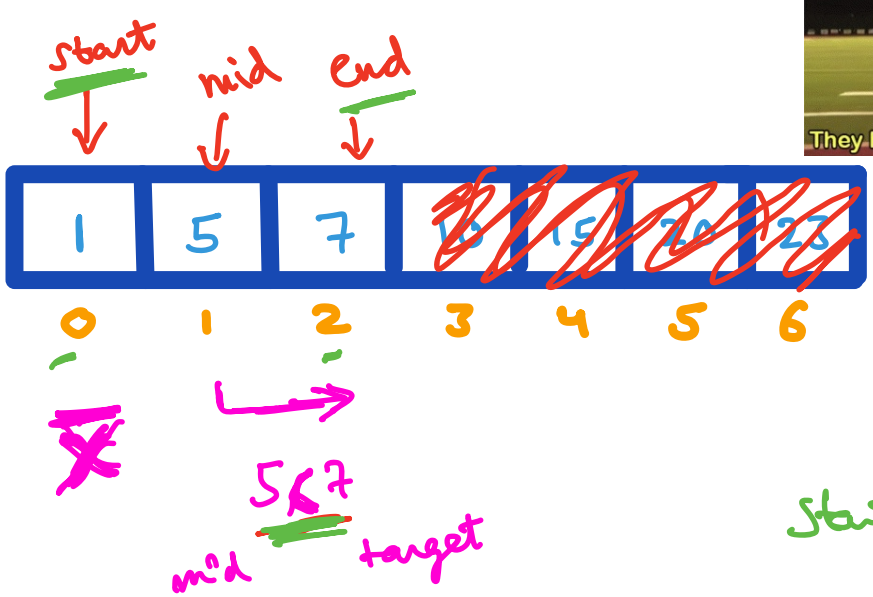
1097

Sorted



Find the index of 7

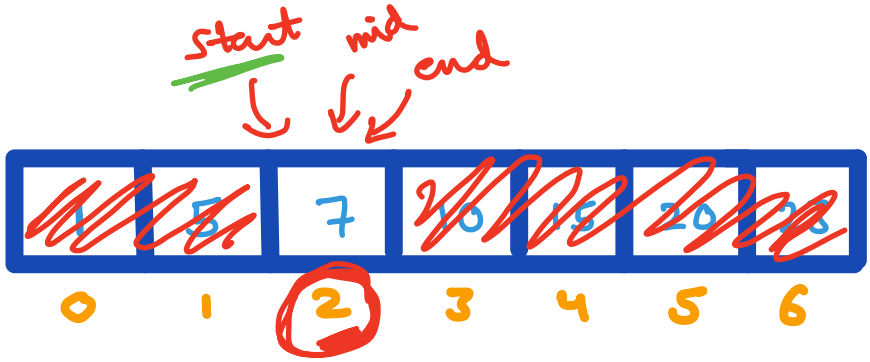
target < mid < right



5 < 7

mid target

start = mid+1



mid element = target

done

Search in Unsorted array

Sort it, Binary Search
O(N log N) + O(log N)

Linear Search - O(N)

Pseudocode

```
a = [. . . . - -]        <= Sorted
start = 0
end = n-1
while start <= end:      <= Perform search till you have elements left
    mid = (start + end) / 2
// Compare mid element and target
```

if a[mid] == target

↳ Found!!

else if a[mid] > target

// Left half

end = mid - 1

else

// Search in right half

start = mid + 1

Lower bound & upper bound

target = 4

lower bound

upper bound

Can we use recursion ?

Yes.

HW

Write a recursive function to perform binary search on a list.

Square root using binary search

Given a positive integer, find its square root without using any inbuilt functions.

$100 \rightarrow 10$

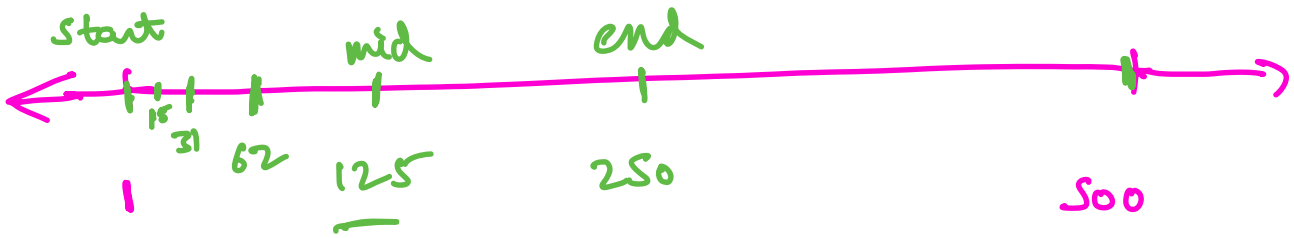
$\sqrt{100} = 10$

$\sqrt{144} = 12$

$N \leq 10^{10}$

N = 500

Worry only about the integer part first.



if $mid^2 <= n$ ⇐ Hurray?! (Atential answer)

→ ans = mid

// right half

else if $mid^2 > n$

// left half

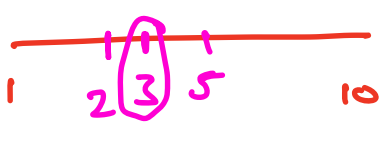
n = 10

$\sqrt{10} = 3$

ans = mid

$3^2 = 9 < 10$

$mid^2 <= n$



n = 500

mid = 22 → $22^2 = 484$

$\sqrt{500} = 22.3$

0 to 9

ans = 22

0.1

$22.1^2 \leq 500$ ✓

+0.1

$22.2^2 \leq 500$ ✓

+0.1

$22.3^2 \leq 500$ ✓ → 22.3

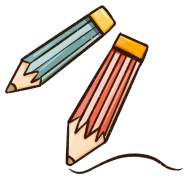
+0.1

$22.4^2 \leq 500$ ✗

$(22.4)^2 > 500$

$(22.31)^2$

Once we are done with finding an integral part, start computing the fractional part.



Initialize the increment variable by 0.1 and iteratively compute the fractional part up to P places. For each iteration, the increment changes to 1/10th of its previous value.