App Academy

Sorting Algorithms

Bubble Sort

```
def bubble_sort(nums):
    len = len(nums)
    for i in range(len):
        for j in range(len):
            if nums[j] > nums[j+1]:
                let tmp = nums[j]
                nums[i] = nums[i+1]
                nums[j+1] = tmp
    return nums
```

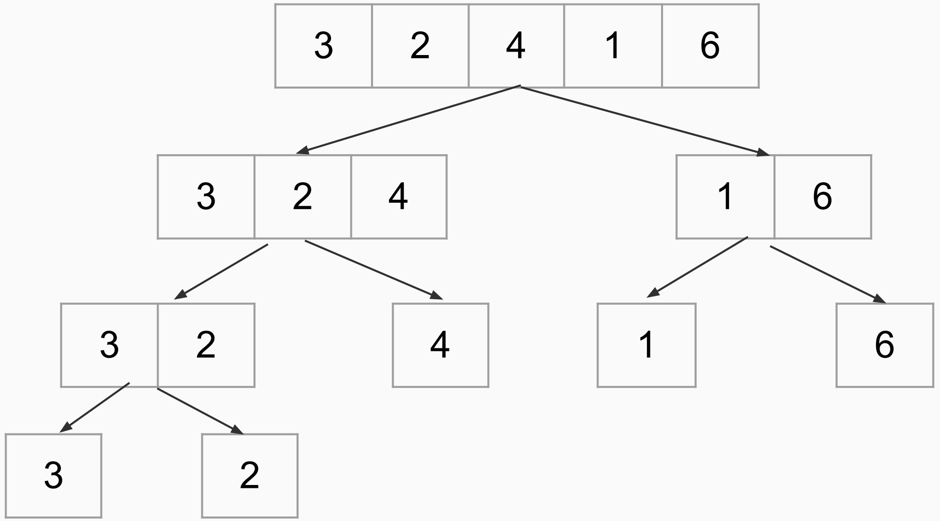
Runtime: $O(n^2)$

visualization: https://visualgo.net/en/sorting

- This sorting algorithm scans through the array several times, taking the largest number and placing it onto the end.
- This algorithm is highly inefficient, and you would never want to sort this way unless specifically asked to.

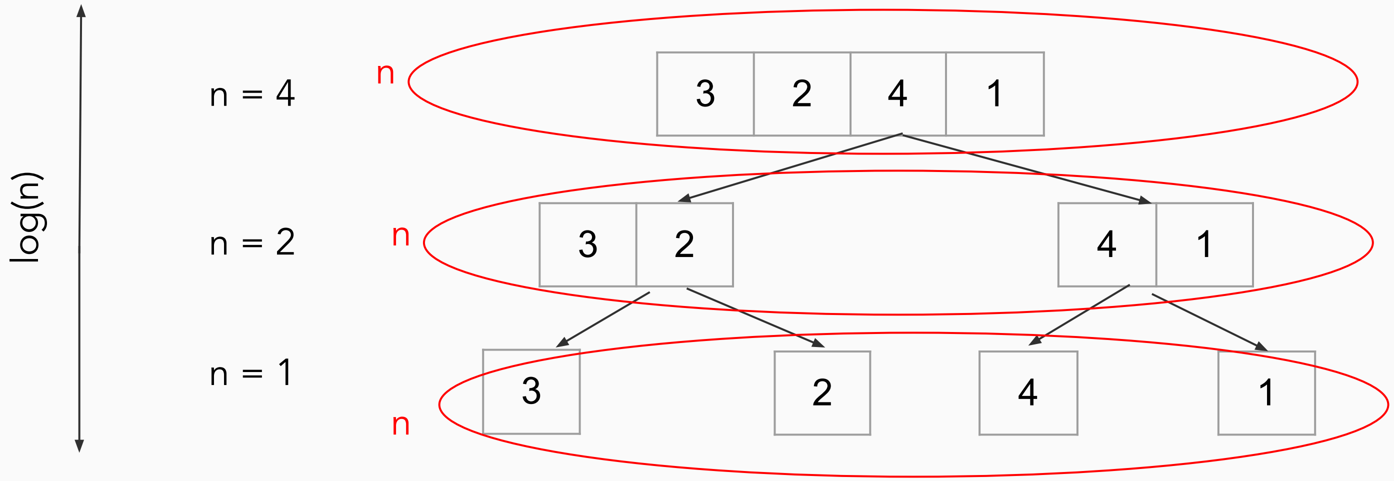
Merge Sort

Demo:
https://leetcode.com/problems/sort-an-array/



- Runtime: O(n*log(n))
- visualization: https://visualgo.net/en/sorting
- This algorithm splits an array in half continuously and then returns the sorted halves.
- This sounds a lot like recursion!
  - Our base case is when we can no longer split any further; i.e. when array.length = 1

Merge Sort (cont.)



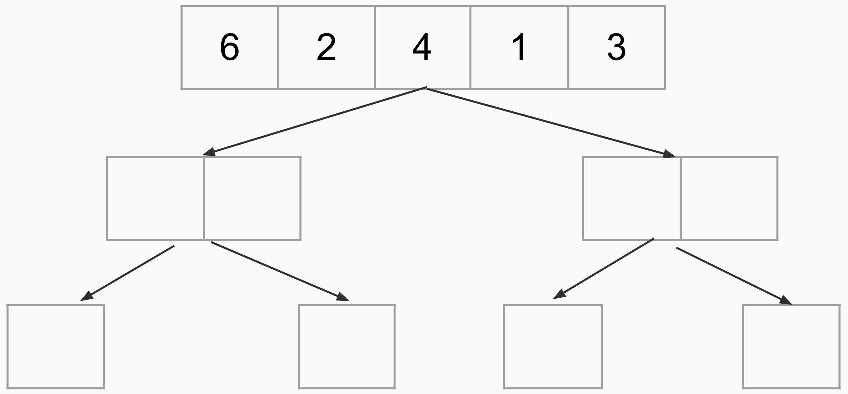
- Notice how the size of n is divided by 2 at each stage of the stack? That means the stack is log(n) deep!
- Also notice that how wide this tree is the length of n
- So we are doing n work log(n) times. That's how we get n*log(n)!!!

Merge Sort (cont.)

```
def sortArray(nums):
    if len(nums) < 2: return nums

    mid = len(nums) // 2
    left = self.sortArray(nums[0:mid])
    right = self.sortArray(nums[mid::])

    return merge(left, right)
```

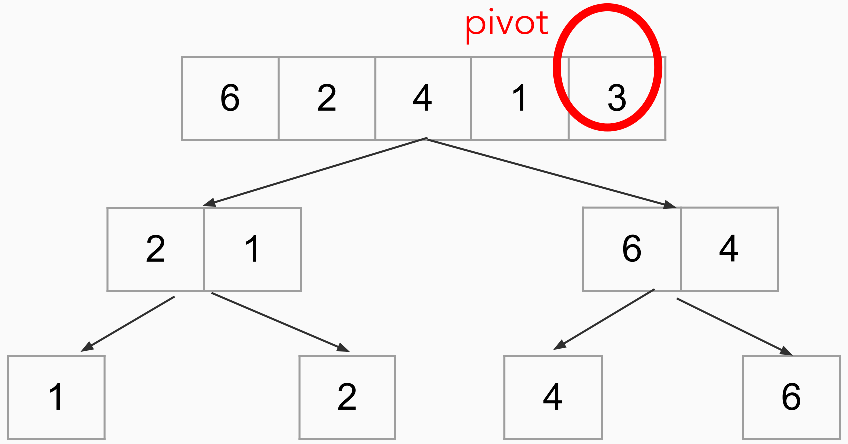
```
def merge(left, right):
    merged = []

    while len(left) and len(right):
        if left[0] < right[0]:
            merged.push(left.pop(0))
        else:
            merged.push(right.pop(0))

    return merged + left + right
```

Quick Sort

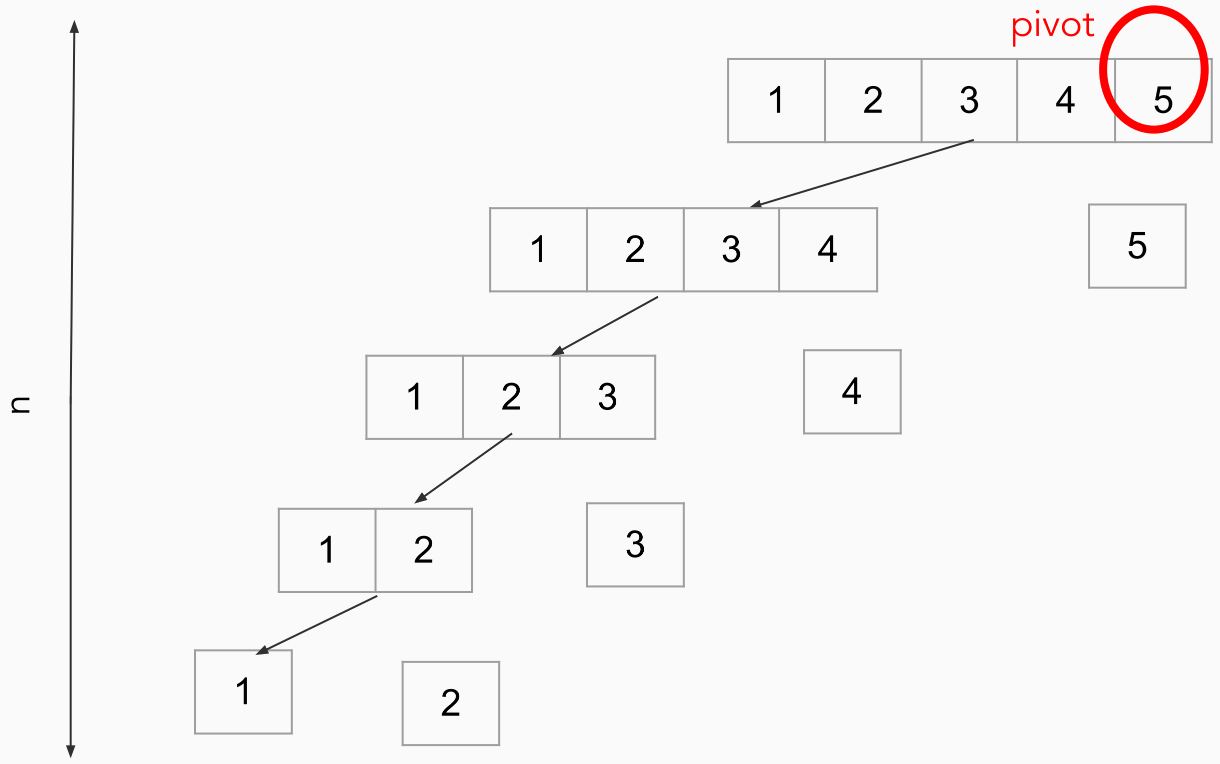


- Runtime:
  - WORST CASE: O(n^2)
  - Amortized: (n*log(n))
- Unlike merge sort which uses a temp array to merge, quicksort can sort IN PLACE with no extra memory.
- This algorithm selects a pivot and then partitions all values into “lesser” and “greater” buckets. It does this until there is only one element at most in either the lesser or greater sides and then concatenates them with the pivot.

Quick Sort (cont.)



Quick Sort - worst case

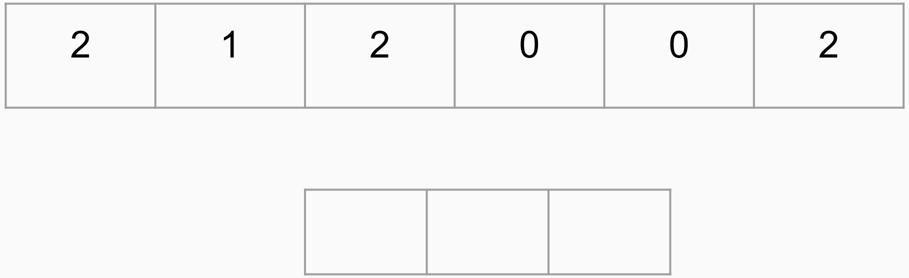


Quick Sort (cont.)

```
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        def quickSort(nums, start, end):
            if end-start+1 <= 1: return nums

            pivot = nums[end]
            left = start

            for i in range(start, end+1):
                if nums[i] < pivot:
                    tmp = nums[left]
                    nums[left] = nums[i]
                    nums[i] = tmp
                    left += 1

            nums[end] = nums[left]
            nums[left] = pivot

            quickSort(nums, start, left-1)
            quickSort(nums, left+1, end)

            return nums

        start=0
        end=len(nums)-1

        return quickSort(nums, start, end)
```

- Note that the example to the left will not pass all test cases. In order to better optimize, we need to select randomized pivots instead

Bucket Sort



- Runtime: O(n)
- ONLY ALLOWED UNDER CONSTRAINTS:
    - We must guarantee that all values fit within a finite range.
- As we iterate through the array, we can simply increment a value in an array.

Bucket Sort

2	1	2	0	0	2

2	1	3

- Runtime: O(n)
- ONLY ALLOWED UNDER CONSTRAINTS:
  - We must guarantee that all values fit within a finite range.
- As we iterate through the array, we can simply increment a value in an array.
- Demo: Sort Colors

Questions?

Let’s practice!

- Review
  - Sort Colors
  - Sort an Array - implement merge sort
  - Sort Characters by Frequency
- Bonus
  - Top K Frequent Elements