Quicksort

Suppose you want to sort a list from

$$[3, -2, -1, 0, 2, 4, 1]$$

$$[-2, -1, 0, 1, 2, 3, 4]$$

Quicksort is a recursive function which accepts 3 arguments.

$$qs(arr, l, r)$$

- arr → given array
- l → starting index to sort
- r → last index to sort.

Suppose $qs(arr, 4, 6)$

$$arr = [3, -2, -1, 0, 2, 4, 1]$$

L R

output = [3, -2, -1, 0, 1, 2, 4]

sorted portion

We need to firstly declare the base case.

```
if l >= r :
```

arr = [3, -2, -1, 0, 2, 4, 1]

L

R

L = R means the section to sort has only "1 element"

L > R means the section to sort has

"no element"

This means the section of array is "already sorted" we don't need to enter into the if statement & the function is completed.

```
if l >= r :
```

if that condition is "not met" it enters the recurcive function.

```
def qs (arr, l, r) :
    if l >= r :
        return
```

$$p = \boxed{\text{partition}(arr, l, r)}$$

recurcive function starts with partition which accepts 3 arguments (arr, l, r) just like qs and the goal is to divide the section into "two groups" and put pivot right between these "two groups" and in the end, it will return the index of the pivot.

$$arr = [3, -2, -1, 0, 2, 4, 1]$$

lets say we choose the last element to be pivot.

$$pivot = 1$$

And after partition function, the array

will look like this.

arr = [3, -2, -1, 0, 2, 4, 1]

- smaller than "1"

- bigger than "1"

arr = [-2, -1, 0, 1, 3, 2, 4]

We don't care about the order within each group.

But we know for a fact that "1" is in right place & that section of array is sorted.

We call qs again to the left group and right group which will contain partition

function.

qs (arr, l , p-1)

from l (the starting index to solve) up to index of p-1.

qs (arr, p+1 , r)

from index of p+1 up to r (the last index to sort)

We recurse this function until

$$\boxed{l >= r}$$

How does partition Work?

We will have to track 2 index

i & j

arr = [2, 1, 4, 13, 14, 12, 3, 16, 5, 2, 10]

j is part of for loop which will iterate from first index up to index of p − 1.

We want to make sure i is always pointed to last number which is lower than pivot and i+1 will be the numbers which are higher than pivot. for example,

arr = [2, 1, 4, 13, 14, 12, 3, 16, 5, 2, 10]

i j

We want to always satisfy "two condition"

1 - all the numbers from beginning up to i will be "lower" than pivot.

2 - all the numbers between i & j not including i are "greater than or equal to pivot".

arr = [2, 1, 4, 13, 14, 12, 3, 16, 5, 2, 10]

i j

if these two conditions are met we move i to the next index.

j

$$arr = [\underbrace{2, 1, 4}_{}, \underbrace{13, 14, 12}_{}, 3, 16, 5, 2, 10]$$

i $\quad$ j

One condition is not met anymore which is

2 - all the numbers between i & j not including i are "greater than or equal to pivot".

We can fix that by moving i to next index & swapping the elements @ i & j

$$arr = [\underbrace{2, 1, 4}_{}, \underbrace{13, 14, 12}_{}, 3, 16, 5, 2, 10]$$

i $\quad$ j

$$arr = [2, 1, 1, 3, 11, 12, 13, 16, 5, 2, 10]$$

i j

if these two conditions are met we move j to the next index.

Repeat until j reaches $p-1$.

$$arr = [\underbrace{2, 1, 4, 3, 5, 2}_{i}, \underbrace{13, 16, 14, 12, 10}_{j}]$$

Then we move the pivot to right index which is $i+1$. We swap P with $i+1$.

The CODE

first choose a pivot.

```
def partition(arr, l, r):
    pivot = arr[r]
```

Then define index of i

```
    i = l - 1
```

Then the for loop

```
    for j from l up to r - 1:
        if arr[j] < pivot:
            increase i by 1
            i += 1
```

and swap places

```
            arr[j], arr[i] = arr[i], arr[j]
```

Continue with for loop until j is at r - 1.

To have pivot in the right place,

swap arr[i+1] and arr[r]

arr[i+1], arr[r] = arr[r], arr[i+1]

return index of pivot

return i+1

TIME COMPLEXITY

Worse case is when it is sorted or a lot of duplicates.

[1, 2, 3, 4, 5, 6, 7]

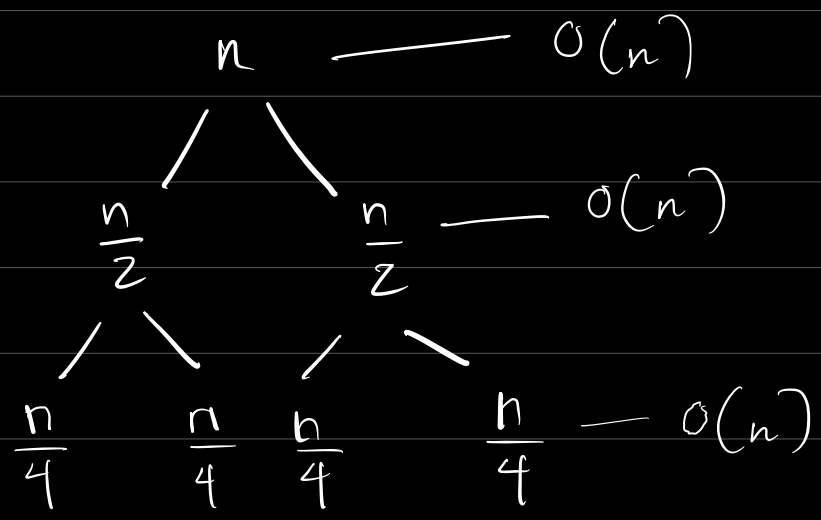
[2, 2, 2, 2, 2, 2, 2]

$n + (n-1) + (n-2) \ldots (1)$

$$O(n^2)$$

Best case is when we choose pivot right which is the median of the array.

$$[-2, 3, -1, 5, 4, -3, \underline{\underline{0}}]$$



$\log(n)$

1 1 1 1 . . . . . .

$O(n \log n)$

Average is also

Assuming there is no duplicate
And ordering in the array is random

CHOOSE A RANDOM ELEMENT AS PIVOT

OR TAKE MEDIAN
OF THREE ELEMENT