#algorithms

Quicksort Benchmarking README

Code Breakdown

Quicksort Implementation

```
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
```

python syntax

- go through the array
- check the if condition
- if true, then add that x to the array called "left"

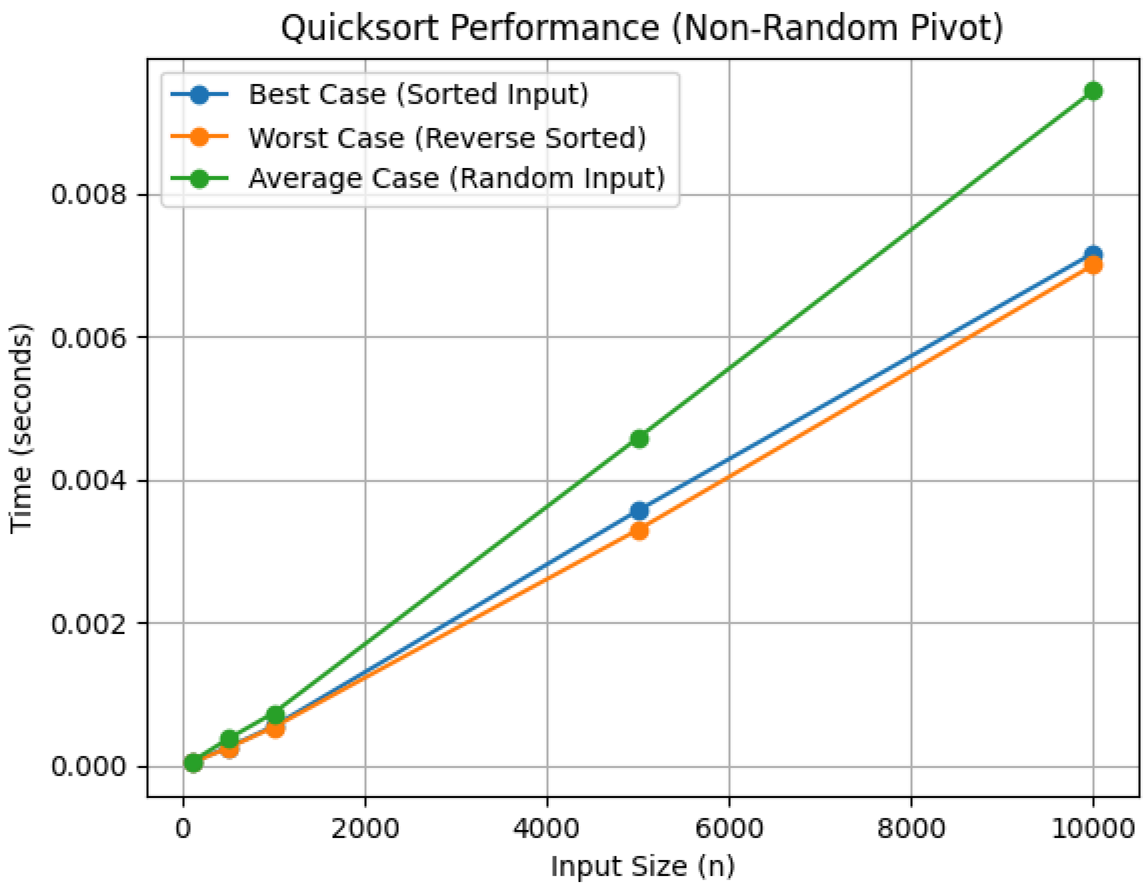
technically this states that we're making a subarray with all elements that meet the condition we're looking for.

Generating Input Cases

```
best_case = list(range(n))  # Already sorted input
worst_case = list(range(n, 0, -1))  # Reverse sorted input
average_case = [random.randint(0, n) for _ in range(n)]  # Random input
```

- **Best case:** Sorted list `[0, 1, 2, ..., n-1]`
- **Worst case:** Reverse sorted list `[n-1, n-2, ..., 0]`.
    - as I did in a previous assignment, range(start, stop, increment)
    - (start at n, stop at 0, decrement by 1)
- **Average case:** Randomly generated list.

Performance Graph



3 Mathematically derive the average runtime complexity of the non-random pivot version of quicksort.

we're taking a midpoint (/2)

left and right subarray (2)

take the midpoint of those arrays (1/2)

repeat

Classic example we've gone over before

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

2t from the fact that we're doing it on left/right array and executing through another n/2 times. + the original array n. Specifically comparing each element to the pivot

Expanding by recurrence

$$T(\frac{n}{2}) = 2T(\frac{n}{4}) + c(\frac{n}{2})$$

substitute back in

$$T(n) = 2(2T(\frac{n}{4}) + c(\frac{n}{2})) + c(\frac{n}{2})$$

$$T(n) = 4T(\frac{n}{4}) + 2cn$$

we can already see it's going roughly

$$T(n) = 2^k T(n/2^k) + kcn$$

don't care.

in our base case with 1 element, the time is 1. T(1) = 1
so when is it 1?

$$\frac{n}{k^2} = 1$$

$$n = k^2$$

$$k = \log_2 n$$

plug it into the equation

$$T(n) = 2^{\log_2 n} T(1) + (\log_2 n)cn$$

$$T(n) = O(n) + O(n \log n)$$

we take the biggest value

$$O(n \log n)$$