Algorithms and Data Structures

Assignment 5: QuickSort Algorithm: Implementation, Analysis and Randomization

Nguyen The Duy Khanh

005019181

Table of Contents

1. Randomized Quicksort Analysis

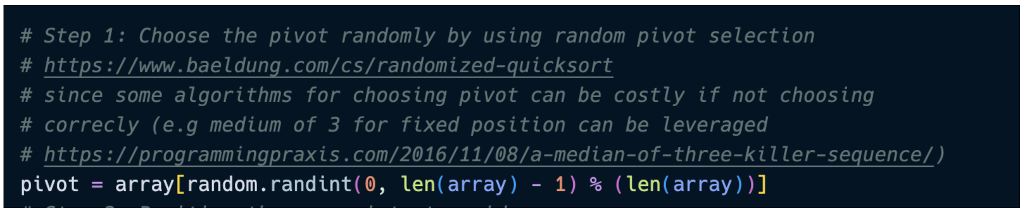
- Each sort has its own use case, and the same can be said for QuickSort, where attackers can leverage its weakness to perform a DDOS attack and crash the server (e.g choosing the pivot poorly in a sorted array [1], a median of three killer sequence [2]). Therefore, in order to leverage the Quick Sort advantage, we need to resolve this weakness, and one of the solutions is using Random Pivot Selection[3], where we **increase the randomness and reduce the likelihood of consistently picking a poor pivot**.

a. Practical Implementation

- The Quick Sort algorithm with Randomized Pivot Selection has three steps:
    - **Step 1:** Choose a pivot (in this case, randomized pivot selection)

$$\mathbf{pivot} = \mathbf{L} + (\mathbf{random}() \mod (\mathbf{R} - \mathbf{L} + \mathbf{1}))$$

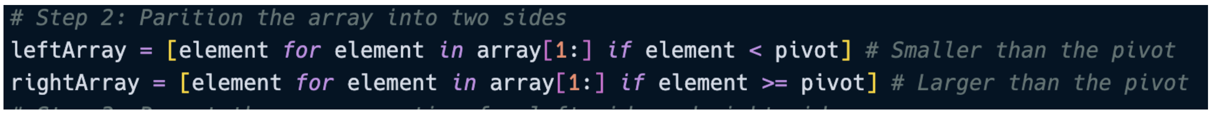
where L is the index of the first element in the range, and R is the index of the last element in the range.



```
# Step 1: Choose the pivot randomly by using random pivot selection
# https://www.baeldung.com/cs/randomized-quicksort
# since some algorithms for choosing pivot can be costly if not choosing
# correcly (e.g medium of 3 for fixed position can be leveraged
# https://programmingpraxis.com/2016/11/08/a-median-of-three-killer-sequence/)
pivot = array[random.randint(0, len(array) - 1) % (len(array))]
```

Figure 1: Randomized Pivot Selection

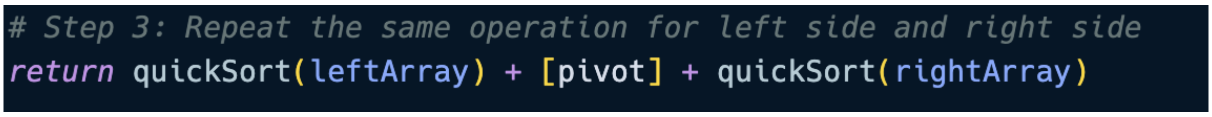
- **Step 2:** Partition the array, which includes splitting and reordering the array so that all elements smaller than the pivot are on the left and all elements larger than the pivot are on the right.



```
# Step 2: Parition the array into two sides
leftArray = [element for element in array[1:] if element < pivot] # Smaller than the pivot
rightArray = [element for element in array[1:] if element >= pivot] # Larger than the pivot
```

Figure 2: Partition the array with Randomized Pivot Selection

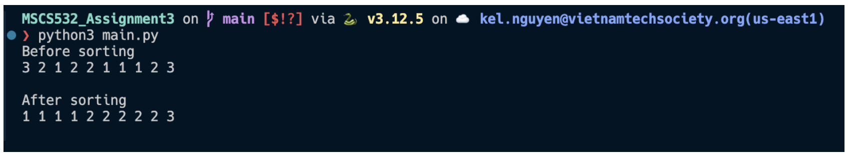
- **Step 3:** After partitioning the array, return the pivot index to perform the exact implementation for both sides of the partitioned array



```python
# Step 3: Repeat the same operation for left side and right side
return quickSort(leftArray) + [pivot] + quickSort(rightArray)
```

Figure 3: Quick Sort with left side and right side's partition

- As a result of Quick Sort with Randomized Pivot Selection



```
MSCS532_Assignment3 on  main [$!?] via 🐍 v3.12.5 on ☁ kel.nguyen@vietnamtechsociety.org(us-east1)
❯ python3 main.py
Before sorting
3 2 1 2 2 1 1 1 2 3

After sorting
1 1 1 1 2 2 2 2 2 3
```

Figure 4: Quick Sort with left side and right side's partition

b. **Theoretical analysis**

- At each step, we partition the array based on the pivot elements, and each partition takes O(n) time complexity at the best-case scenario and average scenario where the sub-array is roughly balanced at each level, then the height of the recursive will be half for each level. Therefore, Quick Sort would have the following time complexity of Ω(n log n) and Θ(n log n).

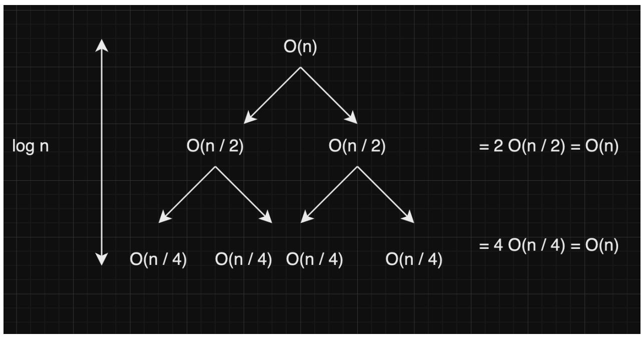


Figure 5: Recursion tree for Quick Sort best and average case complexity

However, if the array is already sorted and the pivot selection is poor, the sub-array won't be balanced, and the recursion tree will have n level, which leads to O(n^2)

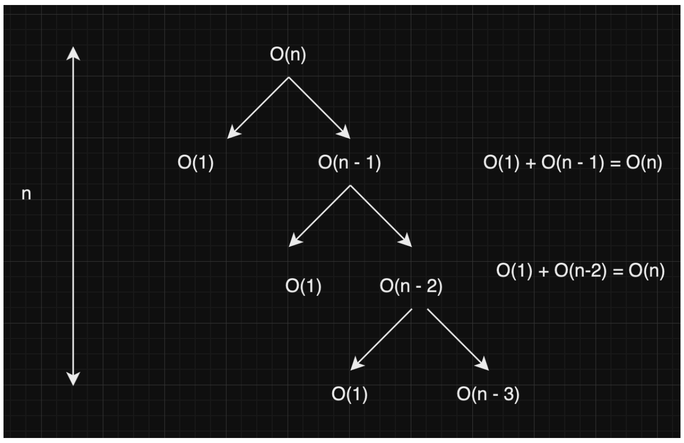


Figure 6: Recursion tree for Quick Sort's worst-case complexity

Since we are performing in-place sort for Quick Sort, it can achieve a space complexity of O(log n).

- The general recurrence relation is:

$$T(n) = T(k) + T(n - k - 1) + O(n)$$

  where k is the number of elements in the left partition.

- With the *Substitution Method[4],* assuming the k is roughly balanced which is k ≈ $\frac{n}{2}$, we will have the following equation when substituting k with the aforementioned recurrence relation:

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

  - However, we must prove that T(n) ≤ c n log n. Therefore,

$$T(n) \leq 2\,c\frac{n}{2}log\,\frac{n}{2} + cn$$

$$\leq cn\,log\,n$$

- With the Recursion Tree Method *[5],* assuming the k is roughly balanced, which is k $\approx \frac{n}{2}$, since the partition takes O(n) and each partition has a size of $\frac{n}{2}$, we will have O(n) at each level of recursive calls. Since the height of the recursion tree is determined by how many times the array can be divided in half, we will have the following equation:

$$Total\ cost\ =\ (O(n)\ +\ O(n)\ +\ ....\ +\ O(n))log\ n\ levels\ =\ n\ log\ n$$

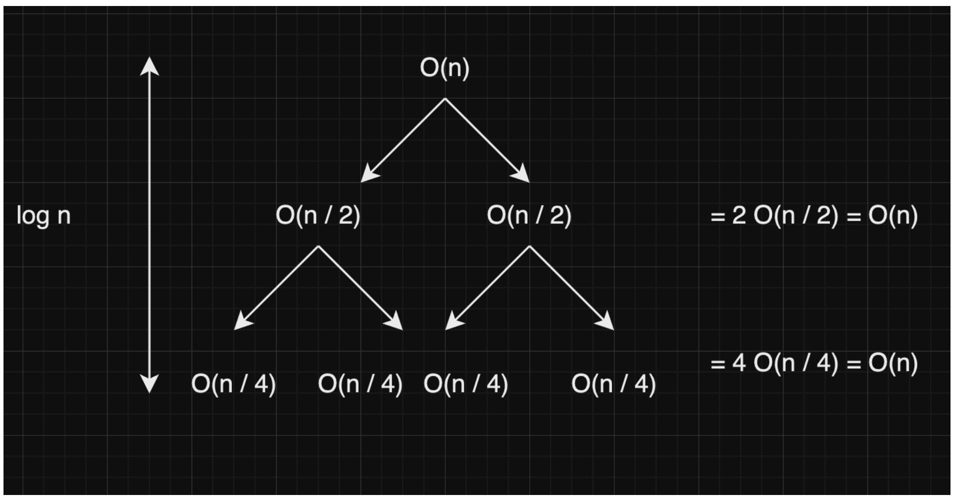


Figure 7: Recursion tree method for quick sort

- With *Master Theorem[6],* we would have the following equation:

$$T(n)\ =\ a\,T(\frac{n}{b})\ +\ O(n^{d})$$

where a is the number of recursive problems, $\frac{n}{b}$ is the size of each sub-problem, and O(n^{d}) is the cost outside recursive calls; however, for quick sort, d = 1 since partitioning only takes O(n). However, assuming the k is roughly balanced which is k $\approx \frac{n}{2}$, and each partition has a size of $\frac{n}{2}$, we can have the following equation for Quick Sort with Master Theorem

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

which falls down to the second case a where a = b^d (2 = 2) and p = 0 > - 1. Therefore,

$$T(n) = O(n^{\log_b a} \log^{p+1} n) = O(n^{\log_2 2} \log^{0+1} n) = O(n \log n)$$

c. Comparison

- When experimenting with Randomized Quick Sort and Deterministic Quick Sort with different datasets, the difference between them can be clearly found:
  - With **random data** from n = 100 to 100,000 elements

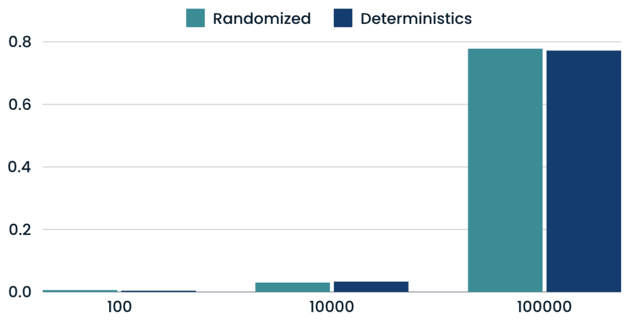


Figure 8: Random Dataset's Profiler with random data(CPU(s)))

  - With **sorted order data** from n = 100 to 100,000 elements

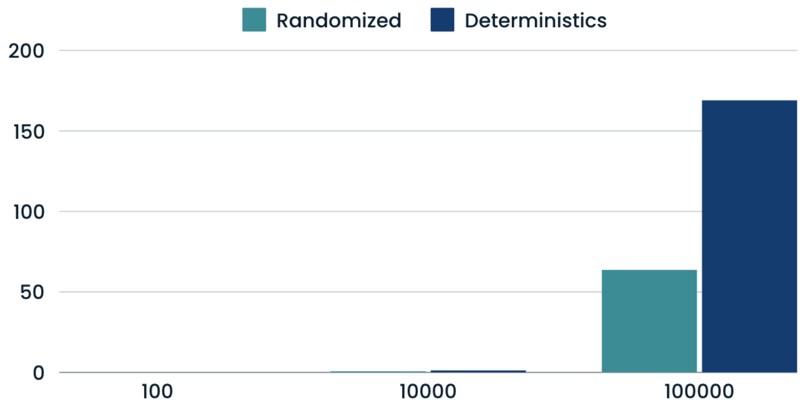


Figure 9: Random Dataset's Profiler with sorted order data (CPU(s)))

  - With **reverse sorted order data** from n = 100 to 100,000 elements

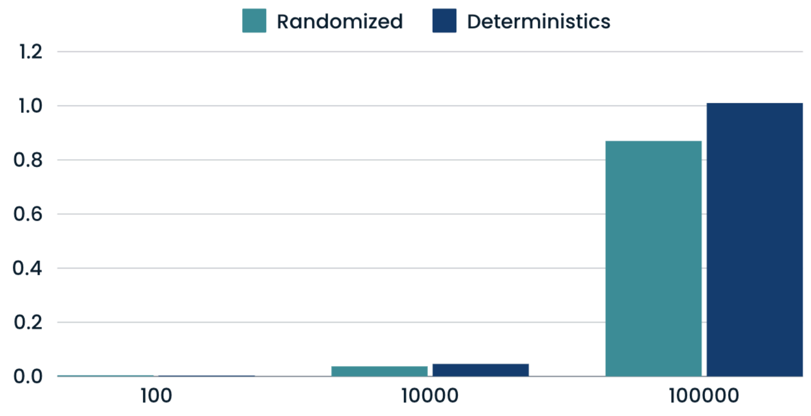


Figure 10: Random Dataset's Profiler with reverse sorted order data (CPU(s)))

- With **repeated elements** from n = 100 to 100,000 elements

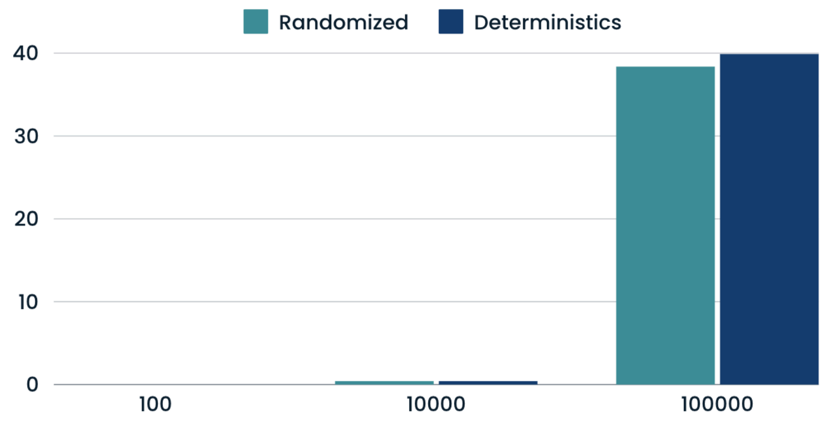


Figure 11: Random Dataset's Profiler with repeated elements(CPU(s)))

- As a result of the experiment, QuickSort with Randomized Pivot Selection has an increased time when sorting with sorted order data (from 0.08s to 64s for 100,000 elements). Even though the worst case is O(n^2), the randomness introduced by Randomized Pivot Selection has maintained the average complexity of Quick Sort to O(n log n). This can be confirmed when applying the same data to Deterministic Quick Sorth (from 0.08s to 167s). However, as we only increase the randomness in choosing a pivot, some edge cases, such as data with repeated elements, will behave the same with or without the pivot selection.

2. References

- [1] *Pavel Minaev* (2009). Is Quick Sort a potential security risk? Stackoverflow. https://stackoverflow.com/questions/1527136/is-quicksort-a-potential-security-risk
- [2] David Muser (2016). A Median-Of-Three-Killer Sequence. Programming Praxis. https://programmingpraxis.com/2016/11/08/a-median-of-three-killer-sequence/
- [3] *Said Sryheni Grzegorz Piwowarek (2024). Understanding the randomized Quick Sort. BaelDung. https://www.baeldung.com/cs/randomized-quicksort*
- [4] *GeeksForGeeks* (2024). How to analyze the complexity of the recurrence equation https://www.geeksforgeeks.org/how-to-analyse-complexity-of-recurrence-relation/
- [5] *Harshit Kumar* June 29, 2024 Recursion tree method
- [6] *GeeksForGeeks* August 15 ,2023 Advanced Master Theorem for divide and conquer recurrences
-