Quick Sort/Quick Select

Monday, 30 May, 2022 23:23

PASS NOTES

Quick Sort - Partitioning

Naïve Partitioning

Find pivot → Left temporary list to store all values <= pivot and right temporary list to store all values > pivot → Combine both temporary lists back to original list with pivot inside as well

Few things here:

- Creating temporary list → Auxiliary Space Complexity occurred → Partitioning not in place
- NOT stable, as everything <= pivot on left of list → multiple same values can have jumbled up order
- But can be stable with more memory (have 2 separate list for == pivot)
- Steps 2 & 3 add O(N) additional space besides the recursive stack

Lomuto's

- Problem, elements swapped multiple times → worse than Hoare's
- Unstable as well

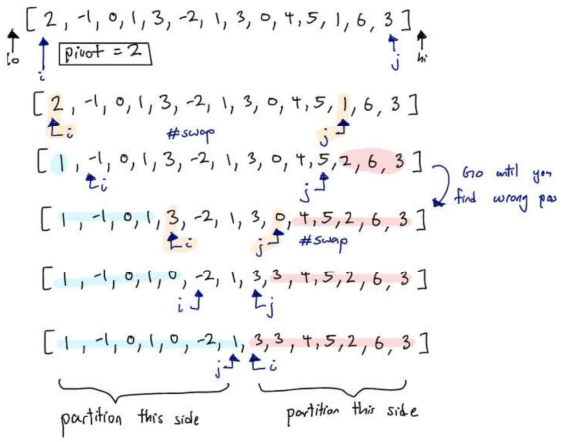
However, easier to understand and implement

Quick Sort - Partitioning

Hoare's

Find pivot → swap pivot to front → Left pointer at 2nd position and right pointer at last position → move left pointer until find value > pivot, move right pointer until find value < pivot, swap these elements, repeat until both pointers cross → swap pivot to right pointer position → repeat left and right of pivot

- There is a bug on step 4, what is it? (Hint: it happens when both pointers cross each other)
- No temporary list created → O(1) Auxiliary → Partitioning in place
- Lesser swapping and no copying → swapping done once only except pivot
- Still cannot be stable → can add condition on left and right pointer, but swapping right pointer with pivot could mess the stability up
- But as always, can be stable with more memory, but won't be in place (since more memory, Aux Space Complexity becomes > O(1))



Quick Sort - Partitioning

Dutch National Flag (DNF)

```
boundary1 = 1
j = 1
boudnary2 = n
while j <= boundary2:
    if array[j] is blue:
        swap(array[boundary1], array[j])
        boundary1 += 1
        j += 1
    elif array[j] is red:
        swap(array[j], array[boundary2])
        boundary2 -= 1
    else:
        j += 1
return boundary1, boundary2
```

- Idea is: The region whereby elements are == pivot, we don't need to sort it
- Minimized swaps and recursive sort (left and right partitions are smaller now since we aren't sorting those == pivot)
- In place
- In a nutshell, we swap elements to their respective region while having our boundaries shrink to the point where we reach the region of all elements being equals to pivot.

Quick Sort - Partitioning

Partitioning Algorithm	In Place?	Stable?
Naïve	No	No
Lomuto's	Yes	No
Hoare's	Yes	No
Dutch National Flag	Yes	Yes

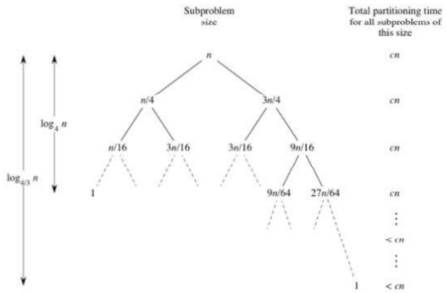
Some Key Takeaways:

- Most Partitioning can sacrifice memory for stability, but it would mess up the algorithm (usually causing the algo to not be in-place, which defeats the purposes of some of the partitioning algorithms.

Quick Sort – Time Complexity Analysis

Average Case Complexity

- Given a list, we divide to 3 sections, N/4, N/2 and N/4
- Probability to land on N/2 is approx. 50%
- Worst case in both N/4 areas are Pivot at position 1 or N-1
- Worst case in N/2 area is N/4 or 3N/4
- From recurrence knowledge, we know that T(N) → T(3N/4), reason why we don't take account to the smaller portion is because it does not go as deep
- Our base case is T(1) since we will reach size 1 eventually after multiple recursions
- And since we know that every recursion we would get 3N/4 of the previous list, we can safely assume that **N(3/4)^h = 1**
- By doing simultaneous maths, we get **log4/3 N**
- Maximum depth for average case is 2h, which gives us **2 log4/3 N**
- Therefore, average case height is O(log N), each partition is O(N), Total average cost is **O(N log N)**
- NOTE: regardless of our split ratio (1:99, 40:60, 30:70), we can still prove that the overall cost is **O(N log N)**
- The reason being that the value that is changing is the log base, whilst all other variables are constant. Hence, the overall average time complexity will never be effected even if the partitioning ratio is not perfectly even.



Quick Select

Fundamentals

- Idea → find k-th smallest elements in array
- Order is not important
- We can retrieve this with partitioning
- Never encouraged to sort and then slice, unnecessarily high complexity! (will give us O(NM log N))
- Cannot use counting sort, complexity will be very high (very large count array)
- Cannot use radix sort as well (number of "columns" can be very large)
- Slicing takes up O(k) → iterating k number of elements and either outputting them or put then in a new list
- For quick select, we do partition first, and then if index > what we want we repeat process on left sublist, vice versa for right, and if index == what we want, we can return the items.

Time Complexity

- Best case → O(N) where partition occurs once
- Worst case → O(N^2) if our pivot always fails

Complexity Analysis

- If using Quick Sort with Quick Select, it will end up being O(N^2) because both requires good pivots. (Note that Quick Select helps us find the median for Quick Sort)
- Because of this, we choose to find "good enough" median (a.k.a median of median)
- Find pivot within 50% area, or N/2, using quick select, then use quick sort
- However, max height is roughly log n, hence time complexity is still O(N log N). However, quick sort is O(1) since we aren't doing partitioning with it anymore after performing quick select
- General Idea: Perform Quick Select on list to partition values, perform recursive call on both sides of the list like Quick Sort format. Repeat until sorted list obtained. (Do everything in Quick Sort format, **except** for partitioning we do Quick Select)

Why are these important?

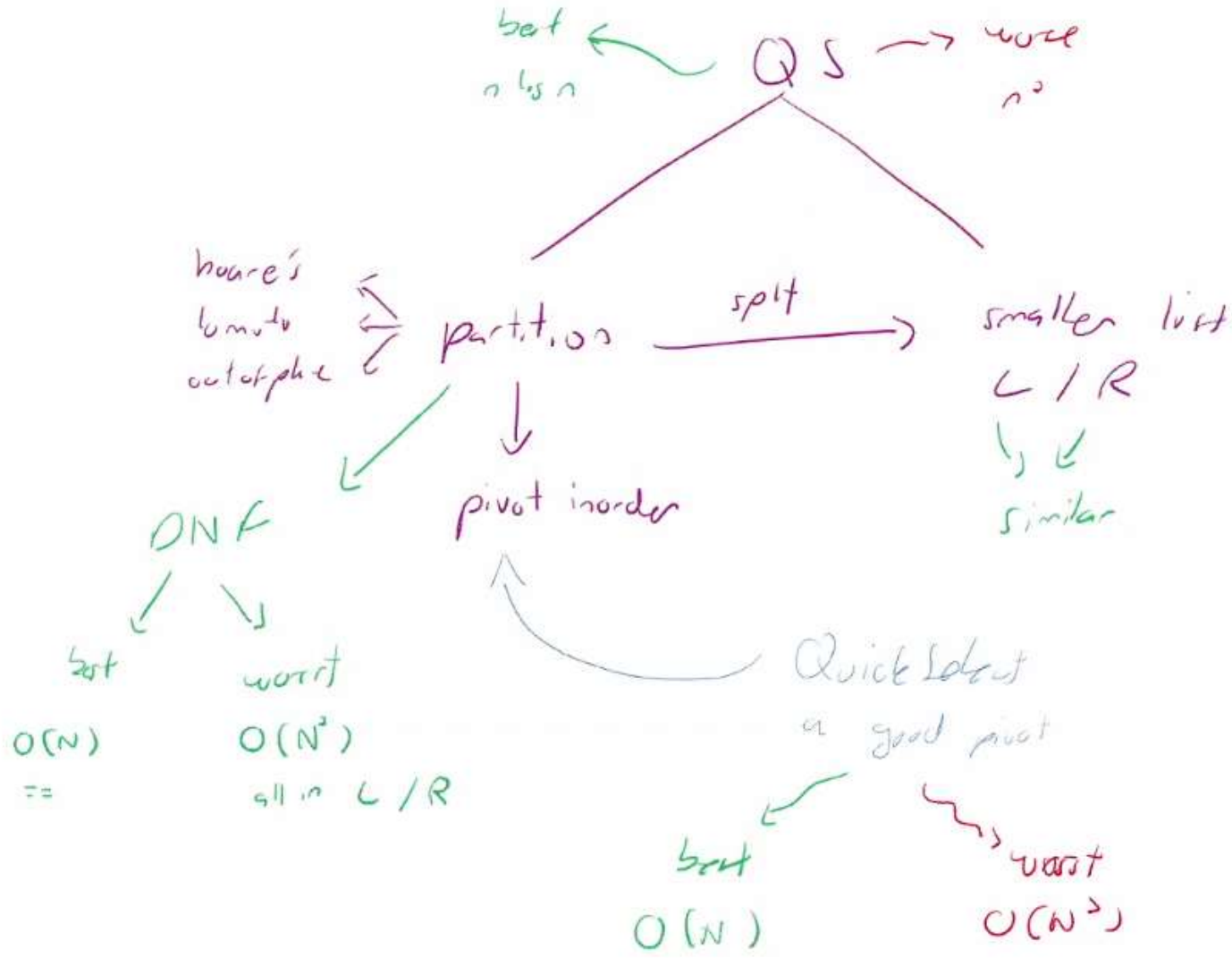
- **QuickSort** → Apparently is the fastest sorting algorithm, used for information searching (something like your Google Search). Used almost anywhere whenever stable sorting is not needed
- **QuickSelect** → Also known as the Hoare's selection algorithm (as this was developed by Tony Hoare). Most often used selection algorithm

What's Next?

- **Dijkstra**
- **Directed Acyclic Graph (if time permits)**

Sanity Check Week04

Tuesday, 31 May, 2022 16:18



Studio04 Q1

Problem 1. What are the worst-case time complexities of Quicksort assuming the following pivot choices:

(a) Select the first element of the sequence.

(b) Select the minimum element of the sequence.

(c) Select the median element of the sequence.

(d) Select an element that is greater than exactly 10% of the others.

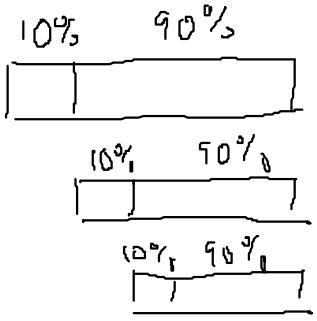
Describe a family of inputs that cause Quicksort to exhibit its worst case behaviour for each of these pivot choices.

a) $O(n^2)$, that is the complexity we use the first element of the sequence. As an example if we do apply quicksort to a sorted list. The first element of the sequence is the smallest. This shows that it is the worst possible case.

b) $O(n^2)$, that is the complexity we use the minimum element of the sequence. We know for a fact that if it is minimum or maximum it will be the worst case possible.

c) $O(n \log n)$ Selecting the median is the best choice. As it can sort nicely the left and the right of the list when its either bigger or smaller with recursion.

d) $O(n \log n)$ Even though selecting 10% will cause a split of 10/90. It can still provide and complexity of n log n since it will have either a log base of 10/9.



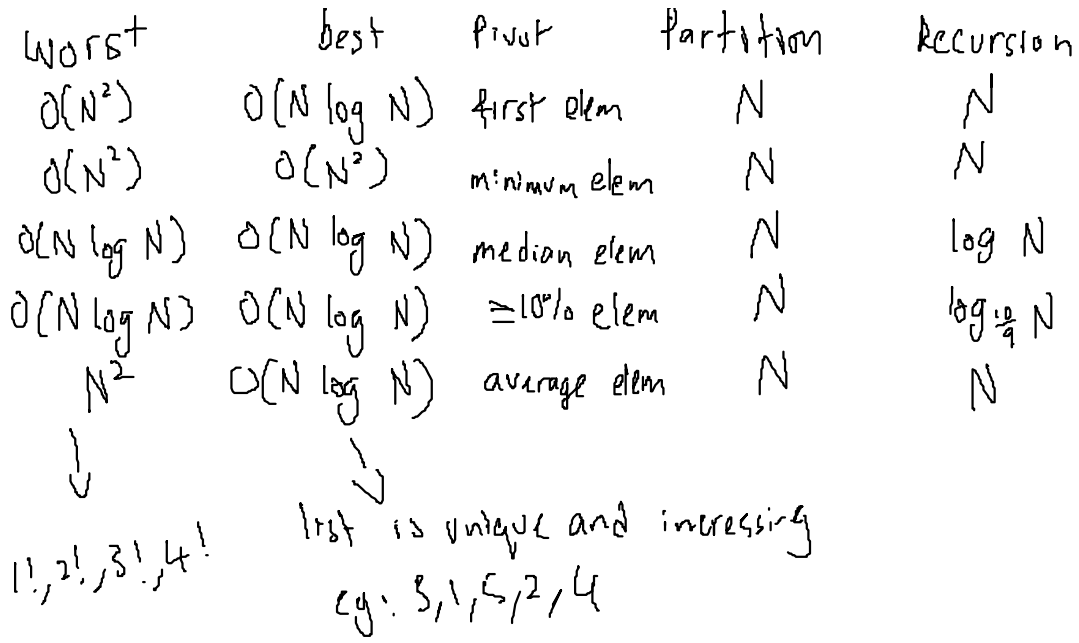
$0.90\,N \rightarrow 0.9^2\,N \rightarrow 0.9^3\,N \rightarrow \dots \rightarrow d$

when reach 1 element in list

$(0.9)^d N = 1$

$d = \log_{\frac{10}{9}} N$

Cause



Median worst case can be O(N^2) when all of the elements in the list are all the same, but using the dutch national flag method then it can be in O(n log n).

Studio04 Q4

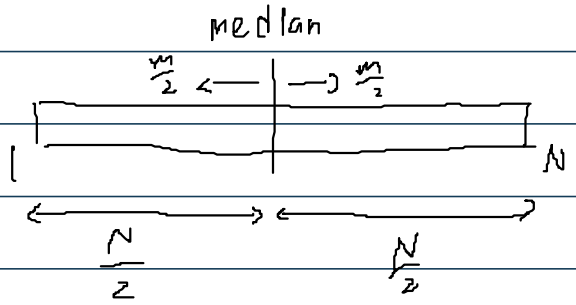
Saturday, 19 March, 2022 20:29

Problem 4. Devise an algorithm that given a sequence of n unique integers and some integer $1 \leq k \leq n$, finds the k closest numbers to the median of the sequence. Your algorithm should run in $O(n)$ time. You may assume that Quickselect runs in $O(n)$ time (achievable by using median of medians).

Use m instead of k

quickselect = O(N)

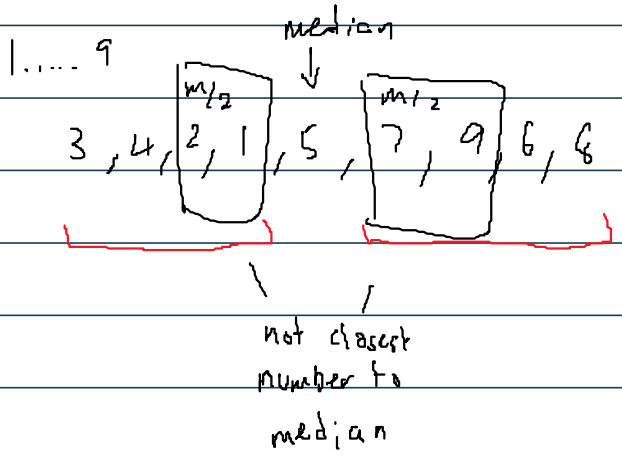
1)Get median
get values from left
and right of median



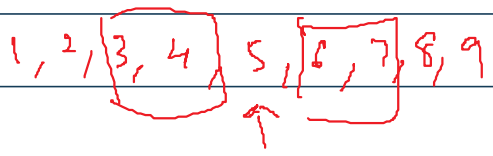
quickselect $k = \frac{N}{2} = O(N)$

m closest item to median

Example

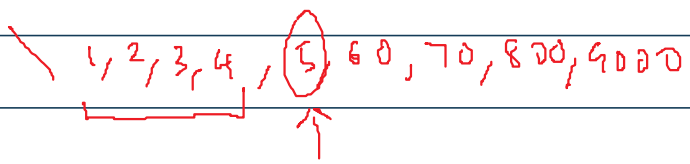


let's say we are lucky



4 closest

m = 4



Another way

median

3, 1, 4, 2, (5), 70, 800, 60, 9000

quickselect median = O(N)

go through list again : O(N)

but we calculate absolute difference

|3-5|, |1-5|, |4-5|, |2-5|, (5), |70-5|, |800-5|, |60-5|, |9000-5|

list_abs_diff 2 4 1 3 0 65 795 55 8995

1)perform quickselect O(n) to get median
2)loop through list O(n) calculate absolute difference
3)call quickselect again and instead of calling median not k=m,k=4

quickselect k=m+1

k=4+1=5

pos 1 2 3 4 5

| 2, 1, 3, 0, 4 | , 65, 795, 55, 895

5 smallest diff

from median inclusive of median

3, 4, 2, ~~5~~, 4 } 5 number closest to median inclusive of median

remove median

1)perform quickselect O(n) to get median = O(n)
2)loop through list O(n) calculate absolute difference = O(n)
3)call quickselect again and instead of
calling median not k=m+1,k=4+1 = O(n)

4)look at the m+1 item to the left of pivot = O(m)

Therefore, N + N + N + M = O(3N+M) = O(N+M)
Since M<N, O(N)

Past Year Q

Wednesday, 1 June, 2022 01:25

Consider a Quick Sort variant which uses 2 pivots to perform 4-way partitioning:

- Partition 1 for items that are of lesser value than pivot 1.
- Partition 2 for items that are of equal value with pivot 1.
- Partition 3 for items with values between pivot 1 and pivot 2.
- Partition 4 for items that are of greater or equal value than pivot 2

Quick sort is then called recursively on Partition 1, 3 and 4.

State the best and worst-case time complexity for this Quick Sort variant, if the partitioning can be performed in O(N) time. Briefly explain when they occur.

Best case complexity would be O(N) where all of the numbers are all the same item.

Worst case complexity would be O(N^2) where all of the numbers are either bigger than both pivot or smaller or in between both pivot. Where all the numbers are in partition 1,3,4.

Quick Sort and Quick Select

Question 8

3 Marks

Consider a list of lap times for a stage in the game of Fall Guys for Twitch Rivals, represented as a list of ***N* unsorted** positive floats with values from *0.0* seconds to *100.00* seconds.

To pass this stage, participants would need to clock in a lap time no more than *20.0* seconds.

As the game master however, you would want **at least 30%** of the players to move on to the next stage; with the following rules:

- If you are **within the top-30% fastest time** and **within the *20.0* seconds** requirement, you would move onto the next stage without any issues.
- If you are **within the top-30% fastest time** but is **above the *20.0* seconds** requirement, then you would be penalized for the next round. The penalty is how much slower are you compared to the **median of the top-30%**; if the time is below the median then there is no penalty.

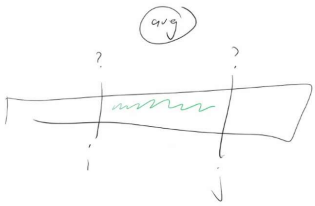
Describe an efficient algorithm using quick select to determine the sum of all penalties; in O(N) time complexity with the assumption that you have access to a quickselect algorithm which runs in O(N) time.

1. Quickselect for k=0.3N
2. Quickselect for k=0.5 of the smaller partition that is performed earlier, get the median value
3. Loop through items from (1) and sum penalty if time is more than 20sec

Consider a list of N positive integers.

Describe a linear time algorithm (using high level plain language) to find the item in a list whose value is closest to the average of the i-th and j-th largest elements in a list.

You may assume that you have a Quick Select function that operates in linear time complexity.



Multiple solutions to this
one of the solution do link up with the tutorial, it is linear
but you can do better as well, still linear

Quick select from i and then quick select from j
Find the closest to the average of i-th and j-th largest elements in a list

by calculating all of the modulus difference of the numbers and the average.