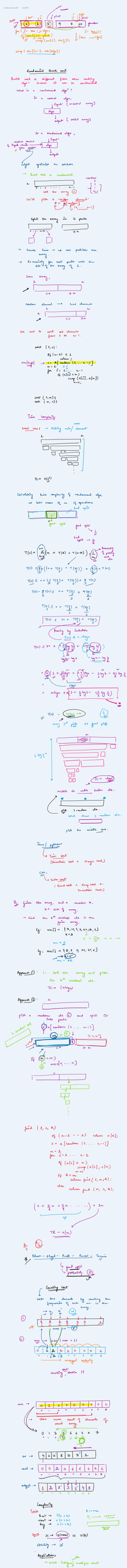
Sorting, 20 Aug 2021 - 12:00 PM
pivot
left
right
partition
pivot
left
right
partition
array = [10, 7, 9, 4, 8, 3]
left = [7, 9, 4, 8, 3]
right = [10]
pivot = 10
array = [7, 9, 4, 8, 3, 10]
left = [7, 4, 3]
right = [9, 8, 10]
pivot = 9
array = [7, 4, 3, 9, 8, 10]
Randomized Quick sort
Quick sort is different from other sorting algo's because it is randomized
What is a randomized algo?
In a normal algo,
input (normal array)
algo
output (sorted array)
In a randomized algo,
random source
input
algo
output
Input optimize on average
→ Quick sort is randomized.
random
array
sort the array
random
pick a random element
[7, 9, 4, 8, 3, 10]
Split the array in 2 parts
< x
> x
→ Issues. Issue → we see problem via array
→ Randomly pick pivot until we obtain the array of 1 i.e.
Issue array.
L
R
< x
> x
random element → bad element
L
R
< x
> x
We need to work all element from 0 to n-1.
sort [l, r]
if (l==r) ≤ 1
return ;
else if (array[l] == array[r])
m = l
for i = l, l+1, ... r-1
if (a[i] ≤ m)
swap (a[i], a[l])
l++;
sort [l, m-1]
sort [m, r]
Time Complexity
Worst case → picking min/max element
max
L
R
T(n) = O(n²)
Calculating time complexity of randomized algo.
we take mean of n operations
bad split
good split
good split
bad split → 1/2
T(n) = T(1/2) + T(1/2) + T(n-1)
Randomly 1/2 probability
T(n) ≤ T(1/2) + T(1/2) + T(n)
T(n) ≤ 1/2 (T(1/2)) + T(1/2)) + 1/2 T(n)
T(n) - 1/2 T(n) ≤ n + T(1/2) + T(1/2)
T(n)/2 ≤ n + T(1/2) + T(1/2)
T(n) ≤ 2n + T(1/2) + T(1/2)
Finally by substitution
T(n/2) ≤ 1. Stop.
T(n) ≤ 2n + 2(2n/2 + T(n/4) + T(n/4))
2n + 2n/2 + 2n/4 + 2n/8 + ... 2n/2^log n
2n(1 + 1/2 + 1/4 + 1/8 + ... 1/2^log n)
2n(2 - 1/2^log n)
2n(2 - 1/n)
= 4n - 2
T(n) = O(n)
every 2^i step → good step
2^log n
2 log n
middle is with better idea.
TC = O(log n)
random idea
sort their random idea.
pick the middle one.
Time | optimal
→ Quick sort
(Insertion sort + merge sort)
Cost
→ Quick sort
(Quick sort + merge sort + Insertion sort)
Given the array, and a number k,
k ≤ size of array.
→ find the kth smallest idea in the given array.
eg array[] = {7, 10, 4, 3, 20, 15}, k = 3
3 < 6 (n = 6)
→ 7
eg. array[] = {7, 10, 4, 3, 20, 15}
k = 4
→ 10
Approach ① 1. Sort the array and pick the kth smallest idea.
TC = O(n log n)
Approach ②
pick a random idea ② and split it.
> 7 smallest idea
pick random {7, 10, 4, 3, 20, 15}
> 7 → 7
if (k == n)
return a[n-1]
if (k < n)
return a[k-1]
if (k > n)
return find (k, m, n-k);
else
return find (m, s, k);
k = 3n/4 + 3n/4 v + ... = 3n
TC = O(n)
Approach ③
Blum - Floyd - Pratt - Rivest - Tarjan
good split probability 3/4
Counting sort
Inside this through by counting the frequency of each array.
array
max
min
① 1 2 3 4 5 6 7 8 9 10
② max = 10 (max + 1)
0 1 2 3 4 5 6 7 8 9 10
→ array[i] = array[i] + array[i-1]
array[i] = array[i] + array[i-1]
array[i] = array[i] + array[i-1]