Romil V. Shah

20008692

Assignment 1

CS/CPE 600

Prof. Reza Peyrovian

Submission Date: 09/18/2022

REINFORCEMENT Questions

Q1. (No.7)

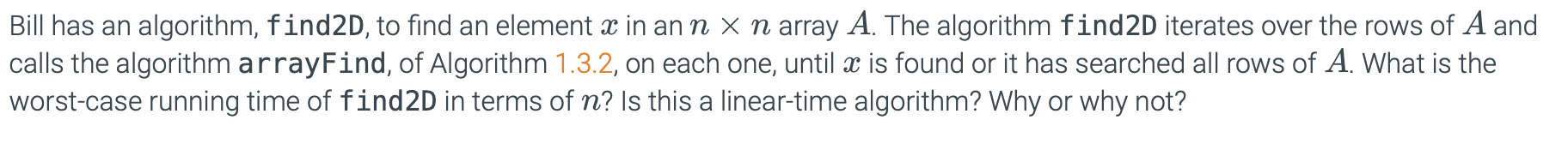
Graphical user interface, text, application

Description automatically generated

Ans. The order from lower to higher functions:

1. 1/n
2. 2100
3. log log n
4. sqrt(log n)
5. log2n
6. n0.01
7. sqrt(n), 3 n0.5
8. 2log n, 5 n
9. n log4n, 6 n log n
10. 2 n log2 n
11. 4 n3/2
12. 4log n
13. n2 log n
14. n3
15. 2n
16. 4n
17. 22n

Q2. (No. 9)



Ans.

The worst-case runtime complexity of FIND2D is O(n2), as it is a quadratic algorithm instead of a linear-time algorithm.

By examining the worst case where the element x is the very last item in the n × n array to be examined. In this case, find2D calls the algorithm arrayFind n times. arrayFind will then have to search all n elements for each call until the final where x is found.

Therefore, n comparisons are done for each arrayFind call. This means we have n × n operations - O(n2) running time.

Q3. (No. 22)

A picture containing logo

Description automatically generated

Ans.

We say that n is o(n log n) if for any constant c > 0 there is any constant n0 >= 0, such that n < c \* n log n for n >= n0.

So, 1/c < log n, we choose n0 = 21/c + 1 (when log is the base of 2).

Q4. (No. 23)

Text

Description automatically generated

Ans.

To show that n2 is w(n), let c > 0 be any constant,

there is a constant n0 > 0 such that n2 > cn. So n > c.

We can choose n0 = c + 1.

Q5. (No. 24)

Logo

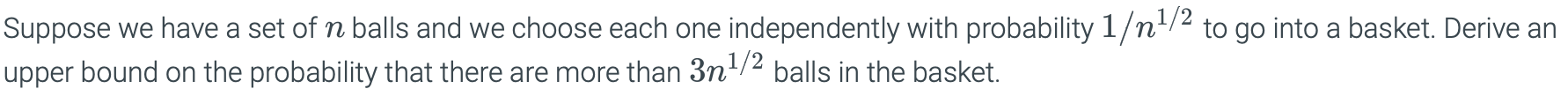
Description automatically generated with low confidence

Ans.

To prove the expression above, we need to find a constant c > 0 and constant n0 >= 1, such that n3 log n >= cn3.

We can choose c = 1 and n0 = 2(suppose log is the base of 2).

Q6. (No. 32)



Ans.

Based on Chernoff Bounds,

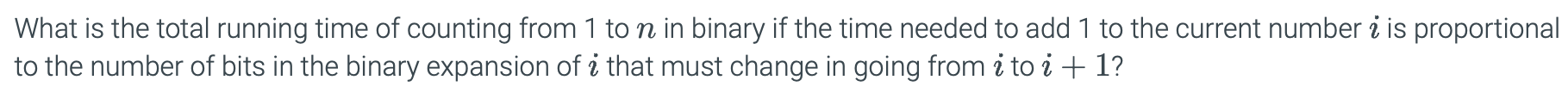
µ = E(X) = n \* (1/n1/2)= n1/2.

Then for δ =2, the upper bound is

Pr[X>(1+ δ)u] < (eδ/(1+δ)(1+δ))u => Pr(X > 3µ) <

CREATIVITY Questions

Q1. (No. 36)



Ans.

Let t be the time to change every single bit and let k be the total bits.

The total work is:

t \* (n/20 + n/21 + n/22… + n/2k) < t \* n \* 2 => O(n)

So, the total running time is O(n).

Q2. (No. 39)

Text

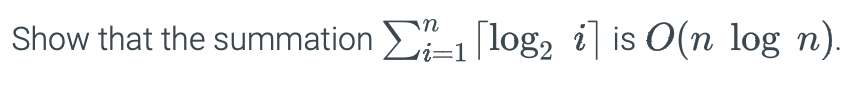
Description automatically generated

Ans.

For T(n):

T(n) = 2 \* T(n-1) = 2 \* 2 \* T(n-2) = … = 2 \* 2n-1 \* T(0) = 2n

Q3. (No. 52)



Ans.

Here, we assume that the base to all log used is 2.

Upper Bound Summation(log i) = log(1) + log(2) + .... + log(n)

= log(n) + log(n) + ...... log(n)

= n \* log(n)

Lower Bound Summation (log i) = log (1) + ... + log(n/2) + ... + log(n)

= log(n/2) + ... + log (n)

= log(n/2) + ... + log(n/2)

= n/2 \* log (n/2)

Hence, we can say that the summation is O(n log(n)).

Q4. (No. 62)

Graphical user interface, text

Description automatically generated

Ans.

The size of the array is expanded from N to N + ⌈N1/2⌉

Based on the amortization, each insertion will cost (N+ N1/2)/ N1/2 = 1+ N1/2 cyber dollars ($).

So, total insertion cost is as follow:

∑ 1+ 1+SQRT(N) = ∑ 2+SQRT(N)= 2n + ∑ SQRT(N) from N=1 to N=n

that we can get it is no more than:

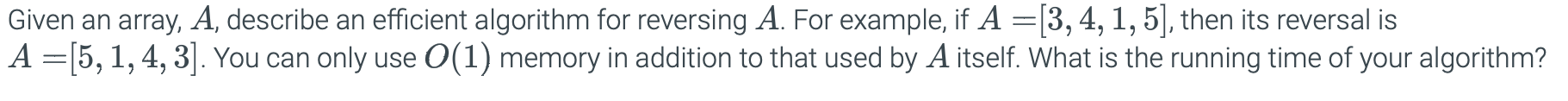
(2/3)n3/2+ (1/2)n1/2- 1/6 but no less than

(2/3)n3/2+ (1/2)n1/2+ 1/3 - (1/2)21/2.

So, the total cost of the array operation is θ(n3/2).

APPLICATION Questions

Q1. (No. 70)



Ans.

Algorithm reversal (start, end, n, A)

Step1: Initialize

n – number of elements in an array

start 0

end n-1

Step2: In a loop,

swap (arr[start], arr[end])

Step3: Start start + 1

End end -1

The Time Complexity will be O(n)

In the above algorithm,

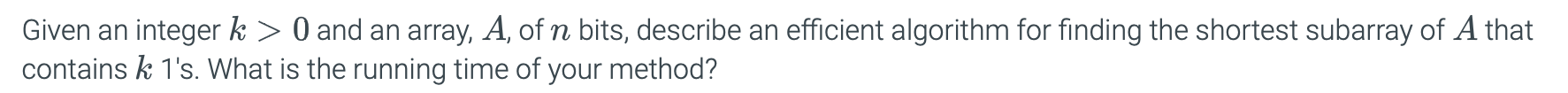
Step 1 will execute in constant time (1) and Step 2 will take n time, swapping will take constant time and the Step 3 again will take the constant time (1).

So, the total running time of the algorithm is O(n).

The running time of this algorithm would be O(n) because it accesses every element in the array the one time.

And no extra space needed, it only needs two variables to record the pointers. So, the space complexity is O(1).

Q2. (No. 77)



Ans.

Input: An array A of n-bits, indexed from 1 to n.

Output: The shortest subarray of A that contains k 1’s.

Count 0

k 0 //maximum found so far

for i 1 to n do

if A[i] = 0 then

count 0

else

count count + 1

k max(k, count)

return k

Run Time:

Count 0 (1 time)

k 0 (1 time)

for i 1 to n do (n times)

if A[i] = 0 then (1 time)

count 0 (1 time)

else (1 time)

count count + 1 (1 time)

k max(k, count) (1 time)

return k (1 time)

Total Run Time = 1 + 1 + n + 1 + 1 + 1 + 1 + 1 + 1

= n + 8

= O(n)

For loop will take n times to run whereas if and else statement will run in constant O(1) time. All other are variables which will take constant time O(1) to run.