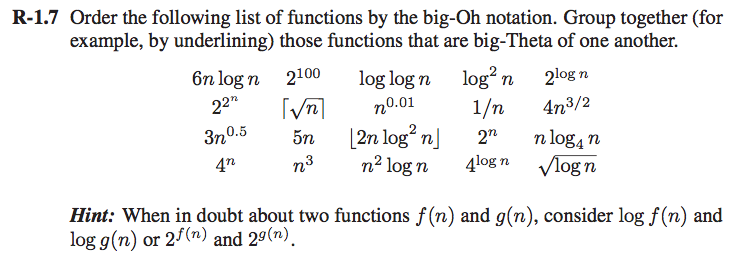
CS600 Homework 1

Atiq Patel CWID:10432883



**Solution:** The following are the list of functions in ascending order by the Big-Oh notation

1/n

2100

log2 n

n0.01

=

2logn = 5n

nlog4 n = 6nlogn

4n3/2

4logn

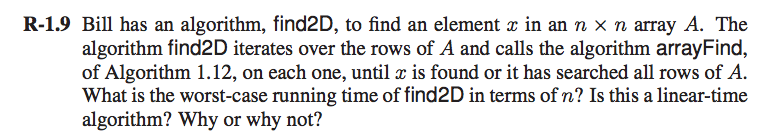
n2 log n

n3

2n

4n

22^n



**Solution:**

**Algorithm Find2D(x, A):**

**Input: An Array A of size n x n and a search element x to be searched in an array.**

**Output: The Index i & j such that x = A[i][j] where element is found. Return -1 if no element is equal to n.**

for i 0 to n – 1 do

for j 0 to n – 1 do

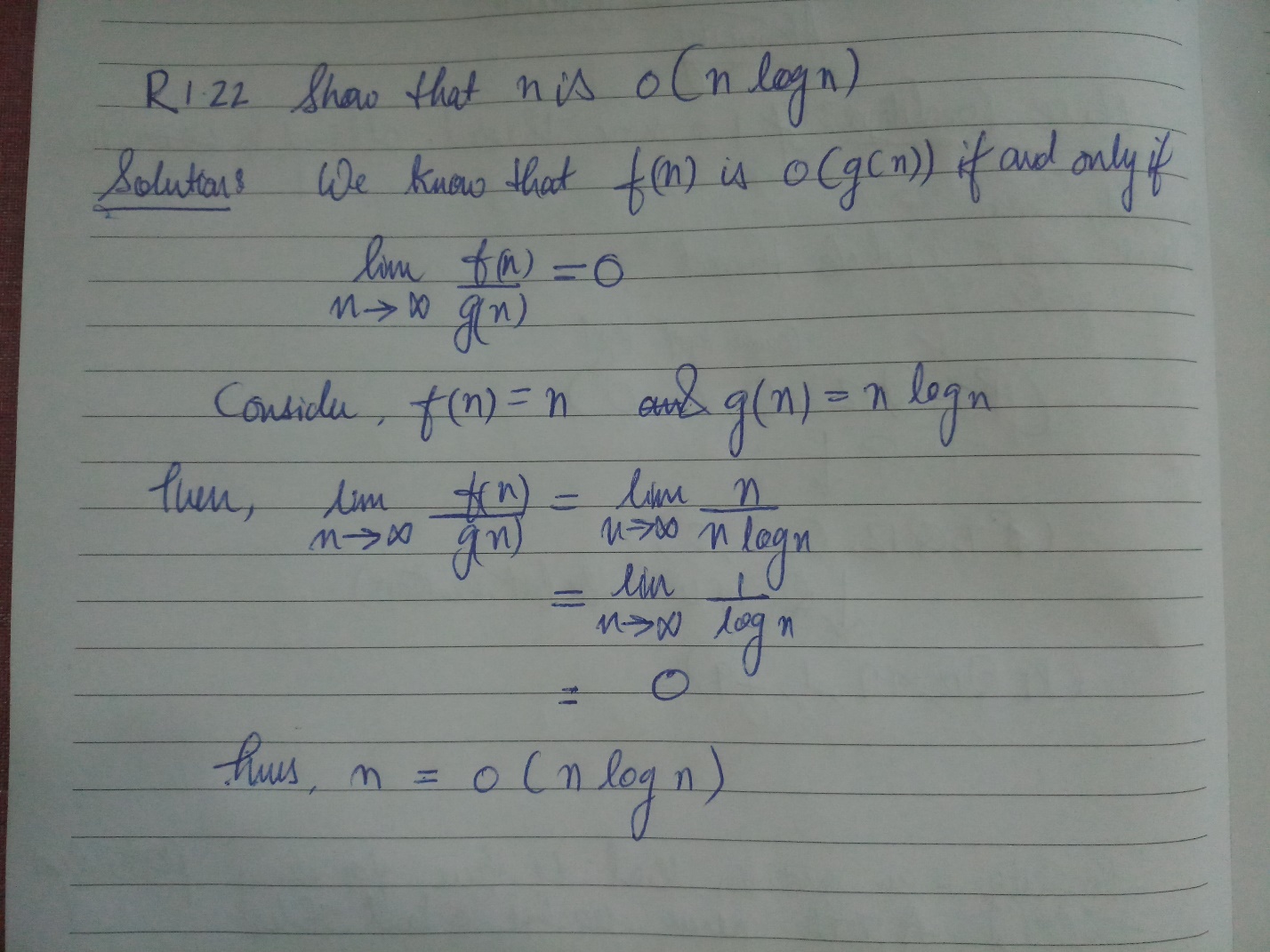
if x == A[i][j]

return i,j

return -1

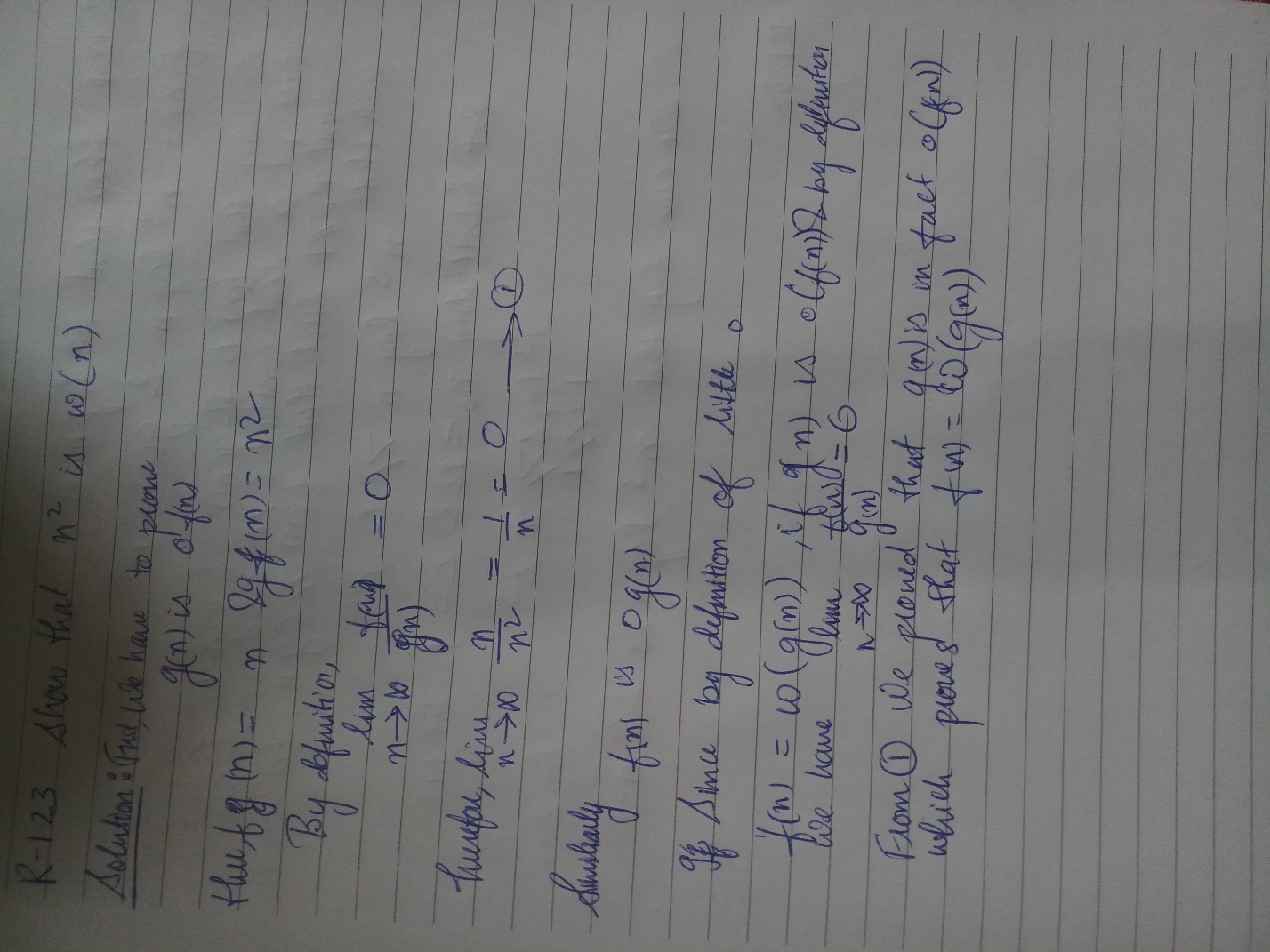
The worst case scenario for Find2D algorithm searches a dimensional array of size n x n. It is evident from the algorithm that there is nested for loop which runs for each “i”, it will run “j” times. Therefore the case would be (n x n)= n². We can observe that the algorithm runs in **O(n²)** which is not linear. It is quadratic time algorithm. That is because unlike from the case of the algorithm ArrayFind which runs from “i” to n because it runs n times, its run time is O(n). But in case of Find2D it has to traverse for each rows in the array which in turn makes n operations for the row and another n for traverse the column. Thus, it runs in quadratic time rather than linear time.

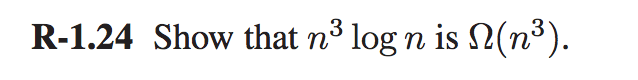
/Users/divyendrapatil/Desktop/Screen Shot 2017-09-01 at 11.34.44 AM.png

**Solution:**

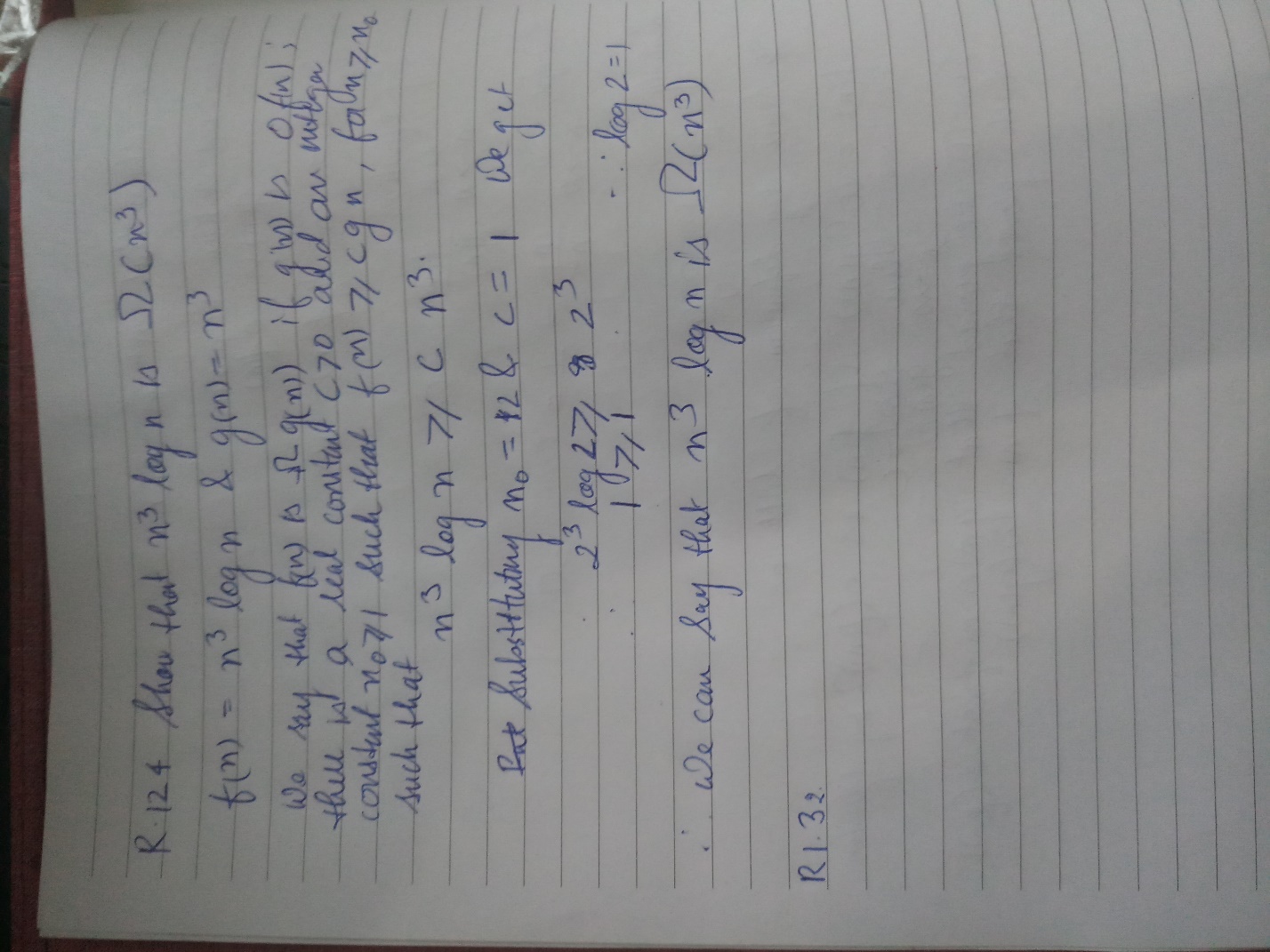
Screen%20Shot%202017-09-01%20at%2012.09.42%20PM.png

**Solution:**



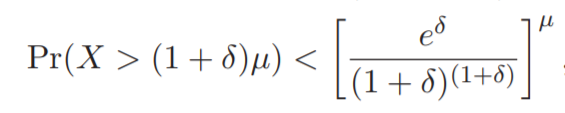


**Solution:**



**R-1.32** Suppose we have a set of n balls and we choose each one independently with probability 1/(n^1/2)to go into a basket. Derive an upper bound on the probability that there are more than 3n1/2 /balls in the basket.

**Solution:**

Based on Chernoff’s Bounds, 

Here µ= E (x)= x \*P(X)=n \* (1/) =

Substituting the value of µ= & *δ* = 2 we get.

Pr(

**C-1.4** What is the total running time of counting from 1 to n in binary if the time needed to add 1 to the current number i is proportional to the number of bits in the binary expansion of i that must change in going from i to i + 1?

**Solution:** Binary representation of 1 = 01

Binary representation of 2 = 10

Binary representation of 3 = 11

Binary representation of 4 = 100

We can notice that after adding 1, the number of bits get changed.

The first bit gets changed n times, the second one n/2 times, the third n/4 and so on….

Let T be the total running time and k be a constant then the series would be

**C-1.7** Consider the following recurrence equation, defining a function

T(n): T(n) = 1, if n = 0

T(n) = 2T (n − 1) otherwise, Show, by induction, that T(n) = 2n.

**Solution**: if n=0 T(n)=1 🡪 Equation 1

T(n) = 2T (n − 1) 🡪Equation 2

Substituting the value of n=1 in equation 2, we get

T(1)=2T(1-1)=2T(0)=2 x 1=2 ….. from equation 1

Substituting the value of n=2 in equation 2, we get

T(2)=2T(2-1)=2T(1)=2x2=4=2² ……. Since T(1)=2

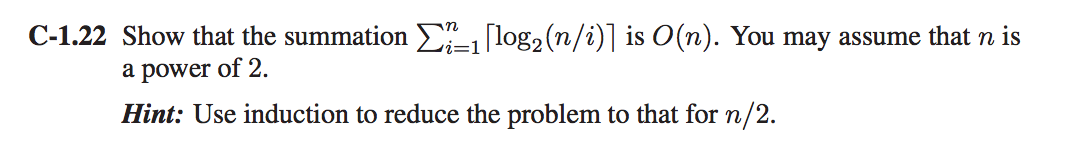
Similarly,

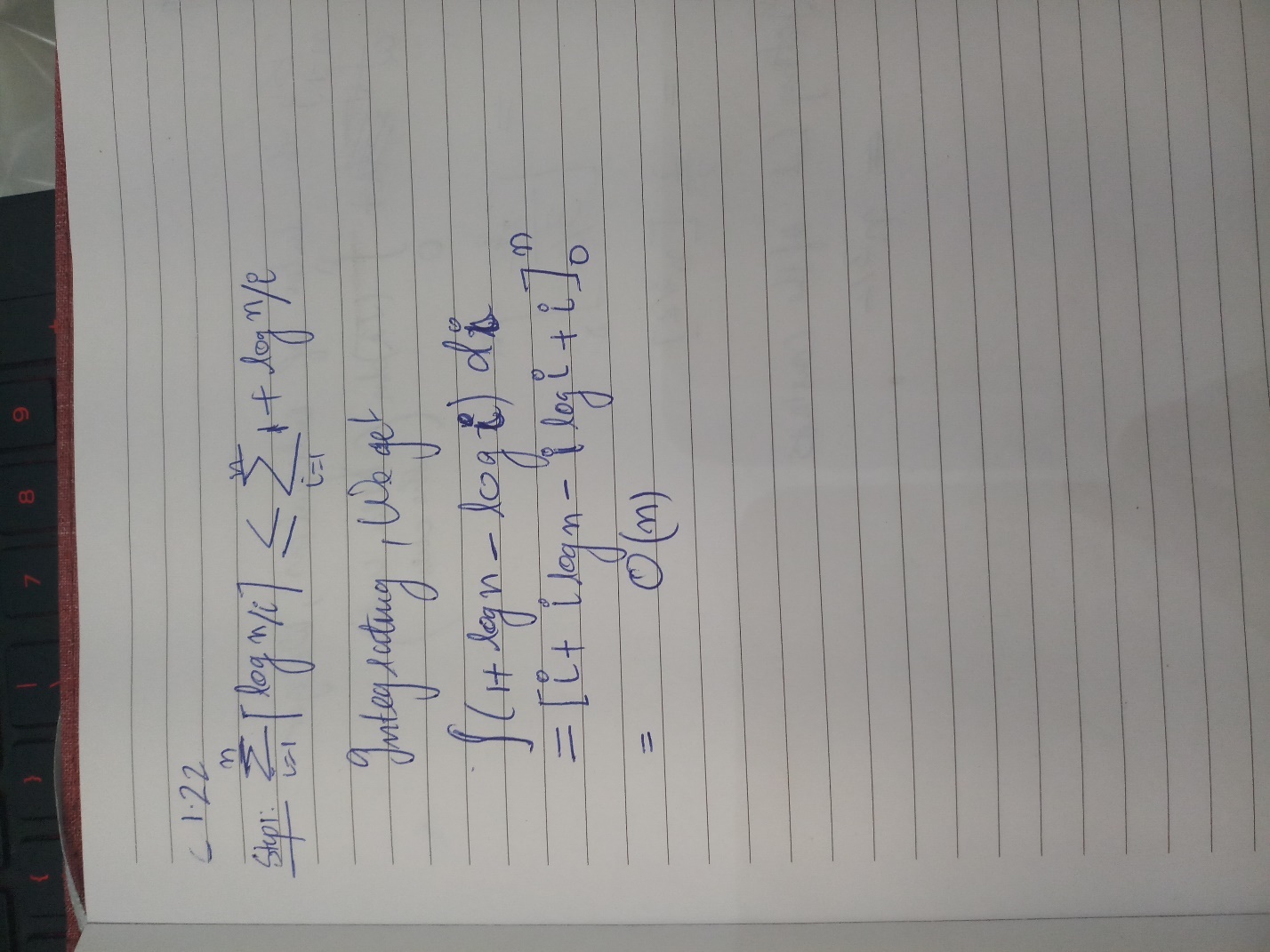
When n=3, T(3)=8=

n=4, T(4)=16= …. And so on

We can see that the values are clearly in a G.P. Therefore the nth term would be **.**

Hence, Proved.

**Solution**:



**C-1.30** Consider an implementation of the extendable table, but instead of copying the elements of the table into an array of double the size (that is, from N to 2N) when its capacity is reached, we copy the elements into an array with additional cells, going from capacity N to N + . Show that performing a sequence of n add operations (that is, insertions at the end) runs in Θ(n3/2) time in this case.

**Solution:**

Using the Amortization concept,  
When the capacity of the array has reach. The size of the array has to be increased from n to

Calculating the insertion values

Elements | Size increase | Overflow | A

1 | 🡪 = 2 | Yes | 1  
1 2 | 🡪 = 4 | Yes | 2

1 2 3 | | No | 4

1 2 3 4 | 🡪 = 6 | Yes | 1  
Here, A stands for number of new elements in the array.

We can see that the Base cost for amortization would be . Therefore **we charge cyber dollars** for each new element.  
We can also clearly see there is an Arithmetic progression in the form of

Integrating and substituting the values we get the bounds between and

From the above graph we can clearly conclude that performing a sequence of n add operations will run

Θ(n3/2).

A-1.8 Given an array, A, describe an efficient algorithm for reversing A. For example, If A=[3,4,1,5],then its reversal is A=[5,1,4,3]. You can only use O(1) memory in addition to that used by A itself. What is the running time of your algorithm?

The following algorithm uses a temporary variable “t” to switch the positions of two numbers through their index. We start from beginning of the array and switch it with the last index of the array and then increment the pointer x. It stops halfway which means the reversing is done. The run time for this algorithm is **O(n)** and also it uses **O(1)** memory.

Algorithm Reverse(X):

**Input:** A single dimensional array X.

**Output:**Reversed Array X.

x 🡨1

**while** (x!=n/2) **do**

t <-A[x]

A[x]=A[n-x-1]

A[n-x-1]=t

x=x+1

**A-1.15** Given an integer k > 0 and an array, A, of n bits, describe an efficient algorithm for finding the shortest subarray of A that contains k 1’s. What is the running time of your method?

**Solution:** We start the solution by creating two pointers, i and j pointer. “j” pointer points to the last element (n-1) of the array and “i” points to the second last element (n-2) in the array. We decrement I until the range is j-i there are k 1’s in it. If I reaches at the beginning of the array then return -1 or else record “j”-“i” as the current shortest length L. Keep on moving “i” lower if the element in the current index equals 1 and move j lower to a point where it points to 1. Try to match j-I and the length L and record the new length L1. If I reaches at the beginning of the array, return the length L1.

The running time of this method would be **O(n)**.

//if you didn’t understand algorithm please read the explanation.

Algorithm Maxk(A):

**Input:** A Single dimensional array A of size n.

**Output:** Shortest subarray of A that contains k 1’s.

i 🡨n

p 🡨0

j 🡨n-1

while (i!=0) do

if(i==1) do

l 🡨j-i

i 🡨i-1