Q-1 (2.2)

$n \longrightarrow$ +ve integer $\Rightarrow n \geq 1 ; n \in \mathbb{Z}^+$

$b \longrightarrow$ base

range given $[n, bn] \Rightarrow bn \geq n \Rightarrow b \geq 1$

- we have to prove $b^x \in [n, bn]$ for some real number x.

  we know that $\frac{n+bn}{2} \in [n, bn]$

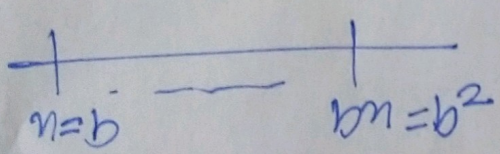
  lets $b^x = \frac{n+bn}{2}$

  $\Rightarrow x = \log_b \frac{n}{2}(b+1)$ Hence proved.

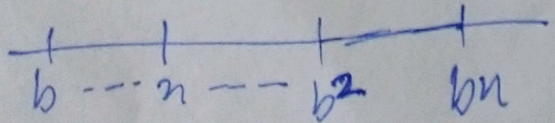
- And if we are asked for solution where x is integer. then consider three cases.

case $n = b$

real number line

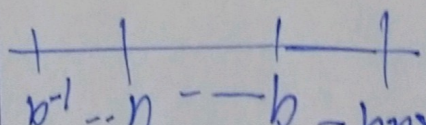


case $b < n < b^2$



case $b^1 < n < b$

real number line



$bn > b$

$n \geq 1$

Hence each case there is atleast one power of b. similarly we could prove for all $n = b^x$.

Q-1 (2.4)

algo-A $T(n) = 5T(n/2) + O(n)$

acc. master theorem $a=5, b=2, d=1$

$\log_2 5 > 2 > 1 \Rightarrow T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 5})$

algo-B $T(n) = 2T(n-1) + O(1)$

$T(n) = 2(2T(n-2) + O(1)) + O(1)$

$= 2^k T(n-k) + O(1) = 2^n T(1) + O(1)$

$\Rightarrow T(n) = O(2^n)$

algo-C $T(n) = 9T(n/3) + O(n^2)$

acc. master theorem $a=9, b=3, d=2$

$\log_3 9 = 2 = d \Rightarrow T(n) = O(n^d \log n)$

$= O(n^d \log n)$

$\Rightarrow T(n) = O(n^2 \log n)$

algo-A $T(n) = O(n^{\log_2 5})$

algo-B $T(n) = O(2^n)$

algo-C $T(n) = O(n^2 \log n)$ ⟶ slowest among A, B, C

we choose algorithm-C

Q-1 (20)

(b) $T(n) = 5T(n/4) + n$

$= 5(5T(n/16) + n/4) + n$

$= 5^k T(n/4^k) + \left(\left(\frac{5}{4}\right)^{k-1} + \cdots + 1\right) n$

assume $n = 4^x$

$\leq c_1 n^{\log_4 5} + c_2 n\left(\left(\frac{5}{4}\right)^k\right)$

$\leq O(n^{\log_4 5})$

(d) $T(n) = 9T(n/3) + n^2$

same as we have solved in algorithm C. above.

$= 9(9T(n/9) + (n/3)^2) + n^2$

$= 9^k T\left(\frac{n}{3^k}\right) + n^2(k)$

$\leq O(n^2 \log n)$

Q-2 (2.17)

A[1 .... n] ⟶ sorted array of distinct integers.

find(A[1,... n])

if n<1 : return "No solution"

else:

if n is multiple of 2 take one middle element @ A[n/2]

if n is not multiple of 2 take two middle elements $A\left[\frac{n-1}{2}\right]$ and $A\left[\frac{n+1}{2}\right]$

Now compare like

if $A\left[\frac{n}{2}\right] = n/2$ return $n/2$

if $A\left[\frac{n-1}{2}+\right] = \frac{n-1}{2}$ return $\frac{n-1}{2}$

if $A\left[\frac{n+1}{2}\right] = \frac{n+1}{2}$ return $\frac{n+1}{2}$

else:

return Find $A[1 \ldots n/2]$, Find $A\left[\frac{n+1}{2} \ldots n\right]$ accordingly.

Time complexity $T(n) = 2T(n/2) + O(1)$

according to master theorem

$a=2, b=2, d=0$

$\log_2 2 = 1 > 0 \geq d \Rightarrow T(n) = O(n^0 \log n)$

$= O(\log n)$

Q-2 (2022)

A[1...m], B[1...n] → sorted list → given
k → constant → given
O(log m + log n) → seems divide and conquer approach as binary search.

FindKth(A, B, k)
midA :→ mid element index of Array A.
midB = k - midA.
endA, endB :- index of last elements. respt.
firstA, firstB :- index of first elements respt.
if k == 1 return min(A[firstA], B[firstB])
else
if A[midA] > B[midB]
return find(A, B[midB ..... endB], midA)
else if A[midA] < B[midB]
return find(A[midA ..... endA], B, k - midA)
else if A[midA] == B[midB]
return A[midA]

Time complexity T(n) ≤ O(log(m+n))
≤ O(log m + log n)

Q-3 (2.19)

k sorted arrays each of size n

(a) merge in section 2.3 is linear ie $O(n)$
Hence to sort first two arrays

$T_1(n,n) = O(n+n) = O(2n)$

$T_2(2n,n) = O(2n) + O(n) = O(3n)$

$T_3(3n,n) = O(3n) + O(n) = O(4n)$

$\vdots$

$T_{k-1} = O\left(\frac{k-D(k)}{2} n\right) \leq O(nk^2)$

$\Rightarrow$ Time complexity is $O(nk^2)$

(b) algorithm

mergeConsecutivePairs (k : arrays)
// make pairs of two arrays each consecutive and merge them. $T(k,n) = O\left(\left(\frac{k}{2}\right)n\right)$

again same function while ($k != 1$)

$T(k/2, 2n) = O\left(\left(\frac{k}{4}\right)2n\right)$

$T\left(\frac{k}{4}, 4n\right) = O\left(\left(\frac{k}{8}\right)4n\right)$

$\vdots$

$T\left(\frac{k}{2^x}, 2^x n\right) = O\left(\left(\frac{k}{2^{x+1}}\right)2^x n\right)$

when $\frac{k}{2^x} = 1$

$\Rightarrow x = \log_2 k$

Time complexity $= O\left(\frac{k}{2k} 2kn\right) = O\left(\frac{kn}{2}\right)$

Q-4 (2.23)

(a)

```
1) ME(A[1...n])        // majority Element counter function.
   if n==1
      return ma = A[1], count = 1
   // ma → majority element is single element with frequency = 1.
   else:
2)    lets ME A[1...n/2] returns ma1, and count1.
      ME A[n/2+1,...n] returns ma2, and count2.
3)    if ma1 == ma2
         return ma1, count1+count2.
      else:
         if count1 > n/2 return ma1, count1
         if count2 > n/2 return ma2, count2.
         else : return "No majority element"
```

Time complexity part 1 will take some $O(1)$ part 2 and 3 will take some $T(n/2)$ and $O(1)$

$\Rightarrow$ overall $T(n) = 2T(n/2) + O(n)$

$\Rightarrow$ $T(n) = O(n \log n)$ using master Theorem.

Q-4 (2.23)

(b) Using Hint lets define some helperfunctions.

1. MergeConse(A[1....n])
   // takes array as input returns pair array of pairs with of two consecutive elements running time O(n), size of output arr n/2
2. Discard(A[1....n])
   // tekes array of pairs as argument and returns smaller array after discarding targated elements running time O(n)

~~If A[E(A[1...n])~~
~~If n==1~~

we need to do apply these two functions iteratively till $n_b = 1$

Hence similar to Q-3_2.19(b) above
time complexity will be linear with O(n)

If there is majority element with count > n/2
then after MergeConsecutive(A[1...n])
there will always be atleast one pair having both elements same and will not be discarded.
which will be left till end when
A.size() = 1. Hence proved.

Q-5 (2.25)

(a) fastmultiply(x, y) ⟶ n^a ⇒ $a = \log_2 3$

$10^n = 10\underbrace{0 \ldots 0}_{n\text{-times}}$

function pwr2bin(n)
  if n=1 : return 1010_2 } O(1)
  else :
    z = pwr2bin(n/2) } T(n/2)
    return fastmultiply(z,z) } $O(n^a)$

⇒ z = pwr2bin(n/2)

⇒ $T(n) = T(n/2) + n^a$

according to master theorem.

A = 1 B = 2 $a = d = \log_2 3$

⇒ $\log_2 3 = d > 0 \geq \log_2 1$

⇒ $T(n) = O(n^a)$

Q-5 (2.25)

(b) function dec2bin(x)
if $n=1$ return binary[x] } O(1)
else:
split x into x_L, x_R with $n/2$ digits
return fastmultiply(dec2bin(x_L), pwr2bin($n/2$))
+ dec2bin(x_R);

$T(n) \leq 2T(n/2) + n^a$

according to master theorem

$A=2, B=2 \quad a=d=\log_2 3$

$\Rightarrow \log_2 3 = d > 1 \geq \log_2 2$

$\Rightarrow T(n) = O(n^d) = O(n^{\log_2 3})$