NAME:- SHISHU
REG :- 2020CA089

Assignment-3

Question 1 solution

Base case:

when $n = 2$

$$T(2) = 2$$
$$= 2 \lg 2$$

i.e solution hold

Induction step :-

Let's assume that there exists a k, greater than 0, such that- $T(2^k) = 2^k \lg 2^k$. Now we proof that formula true for $k+1$ too.

i.e $T(2^{k+1}) = 2^{(k+1)} \lg 2^{(k+1)}$

by recurrence

$$T(2^{k+1}) = 2T(2^{k+1}/2) + 2^{k+1}$$
$$= 2T(2^k) + 2 \cdot 2^k$$
$$= 2 \cdot 2^k \lg 2^k + 2 \cdot 2^k$$
$$= 2 \cdot 2^k (\lg 2^k + 1)$$
$$= 2^{k+1} (\lg 2^k + \lg 2)$$
$$= 2^{k+1} \lg 2^{k+1}$$

proved

Ques 2 solution

for insertion sort to beat merge sort for input of size of n, $8n^2$ must be less than $64 n \lg n$.

$$8n^2 < 64 n \lg n$$
$$8 n \cdot n < 64 n \lg n$$
$$n < 8 \lg n$$
$$2^{n/8} < n \quad \text{— (1)}$$

Now we will check for diff n Now inequality ① varies —

$n = 8$, $2 < n$

$n = 16$, $4 < n$

$n = 32$, $16 < n$

$n = 64$, $256 > n$

Now

$n = 48$, $64 > n$

$n = 40$, $32 < n$

$n = 44$, $44.8 > n$

$n = 42$, $38.4 < n$

$n = 43$, $42.4 < n$

So at $n = 44$ merge sort ~~stands~~ bead to beat insertion sort again. therefore, for $2 \leq n \leq 43$ insertion sort beats merge sort.

Question & Solution

for A to run faster than B, $100n^2$ must be smaller than 2^n.

Calculation

Intuitively we can realize that A (quadratic time complexity) will run much faster than B (exponential time complexity) for very large values of n.

Let's start checking from $n = 1$ and go up for value of n which are power of 2 to see where that happen.

$n = 1 \Rightarrow 100 \times 1^2 = 100 > 2^n$

$n = 4 \Rightarrow 100 \times 4^2 = 1600 > 2^n$

$n = 16 \Rightarrow 100 \times 16^2 \Rightarrow 25600 < n^2$

Somewhere between 8 and 16, A starts to run faster than B, Let's do what we were doing but now we are going to try middle value of the range. repeatedly (binary search).

$n = 12 \Rightarrow 100 \times 12^2 = 14400 > 2^n$

$n = 14 \Rightarrow 100 \times 14^2 = 19600 > 2^n$

$n = 15 \Rightarrow 100 \times 15^2 = 22500 < 2^n$

So, at $n = 15$, A starts to run faster than B.

Question 5 solution

In a heap of height h:-

minimum no. of elements :- when the last level contain only one node.

$$2^0 + 2^1 + 2^2 + \cdots 2^{h-1} + 1 = 2^h$$

maximum no. of elements :- when the last level is full

$$2^0 + 2^1 + 2^2 + \cdots 2^h = 2^{h+1} - 1$$

Ques 6 solution

{ h = height
n = no. of element

We know that

no. of nodes $= 2^{(h-1)}$

$\lg(\text{no. of nodes}) = \log(2^{h-1})$

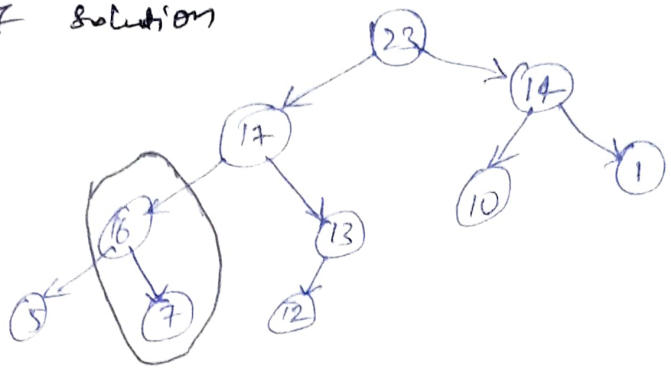
$\lg(n) = \lg(2^{h-1})$

$\lg(n) = (h-1)\lg 2$

$h = \lg n + 1/\lg 2$

$h = [\log n]$ { h = floor($\log n$)

Question 7 Solution



The array is not max heap. Since at above tree node 6 (parent) is less than the child 7.

Quest 11 Solution

If all the elements are same, the quick sort partitions return index $q = r$. This means the problem with size n is reduced with size $n-1$.

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

Here

$$T(n) = \Theta(n^2)$$

Belonging to $\rightarrow \Theta(n^2)$