Programs of insertion sort, and Merge sort with outputs

**********Insertion Sort**********

```
#include<iostream>
using namespace std;
int main()
{
   int temp, x, y, i, len;
   int a[100];
   cout<<" Enter the number  of elements in array ";
   cin>>len;
   cout<<" Enter the  elements of array ";
   for(i=0;i<=len;i++)
 {
cin>>a[i] ;
}
   for (x = 1; x < len; x++)     // logic for insertion sort begins here
 {
 temp = a[x];
  y = x – 1;
while (temp < a[y] && y >= 0)
   {
 a[y + 1] = a[y];
 y = y - 1;
        }
a[y + 1] = temp;
   }
 cout << "Sorted list of insertion sort ";
  for (i = 0; i<len; i++)
 cout << a[i] << " ";
```

}

```
 Enter the number  of elements in array 6
 Enter the  elements of array 8
3
7
2
6
9
2
Sorted list of insertion sort 2 3 6 7 8 9

...Program finished with exit code 0
Press ENTER to exit console.
```

**********Merge Sort Outputs**********

```
#include <iostream>
using namespace std;
int a[500];
void merge(int,int,int);
void merge_sort(int low, int high)
{
   int mid;
   if(low<high)
   {
      mid=(low+high)/2;
      merge_sort(low,mid);
      merge_sort(mid+1,high);
      merge(low,mid,high);
   }
}
void merge(int low,int mid,int high)
{
   int h,i,j,b[50],k;
   h=low;
```

```
i=low;
j=mid+1;
while((h<=mid)&&(j<=high))
{
   if(a[h]<=a[j])
   {
      b[i]=a[h];
      h++;
   }
   else
   {
      b[i]=a[j];
      j++;
   }
   i++;
}
if(h>mid)
{
   for(k=j;k<=high;k++)
   {
      b[i]=a[k];
      i++;
   }
}
else
{
   for(k=h;k<=mid;k++)
   {
      b[i]=a[k];
```

```
        i++;
      }
   }
   for(k=low;k<=high;k++)
   {
   a[k]=b[k];
   }
}
int main()
{
  int num,i;
  cout<<"Please enter the number of elements ";
  cin>>num;
  cout<<"enter elements :\n";
  for(i=1;i<=num;i++)
  {
     cin>>a[i];
  }
  merge_sort(1,num);
  cout<<"Sorted list of merge sort ";
  for(i=1;i<=num;i++)
  cout<<a[i]<<" ";
  return 0; }
```

```
Please enter the number of elements 5
enter elements :
7
5
3
8
1
Sorted list of merge sort 1 3 5 7 8

...Program finished with exit code 0
Press ENTER to exit console.
```

```
}
```

Program to find the input size n where merge sort starts to beat insertion sort in terms of the worst-case running time. You can use clock_t function (or other time function for higher precision) to obtain running time. You need to set your input such that it results in the worst-case running time. Report running time of each algorithm for each input size n

```
#include <iostream>

#include<ctime>

using namespace std;

int a[500], i;                    // array is defined here

void insertion_sort(int);

void merge(int, int, int);

void merge_sort(int low, int high)

{

   int mid;

   if (low < high)

   {

      mid = (low + high) / 2;

       merge_sort(low, mid);

      merge_sort(mid + 1, high);

       merge(low, mid, high);



   }

}

void merge(int low, int mid, int high)

{

 int h, i, j, b[50], k;

   h = low;

   i = low;
```

```
j = mid + 1;
  while ((h <= mid) && (j <= high))
  {
     if (a[h] <= a[j])
     {   b[i] = a[h];
         h++;
     }
     else
     {
         b[i] = a[j];
          j++;
     }
     i++;
  }
  if (h > mid)
  {
     for (k = j; k <= high; k++)
     {
         b[i] = a[k];
         i++;
     }
  }
  else
  {
for (k = h; k <= mid; k++)
```

```
      {
         b[i] = a[k];
          i++;
  }


   }
 for (k = low; k <= high; k++)
   {
      a[k] = b[k];
   }
}
void insertion_sort(int len)
{
 int temp, x, y;
for (x = 1; x < len; x++)
{
 temp = a[x];
 y = x - 1;
while (temp < a[y] && y >= 0)
{
a[y + 1] = a[y];
y = y - 1;
 }
 a[y + 1] = temp;
```

```
    }

    cout << "Sorted list of insertion sort  /n";

    for (i = 0; i <= len; i++)

        cout << a[i] << " ";

}

int main()
{
    int num; int temp1;
    clock_t time_req1;
    clock_t time_req2;
    cout<<" set your input for this program such that it results in the worst-case running time.  /n ";
    for (num = 3; num <= 100; num = num + 10)
    {
        for (i = 1; i <= num; i++)
        {
            a[i]=i;}
        for (int q = 1; q <= num; q++)
  {
             for (int r = q + 1; r <= num; r++)
           {
```

```
        if (a[q] < a[r])
        {   temp1 = a[q];
            a[q] = a[r];
           a[r] = temp1;
        }
     }
  }

  time_req1 = clock();

  merge_sort(1, num);

  time_req1 = clock() - time_req1;

  cout << "Sorted list of merge sort  \n";

  for (i = 0; i <= num; i++)

     cout << a[i] << " ";

  cout << " the time run of the merge sort " << (double)time_req1 / CLOCKS_PER_SEC << " seconds
\n";

  time_req2 = clock();

  insertion_sort(num);
```

```
        time_req2 = clock() - time_req2;


        cout << " the time run of the insertion sort " << (double)time_req2 / CLOCKS_PER_SEC << " seconds
\n";


        if (time_req2 > time_req1)


        {
           cout << "  nth element where merge hits insertion sort \n" << num<< "\n";
        }


    }
    return 0;
}
```

Output of the code is :

```
set your input for this program such that it results in the worst-case running time
Sorted list of merge sort
 0 1 2 3  the time run of the merge sort 2e-06 seconds
Sorted list of insertion sort
 0 1 2 3  the time run of the insertion sort 9e-06 seconds
  nth element where merge hits insertion sort  : 3
Sorted list of merge sort
 0 1 2 3 4 5 6 7 8 9 10 11 12 13  the time run of the merge sort 2e-06 seconds
Sorted list of insertion sort
 0 1 2 3 4 5 6 7 8 9 10 11 12 13  the time run of the insertion sort 8e-06 seconds
```

nth element where merge hits insertion sort 13

Sorted list of merge sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 the time run of the merge sort 2e-06 seconds

Sorted list of insertion sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 the time run of the insertion sort 1e-05 seconds

nth element where merge hits insertion sort 23

Sorted list of merge sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 the time run of the merge sort 4e-06 seconds

Sorted list of insertion sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 the time run of the insertion sort 2e-05 seconds

nth element where merge hits insertion sort 33

Sorted list of merge sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 the time run of the merge sort 5e-06 seconds

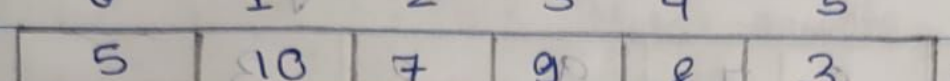
Sorted list of insertion sort

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 the time run of the insertion sort 1.9e-05 seconds

(Q.2) Insertion Sort for 10, 5, 7, 9, 8, 3. Iterations each step.

Original array.

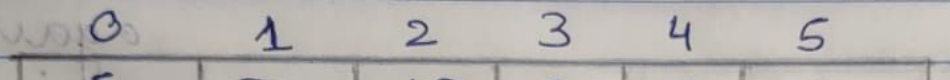
After 1st iteration output



compare 1st and 2nd element before it sorts

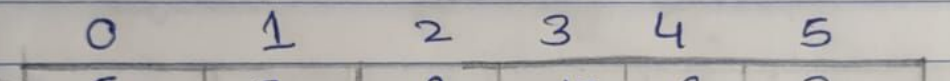
compared 3rd element to all before it sorts

2nd



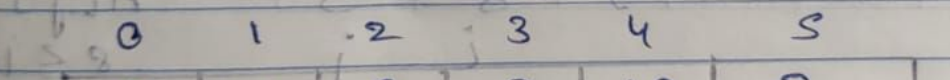
compare 4th element to all before it sorts

3rd



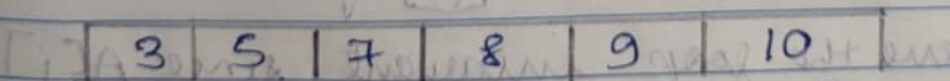
compare 5th element to all befor e it sorts

4th



compare last element to all before it sorts.

5th



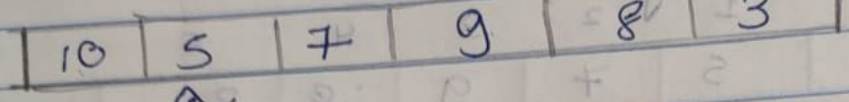
Hence above it sorted insertion sort array

↑ = denotes where

Partition Sub Routine:

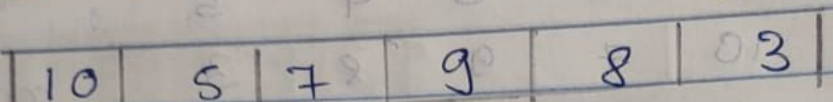
A[p] = 10 pivot → x, Hence A[i] = 10

Loop j ← p+1 to r

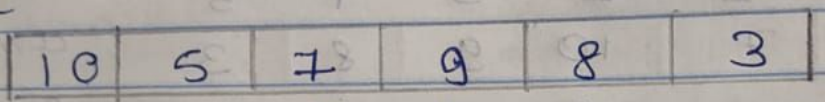


as A[j] ≤ x i.e 5 ≤ 10 move boundary i.

1st iteration

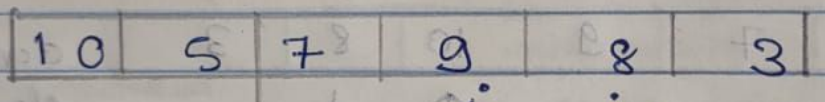


2 iteration



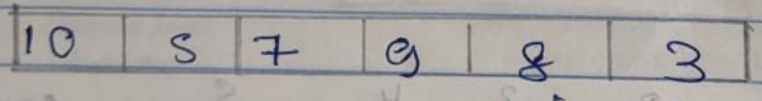
again A[j] ≤ x
7 ≤ x
7 ≤ 10
move boundary i.

3rd iteration



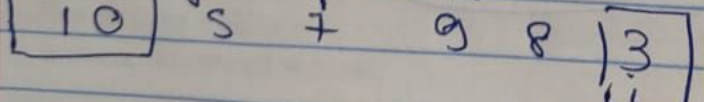
A[j] ≤ x
9 ≤ 10.
move boundary i

4th iteration

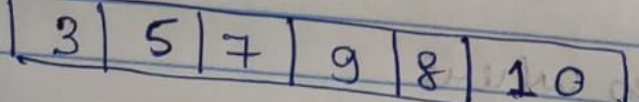


A[j] ≤ x
8 ≤ 10
move the boundary i.

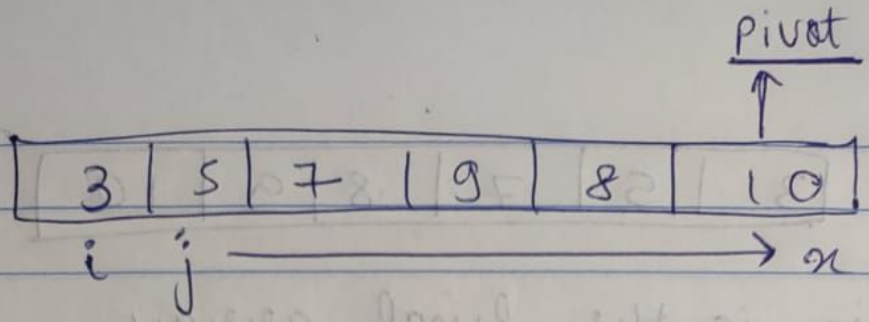
5th iteration the j loop ends.



To preserve the loop invariant since A[j] ≤ x we move the boundary i & now can exchange the value of A[i] and A[j]. Hence answer is

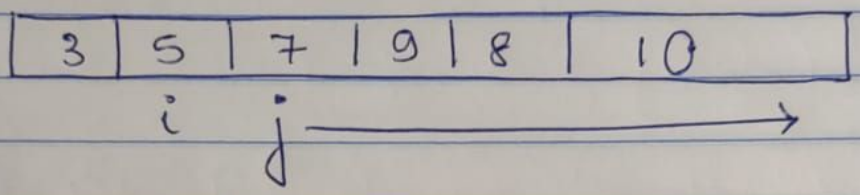


6th iteration



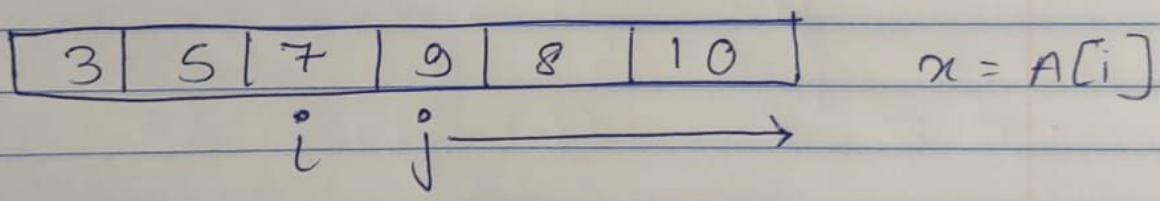
Now taking $A[j] > x$ we add to j by 1 hence loop doesn't change as 3 is smallest. then shift i.

7th iteration



$A[j] > x$ we add to j by 1 hence loop doesn't change as 5 is smallest. as compared 7, 9, 8, 10.

8th iteration.

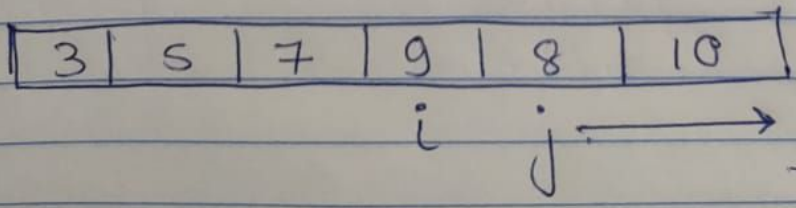


$A[j] > x$ we add to j by 1 hence the set of value doesn't change as.

7 is less than 9 8 10

9th iteration

$A[j] \leq x$



average here now swap the values $A[i]$ and $A[j]$

3	5	7	8	9	10

this is the final array.

Q3) True or False.

(Q.1) $n+3 \in \Omega(n)$

condition says

$\Omega\, g(n) = f(n)$ there exists constants. $c>0$, $n_0>0$ such that.

$0 \leq c\,g(n) \leq f(n)$ for all $n \geq n_0$

Hence

$0 \leq cn \leq n+3$.

Taking no w.r.t to c & solving above in equalities for any constants.

$$cn \leq n+3$$
$$-3 \leq n-cn$$
$$-3 \leq n(1-c)$$
$$\frac{-3}{1-c} \leq n.$$

condition (i) $n=2$ and $c=3$.

$$\frac{-3}{1-3} \leq 2$$
$$\frac{-3}{-2} \leq 2$$
$$1.5 \leq 2$$

which is true.

Hence $n+3 \in \Omega(n)$ is also true.

Condition (ii) / case 2)

Taking No w.r.t to C.

$$n_0 = \frac{-3}{1-c}$$

taking $c = 2$ $(c > 0)$

$$n_0 = 1.5 \quad (n_0 > 0)$$

Taking $n \geq n_0$

assume $n = 3$. so

$3 > 1.5$ - satisfies $n \geq n_0$.

so substituting in

$$cn \leq n+3.$$

$$2 \times 3 \leq 3+3$$

$6 = 6$. is also true

$\therefore$

$n+3 \in \Omega(n)$ is true.

(ii) $n+3 \in O(n^2)$

For big omega $f(n) = \Omega g(n)$ there are constants such that $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$

$g(n) = n^2$

$f(n) = n+3$.

so. $f(n) \leq cg(n)$

$0 \leq n+3 \leq cn^2$

solving no w.r.t to c will give us no.

Hence

$$cn^2 - n - 3 \geq 0$$

quadratic equation to take out roots.

$$= \frac{-b \pm \sqrt{b^2 \mp 4ac}}{2c}$$

$$n_0 = \frac{1 \pm \sqrt{1+12c}}{2c}$$

taking $c = 1$ and $n = 3$ (where $c > 0$, $n > 0$. and no +ve value is $(2,3)$ so $n \geq n_0 > 0$)

$1(3)^2 - 3 - 3 \geq 0$

$3 \geq 0$

which is true, hence $n+3 \in O(n^2)$

(iii). $n+3 \in \Theta(n^2)$

For big Θ.

$\Theta(g(n)) = O\,g(n) \cap \Omega\, g(n)$ such that for

$f(n)$ there exist a constant where c_1, c_2 such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Taking $g(n) = n^2$

and then

$f(n) = n+3$.

$0 \leq c_2 g(n) \leq n+3$. [Taking condition of $\Omega g(n)$ where $f(n) \leq c_2 g(n)$]

$c_2 n^2 \leq n+3$.

Here $c_1 \leq \frac{n+3}{n^2}$

$c_1 \leq \frac{1}{n} + \frac{3}{n^2}$

taking, $c_2 = 1$, $n = 1$.

we get

$1(1)^2 \leq 1+3$

$1 \leq 4$

Hence this is true.

Now, $c_1 g(n) \leq f(n)$ seems to be true

now take

$$f(n) \leq c_2 g(n)$$

$$n+3 \leq c_2 n^2$$

Taking $n = 1$ $\quad n \geq n_0 > 0$

$c_2 = 2$ $\quad c_2 > 0$

$$1+3 \leq 2(1)$$

$$4 < 2$$

This condition is not true

hence.

$n+3 \in e(n^2)$

is also not true. and is false.

(iv) $2^{n+1} \in O(n+1)$

For big omega $f(n) = O\,g(n)$ there are constants such that $c>0$, $n_0>0$ such that $0 \le f(n) \le c\,g(n)$ for all $n \ge n_0$

taking $g(n) = n+1$
$f(n) = 2^{n+1}$

so $f(n) \le c\,g(n)$

$2^{n+1} \le c(n+1)$

substituting a value $c = \underline{1}$ and $n = \underline{1}$ we get. $(\underline{c>0}, \underline{n>1})$

$2^{1+1} \le \underline{1}(1+\underline{1})$

$2^2 \le 2$

Hence this condition is false.

Therefore.

the condition is false.

(v) $2^{n+1} \in \Theta(2^n)$

For theta big Θ the condition says that.

$\Theta g(n) = O\,g(n) \cap \Omega\, g(n)$

$f(n)$ there is enisisting such that a constant exist there where c_1, c_2 are such that

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$
and c_1 and c_2 are some constants.

taking part one by taking

$$g(n) = 2^n$$
$$f(n) = 2^{n+1}$$

$$0 \leq c_1 2^n \leq 2^{n+1} \leq c_2 2^n$$

$$c_1 = 1$$
$$c_2 = 3$$

taking $n = 1$ when $c_1, c_2 > 0$ & $n > 0$

$$0 \leq (1)2^{(1)} \leq 2^{1+1} \leq 3(2^1)$$

$$0 \leq 2 \leq 2^2 \leq 6$$

$$0 \leq 2 \leq 4 \leq 6$$

Hence the condition is true

Hence $2^{n+1} \in \Theta\, 2^n$ is also true.

(Q.4) Using master method determine $T(n)$ for following recurrences,

Ans

(i) $T(n) = 8T\left(\frac{n}{2}\right) + n.$

taking $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

where $a = 8 \quad a \geq 1$

$b = 2 \quad b > 1$

find $n^{\log_b a} = n^{\log_2 8} = n^3.$

case 1: $f(n) = O(n^{\log_b a - \varepsilon})$ where $\varepsilon > 0$

$n \leq O(n^{3-\varepsilon})$ for $\varepsilon > 0$ should be true so. big omega says that.

$f(n) \leq c\, g(n)$

so here $f(n) = n$ and $g(n) = n^{3-\varepsilon}$

$n \leq c_1 n^{3-\varepsilon}$ $\quad c_1 > 0 \quad \varepsilon > 0$

$n \geq n_0 > 0$

taking $n = 1$

$c_1 = 1$

$\varepsilon = 1$

$1 \leq 1$ is true hence

$f(n) = O(n^{\log_b a - \varepsilon})$ is true

$f(n) = O(n^{3-\varepsilon})$ is true.

since $n^{\log_b a}$ i.e n^3 is largest than the $f(n)$ i.e n.

$n^3 > n$

$n^{\log_b a} > f(n)$ //

Here $T(n) = \theta(h^{\log_b a})$

$T(n) = \theta(n^{\log_2 8})$

$\boxed{T(n) = \theta(n^3)}$ // (ans)

(ii) $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

For the master theorem in recurrances it says that for

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Here $a \geq 1$ and $b > 1$ are constants, let $f(n)$ be a function so $T(n)$ can be defined above on the non-negative integers by recurrances.

here $\frac{n}{b}$ is intercept and considering the asymptotic bounds we get and take case

case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$,

Here $f(n) = n^2$

$a = 8$

$b = 2$

Hence $n^2 = O(n^{\log_2 8 - \varepsilon})$

$n^2 = O(n^{3-\varepsilon})$ for some $\varepsilon > 0$

as big O states that.

$f(n) \leq c\, g(n)$

Hence $n^2 \leq c_1 n^{(3-\epsilon)}$ for some $\epsilon > 0$

$n^2 \leq c_1 n^{(3-\epsilon)}$ taken $n = 1$, $\epsilon = 1$, $c_1 = 2$

$1 \leq 2$ which is true hence condition

$f(n) = O(n^{3-\epsilon})$

$n^2 = O(n^{3-\epsilon})$ is true.

Moreover

$n^{\log_b a} > f(n)$

$n^3 \not> n^2$

∴ Solution for case 1 in master theorem is

$T(n) = \Theta(n^{\log_b a})$

$= \Theta(n^{\log_2 8})$

$\boxed{T(n) = \Theta(n^3)}$ //

(iii) $T(n) = 8T\left(\frac{n}{2}\right) + n^3$.

Ans Let there be constants $a \geq 1$, $b > 1$, $f(n)$ can be a function. a and b are constants so $T(n)$ can be defined a function where

$T(n) = a\left(\frac{n}{b}\right) + f(n)$

where $\frac{n}{b}$ is a inter cept.

Here $f(n) = n^3$.

and $a = 8$

$b = 2$

so $n^{\log_b a} = n^{\log_2 8} = n^3$.

so,

if $f(n) = \Theta(n^{\log_b a})$ considering case 2: of master theorem. we see that.

$f(n)$ is equal to $n^{\log_b a}$ i.e.

$f(n) = n^{\log_b a}$

$n^3 = n^3$.

∴ in case 2:

$n^3 = \Theta(n^3)$. is true as

$0 \leq c_1 n^3 \leq n^3 \leq c_2 n^3$ taking $c_1 = 1, c_2 = 2 \Rightarrow 0 \leq 1 \leq 1 \leq 2$ is true

Ans is: $T(n) = \Theta(n^{\log_b a} \log n)$.

$$\boxed{T(n) = \Theta(n^3 \log n).}$$

as the case 3 gets satisfied.

iv). $T(n) = 8T\left(\frac{n}{b}\right) + n^4$

Ans. Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function, then let $T(n)$ be defined as non-negative integer by recurrance

$$T(n) = aT(n/b) + f(n).$$

where we interpret n/b be a mean either $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$.

$f(n) = n^4$

$a = 8, \; b = 2.$ $= n^{\log_b a} = n^{\log_2 8} = n^3$ //

$$n^{\log_b a} > f(n)$$

$$n^4 > n^3.$$

Taking <u>case 3</u>

If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta f(n)$

$\therefore f(n) = \Omega(n^{3+\epsilon})$

$n^4 = \Omega(n^{3+\epsilon})$

$0 \leq cg(n) \leq f(n)$

$c_1 n^{3+\epsilon} \leq n^4$ Taking $c_1 = 1$ $\epsilon = 1$

$n = 1$

The above condition is true.

Also, $af\left(\frac{n}{b}\right) \leq (1+\epsilon) f(n)$

$(1-\epsilon')$ is where a and n is constant large value $\epsilon' > 0$.

$8 f\left(\frac{n}{2}\right)^4 \leq (1+\epsilon) n^4$

$8 \frac{n^4}{2^4} \leq (1+\epsilon') n^4$

$\frac{n^4}{2} \leq (1-\epsilon') n^4$

also hold to be true for value $\epsilon = 0.5$

$\frac{1}{2} \leq 0.5$

they are equal so

$af\left(\frac{n}{b}\right) \leq (1+\epsilon) f(n)$ is also true.

$\therefore$ $T(n) = \Theta f(n)$.

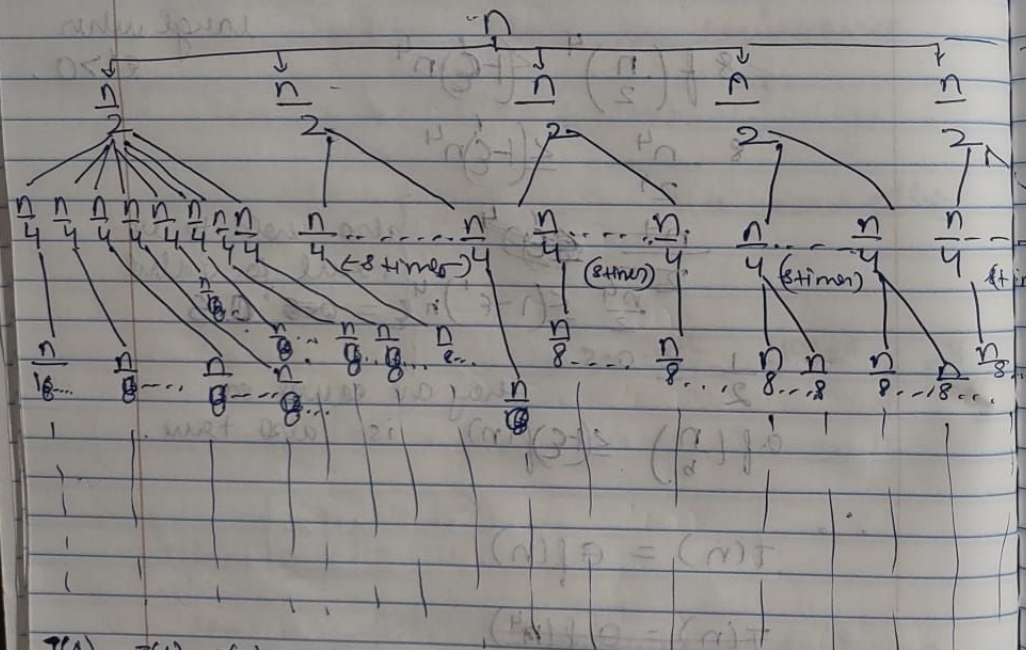
~~$T(n) = \Theta f(n^4)$~~

$\boxed{T(n) = n^4}$ //.

$Tn = \Theta f(n) = n^4$ //.

Q.5 Recursion tree for

$$T(n) = 8T\left(\frac{n}{2}\right) + n.$$



Leaves nodes at the bottom $= n^{\log_2 8} = n^3$.

$\rightarrow T(n) = 8T(n/2) + n$

$\rightarrow T(n/2) = 8T(n/4) + n/2$

$\rightarrow T\left(\frac{n}{4}\right) = 8T(n/8) + n/4$

$T\left(\frac{n}{8}\right) = 8T(n/16) + \frac{n}{8}$

$\hookrightarrow \theta(n^{\log_b a})$

$\theta(n^{\log_2 8})$

$\theta(n^3)$.

$$\therefore T(n) = n^{\log_b a} T(1) + \sum_{i=0}^{h-1} 2^i n$$

Total nodes leaf node = $\frac{3}{n}$

height $= \log_b n = \log_2 n$

Level #	# of nodes	
0	1	$\rightarrow \Sigma n$
1	8	$\rightarrow \Sigma = 4n = 8 \times \frac{n}{2}$
2	16	$\rightarrow \Sigma = 4^2 n = 8^2 \times \frac{8}{n^2} = 4^2 n$
i	8^i	$\rightarrow \Sigma = 4^i n$

Taking geometric progression:

$$\sum_{i=0}^{i=\log_2 N} 4^i = \frac{n(4^{\log_2 n} - 1)}{4-1} = \frac{n(n^2-1)}{3}$$

$$t(n) \leq n^3$$

$$\therefore \boxed{T(n) = \Theta n^3}$$

using the substitution method we get.

$$T(n) = \Theta(n^3)$$

substitution requires the above method as

(i) $T(n) \leq cn^3$

$\therefore T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)^3$

by putting it in the above recurrence we get.

$$T(n) \leq 8.c.\left(\frac{n}{2}\right)^3 + n$$

$$\leq 8.c.\frac{n^3}{8} + n$$

$$T(n) \leq cn^3 + n. \qquad = \Theta^3.$$

as this gets satisfied so we can say

$$T(n) = \Theta(n^3).$$

(ii) ALSO using another substitution eg.

$$T(n) = c_1n^3 - c_2n$$

where

Assuming. $T(k) \leq c_1k^3 - c_2k.$

for $k \leq n.$

$$T(n) = 8\left(\frac{n}{2}\right) + n \leq 8.\left[c_1\left(\frac{n}{2}\right)^3 - c_2\left(\frac{n}{2}\right)\right]$$

$$8\left(\frac{n}{2}\right) + n \leq c_1 n^3 - 4c_2 n.$$

$$4n + n \leq (c_1 n^3 - c_2 n) - 3c_2 n + n.$$

where c_1 and c_2 are positive constants. and c_1 is sufficiently greater than c_2 having base case as below

Base case $T(1) \leq c_1 - c_2$.

Hence

$$T(n) \leq (c_1 n^3 - c_2 n) - \underbrace{3c_2 n + n}_{\text{residual part} > 0}$$

∴ This substitution is also true.

$T(n) \leq c_1 n^3 - c_2 n$ is true. and gets satified by substitution method.

Hence

$T(n) = \Theta n^3$.