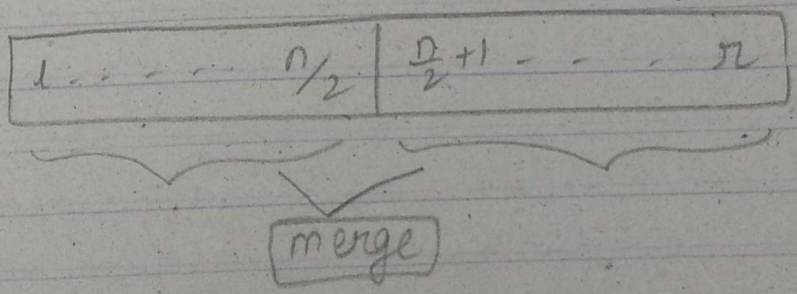
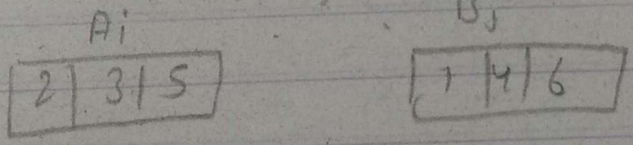
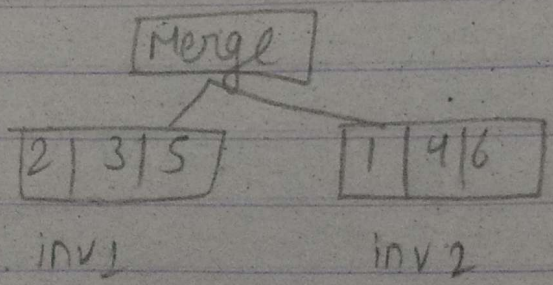
```
{
    temp[k++] = arr[j++];
    inv_count = inv_count + (mid-i);
}
while(i<=mid)
    temp[k++] = arr[i++];
while(j<=right)
    temp[k++] = arr[j++];
for(i=left; i<=right; i++)
    arr[i] = temp[i];
return inv_count;
}
```





Ai > Bj

if (A[i] > B[j]) then there are (mid-i) inversions



inv1 + inv2 + inversions that cross $\frac{b}{n}$ first + second

Q Give an algorithm that determines the number of inversions in any permutation on n elements in O(nlogn) worst case time

→
```
MergeSort(int arr[], int n)
{
    int *temp = (int*) malloc (sizeof(int) * n);
    return A_Mergesort(arr, temp, 0, n-1);
}

A_MergeSort(int arr[], int temp[], int left, int right)
{
    inv_count = 0;
    if (right > left)
    {
        mid = (left + right)/2
        inv_count = A_Mergesort(arr, temp, left, mid)
        inv_count += A_Mergesort(arr, temp, mid+1, right)
    }
    return inv_count;
}

int merge(int arr[], int temp[], int left, int mid,
                                        int right)
{
    j=left, j=right, k=left
    inv_count = 0;
    while (i<=mid && j<=right)
    {
        if (arr(i) <= arr(j))
            temp[k++] = arr(i++);
        else
```

Q. Find the total no. of operation performed by the following code. Express the same in θ notation.

```
Algorithm Exponentiate(x,n)
{
    m=n; power=1; z=x;                    ---- c1
    while (m>0) do
    {
        while (m mod 2 == 0) do
        {
            m = ⌊(m/2)⌋;                  }
            z = z * z;                    } ---- c2 log2 n
        }
        m = m-1;
        power = power * z;
    }
    return power;
}
```

Eg:- Expo (2, 6)

m = 6, power = 1, z = 2

{ m = 3
z = 2 * 2 = 4 }

m = 2, power = 4

{ m = 2/2 = 1
z = 4 × 4 = 16 }

m = 0, power = 4 × 16 = 64

∴ power = 64

$2^6 = 64$

running time $c_2 \log_2 n = O(\log_2 n)$

($\log n$)

Scanned by CamScanner

8 Suppose you have algorithm with the following running time listed below. (Assume these are the exact running times). How much slower do each of these algorithms get when you double the input size?

(i) n^2

(ii) n^3

(iii) $100n^2$

(iv) $n\log_2 n$

(v) 2^n

(i) n^2

$(2n)^2 = 4n^2$

$\frac{4n^2}{n^2} = 4$ times

(ii) n^3, $(2n)^3 = 8n^3$

$\frac{8n^3}{n^3} = 8$ times

(iii) $100n^2$, $100(2n)^2 = 400n^2$

$\frac{400n^2}{100n^2} = 4$ times

(iv) $n\log_2 n$, $2n\log_2(2n) = 2n(\log_2 2 + \log_2 n) = 2n(1+\log_2 n)$

$\frac{2n(1+\log_2 n)}{n\log_2 n} = \frac{2}{\log n} + 2$

(v) 2^n, $2^{2n} = 4^n$

$\frac{2^{2n}}{2^n} = 2^n$ times slower

Q An algorithm SELECT, finds the i^{th} smallest by dividing the input elements into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Prove that SELECT does not run in linear time if group of 3 are used.

→

Consider the analysis of the algorithm for group of k. The number of elements less than (or greater than) the median of the medians x will be at least $\frac{k}{2}\left(\frac{1}{2}\left\lceil\frac{n}{k}\right\rceil - 2\right) \geq \frac{n}{4} - k$.

Hence in the worst case SELECT will be called recursively on at most $n - \left(\frac{n}{4} - k\right) = \frac{3n}{4} + k$ elements

The recurrence is

$$T(n) \leq T\left(\left\lceil\frac{n}{k}\right\rceil\right) + T\left(\frac{3n}{4} + k\right) + O(n).$$

Solving by substitution we obtain a bound for which k the algorithm will be linear.

Assume $T(n) \leq cn$ for all smaller n. we have

$$T(n) \leq c\left\lceil\frac{n}{k}\right\rceil + c\left(\frac{3n}{4} + k\right) + O(n)$$

$$\leq c\left(\frac{n}{k} + 1\right) + \frac{3cn}{4} + ck + O(n)$$

$$\leq \frac{cn}{k} + \frac{3cn}{4} + c(k+1) + a(n)$$

$$= cn\left(\frac{1}{k} + \frac{3}{4}\right) + c(k+1) + a(n)$$

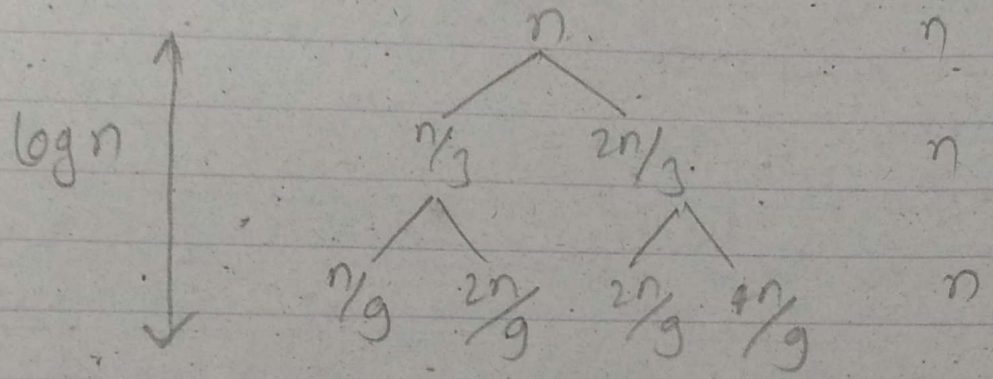
$$\leq cn$$

where the last equation only holds for $k > 4$.

Thus, we have shown that the algorithm will compute in

linear time for any group size of 4 or more.
The algorithm is $\Omega(n \log n)$ for $k = 3$

$$T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$



Total cost $\Omega(n \log n)$

Q Prove that $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$ $\sum_{j=0}^{n} 2^j = \frac{2^{n+1} - 1}{2-1}$

1. Inductive base
   Substituting $n = 0$ in $2^{0+1} - 1 = 2 - 1 = 1$
   The first term in the sum $1 + 2 + 2^2 + \dots + 2^n$ is 1 and hence S_0 is 1 ∴
   ∴ P(0) is true
2. Inductive step
   → Prove that if $S_K = 2^{K+1} - 1$
   then $S_{K+1} = 2^{K+2} - 1$
   → Assume that $P(K): S_K = 2^{K+1} - 1$ is true
   we will show that under this assumption
   $P(K+1): S_{K+1} = 2^{K+2} - 1$ is also true.
   → $S_{K+1} = S_K + 2^{K+1}$ by the definition of subsequent sums here 2^{K+1} is the $(K+1)^{th}$ term in the sum
   → Substituting S_K we obtain:
   $S_{K+1} = 2^{K+1} - 1 + 2^{K+1} = 2 \cdot 2^{K+1} - 1 = 2^{K+2} - 1$
   ∴ $S_{K+1} = 2^{K+2} - 1$

By principle of mathematical induction it follows that for all integers n, $S_n = 2^{n+1} - 1$

Q Consider the following list of functions and arrange them in ascending order of their growth rate using Big-oh notation

$g_1(n) = 2^{\sqrt{\log n}}$, $g_2(n) = 2^n$, $g_3(n) = n^{4/3}$, $g_4(n) = n(\log n)^3$

$g_5(n) = n^{\log n}$, $g_6(n) = 2^{2^n}$, $g_7(n) = 2^{n^2}$

$2^{\sqrt{\log n}}$ $\quad 2^n$ $\quad 2^{2^n}$ $\quad 2^{n^2}$

$\sqrt{\log n}\log 2$ $\quad n\log 2$ $\quad 2^n \log 2$ $\quad n^2 \log 2$

$(\log n)^{1/2}$ $\quad n$ $\quad 2^n$ $\quad n^2$

$\frac{1}{2}\log\log n$ $\quad \log n$ $\quad n\log 2$ $\quad 2\log n$

$n = 2^{128}$

$\frac{1}{2}\log 128$ $\quad 128\log 2$ $\quad 2^{128}\log 2$ $\quad 2\cdot 128\log 2$

$\frac{1}{2}\cdot 7\log 2$ $\quad 128\log 2$ ④ $\quad 2^{128}\log 2$ ① $\quad 256\log 2$ ③

$2.5\log 2$ ②

$g_6 > g_7 > g_2 > g_1$

$n^{4/3}$ $\quad n(\log n)^3$ $\quad n^{\log n}$

$\frac{4}{3}\log n$ $\quad \log n + 3\log\log n$ $\quad \log n \cdot \log n$

$n = 2^{128}$

$\frac{4}{3}\cdot 128\log 2$ $\quad 128\log 2 + 21\log 2$ $\quad (128)^2(\log 2)^2$

$170.666\log 2$ ④ $\quad 149\log 2$ ⑤ $\quad$ ②

$g_5 > g_3 > g_4$

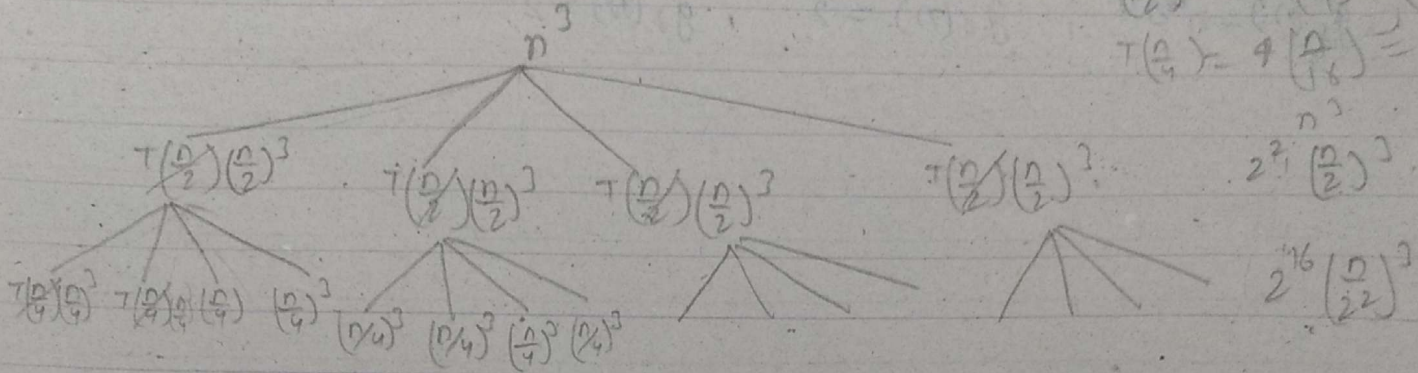
$g_1 < g_2 < g_4 < g_3 < g_7 < g_5 < g_6$

Q Solve the following recurrence using recursion tree method

$$T(n) = \begin{cases} 4T(n/2) + n^3 & \text{if } n > 1 \\ 1 & \text{if } n \le 1 \end{cases}$$

$T(\frac{n}{2}) = 4T(\frac{n}{4}) + (\frac{n}{2})^3$

$T(\frac{n}{4}) = 4(\frac{n}{8}) = (\frac{n}{4})^3$



No. of node at each level 4^i

subproblem size at $i = \frac{n}{2^i}$

$\frac{n}{2^i} = 1 \Rightarrow n = 2^i$

$i = \log_2 n$

Cost of each subproblem at level $i = c\left(\frac{n}{2^i}\right)^3$

Total cost $= \left(\frac{n}{2^i}\right)^3 c \cdot 4^i$ $\quad c\left(\frac{n}{2^i}\right)^3 4^i$

$= cn^3 \frac{4^i}{2^{3i}} = cn^3 \left(\frac{4}{8}\right)^i$

$= cn^3 \left(\frac{1}{2}\right)^i$

Total cost $= \sum_{i=0}^{\log_2 n} cn^3 \left(\frac{1}{2}\right)^i + 4^{\log_2 n} \cdot 1$

$= cn^3 \left(\frac{1/2}{1-\frac{1}{2}}\right) + 4^{\log_2 n}$

$= cn^3 + 4^{\log_2 n}$

$O(n^3)$

Q Find the asymptotic tight bound for the following

(i) $T(n) = 3T\left(\frac{n}{2}\right) + n\log n$

(ii) $T(n) = 2T\left(\frac{n}{4}\right) + n$

(iii) $T(n) = 2T(\sqrt{n}) + 1$

$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$

(i) $T(n) = 3T\left(\frac{n}{2}\right) + n\log n$

$a=3, b=2, k=1, p=1$

$a > b^k$

$3 > 2^1$

$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_2 3})$

(ii) $T(n) = 2T\left(\frac{n}{4}\right) + n$

$a=2, b=4, k=1, p=0$

$a < b^k$

$2 < 4^1, \quad p \geq 0$

$T(n) = \theta(n^k \log^p n) =$

$= \theta(n \log^0 n) = \theta(n)$

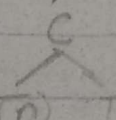
(iii) $T(n) = 2T(n^{1/2}) + 1$

$a=2, b=1, k=0$

$a \; b^k$

$2 > 1^0$

$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_1 2})$



Q Use Master method to solve the recurrence

$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n \log n)$

$a=2, b=2, k=1, p=1$

$a \; b^k$

$2 = 2^1, \quad p > -1$

$T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_2 2} \log^2 n)$

$= \theta(n \log^2 n)$

Q Suppose you have four algorithms with the running time listed below. These running times are the exact number of instructions performed as a function of the input size n. Suppose you have a computer that can perform 10^{10} instructions per second. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

(i) n^2 (ii) n^3 (iii) $100n^2$ (iv) 2^n (v) $n\log n$

Speed = 10^{10} inst/sec

Time = 1 hour = $60 \times 60 = 3600$ sec

(i) n^2 ⇒

$10^{10} = \frac{n^2}{3600} \Rightarrow n^2 = 10^{10} \times 3600 \Rightarrow n^2 = 3.6 \times 10^{13}$

$n = 6000000$

(ii) n^3

$10^{10} = \frac{n^3}{3600} \Rightarrow n^3 = 10^{10} \times 3600 = 3.6 \times 10^{13}$

$n = 33019.27249$

$\therefore n = 33019$

(iii) $100n^2$

$10^{10} = \frac{100n^2}{3600} \Rightarrow n^2 = 10^{10} \times 36 = 3.6 \times 10^{11} \Rightarrow n = \sqrt{3.6 \times 10^{11}}$

$\therefore = 600000$

(iv) 2^n

$10^{10} = \frac{2^n}{3600} \Rightarrow 2^n = 10^{10} \times 3600 = 3.6 \times 10^{13} \Rightarrow n\log 2 = \log(3.6 \times 10^{13})$

$n = \frac{13.556302}{0.30102}$

$n = 45.033062$

$\therefore n = 45$

(v) $n\log n = 10^{10} \times 3600 = 10^{12} \times 6^2$

Q Define Θ Notation. Prove that $f(n) = 5n^2 + 6n - 2 = \Theta(n^2)$

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$

c_1, c_2, n_0 are +ve constant

$f(n) = 5n^2 + 6n - 2$

$g(n) = n^2$

$c_1 g(n) \le f(n)$

$c_1 n^2 \le 5n^2 + 6n - 2$

$c_1 = 1 \; ; \; n_0 = 1$

$1 \cdot 1 \le 5 + 6 - 2 = 9$

$1 \cdot 2 \le 30$

$\therefore c_1 = 1, \; n \ge 1$

$c_1 = 1, \; n_0 = 1 \Rightarrow 1 \le 9$ ✓

$c_1 = 1 \; n_0 = 2 \Rightarrow 4 \le 30$ ✓

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

$5n^2 + 6n - 2 \le c_2 n^2$

$n_0 = 1$

$5 + 6 - 2 \le c_2 \cdot 1$

$9 \le c_2$

$c_2 = 9, \; n_0 \ge 2$

$30 \le 9 \cdot 4 = 36$ (1)

$c_2 = 8 \; n_0 = 1 \; , c_2 = 8, n_0 = 2$

$9 \not\le 8$ ✗ $30 \le 16$ ✗

$c_2 = 9 \; n_0 = 2 \Rightarrow 30 \le 36$ ✓

$c_2 = 9, \; n_0 = 3 \Rightarrow 61 \le 81$ ✓

Q Find the asymptotic upper bound for the recurrence

$T(n) = 2T(n-1) + \Theta(n)$.

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

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

$= n + 2[2T(n-2) + n-1] = n + 2(n-1) + 2^2 T(n-2)$

$= n + 2(n-1) + 2^2 [2T(n-3) + n-2] = n + 2(n-1) + 2^2(n-2) + 2^3 T(n-3)$

$= n + 2(n-1) + 2^2(n-2) + \cdots \; 2^{k-1}(n-k+1) + 2^k T(n-k)$

$n - k = 1 \Rightarrow k = n-1$

$= n + 2(n-1) + 2^2(n-2) + \cdots 2^{n-2} \cdot 2 + 2^{n-1} \cdot 1$

$2^k(n-k)$

$a = 2, \; b = 1, \; k = 1$

$\therefore a > b^k$

$2 > 1$

$T(n) = O(n^{\log_b a}) = O(n^{\log_1 2})$

Q (A) Consider 2 programs A and B whose running times are $T_A = 100n$ and $T_B = 2n^2$ respectively. Suppose that both these running times are the number of milliseconds taken on a particular computer on an input of size n. Find the range of values for n for which program B runs faster than A. Also plot the running time against n values.

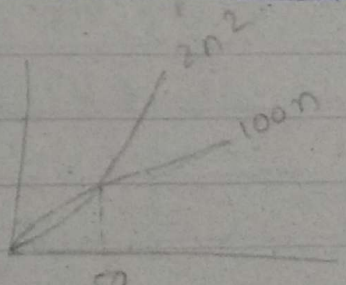
(B) Consider the programs A and B above. Complete the entries in the following table by finding the maximum problem size allowed for programs A and B for a time (sec) given in column 1.

Time sec	Maximum Problem size solvable with program A	Maximum Problem size solvable with program B
1	10	?
10	? 100	?
100	? 1000	?
1000	? 10000	?

(A) $T_A = 100n$, $T_B = 2n^2$

$2n^2 = 100n$

$n = 50$



$n < 50$: $T_A > T_B$, $100n > 2n^2$

$n > 50$: $T_A < T_B$, $100n < 2n^2$

∴ range is $0 < n < 50$

(B) Speed $= \frac{100 \times 10}{1} = 10^3$

$B_1 = 10 \times 1 = 10$

$A_2 = 10 \times 10 = 100$, $A_3 = 10 \times 100 = 1000$

Matrix-Chain-Order(p)

{

n = p.length - 1

Let m[1...n, 1...n] and s[1...n, 2...n] be new tables

for i = 1 to n

m[i, i] = 0

for l = 2 to n

for i = 1 to n - l + 1

j = i + l - 1

m[i, j] = ∞

for k = i to j - 1

q = m[i, k] + m[k+1, j] + $p_{i-1} p_k p_j$

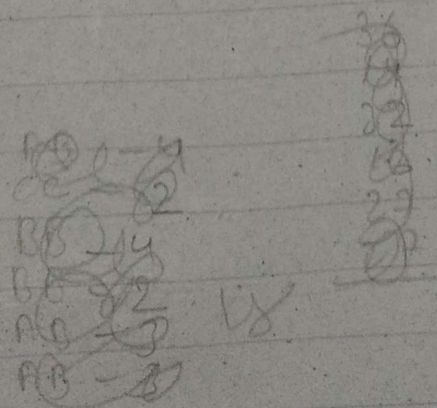
if q < m[i, j]

m[i, j] = q

s[i, j] = k

return m and s.

}



1050

200

91 74112 25461

quietambience

Randomized selection problem
Selection in worst case linear time
Implement binary heap with all operation

9 6 5 0 8 2 4 7

i j

9<7 ✗

i j

6<7 ✓

i

6 9 5 0 8 2 4 7

i j 5<7 ✓

i

6 5 9 0 8 2 4 7

i j 0<7 ✓

6 5 0 9 8 2 4 7

i j 8<7 ✗

j

i 2<7 ✓

6 5 0 2 8 9 4 7

i i 4<7 ✓

6 5 0 2 4 9 8 7

i

6 5 0 2 4 7 8 9

exchange A[i] & A[j]

KMP
NP
DBST

$x+2=0$

$2x+y+2=0$

$n = 123/10 = 12$

$rem = 123\%10 = 3$

$rem = n\%10$