Quadratic Time Algorithm

```
Algorithm Transpose(a,n){

for (int i = 0; i <= N; i++)

    for (int j = i+1; j <= N; j++)

        swap(A[i][j], A[j][i]);

}
```

$\boxed{\Theta(n^2)}$

2 Loop Analysis

$i=2$

Swap is constant time

$$\sum_{i=1}^{n} \sum_{i+1}^{n} 1$$

$n - i - \cancel{1} + \cancel{1}$

$UB - LB + 1$

A^T

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

diagonal element

$i:$

$i=2$

$j=3$

$i=3$

$j=4$

$$\begin{matrix} \boxed{1} & 2 & 3 & 4 \\ 5 & 6 & \boxed{7} & 8 \\ 9 & 10 & \boxed{11} & 12 \\ 13 & 14 & 15 & 16 \end{matrix}$$

$$\begin{matrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 8 \\ 3 & 7 & 11 & 15 \\ 4 & 14 & 12 & 16 \end{matrix}$$

$$= \sum_{i=1}^{n} n - i$$

$(1+2+3 \cdots N)$

$$= n^2 - \sum_{i=1}^{n} i$$

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

$$= \frac{n(n-1)}{2}$$

Design An algorithm

Computing Matrix Multiplication

Cubic Time Algorithm

```
Algorithm Mult(a,b,c,n){
for i = 1 to n do
    for j=1 to n do{
          c[i,j]= 0;
        for k=1 to n
          c[i,j]= c[i,j]+a[i,k]*b[k,j]
     }
}
```

Binary Search

```
int binarySearch(int arr[], int lb, int ub, int x){
int mid;
if (lb > ub)
      return -1
else{
      mid = (lb + ub)/2;
      if (arr[mid] == x)
            return mid;
      else
      if (x < arr[mid])
            return binarySearch(arr, lb, mid-1, x);
      else
            return binarySearch(arr, mid+1, ub, x);
   }
}
```

Searching – Sorted (ordered Array)

Binary Search

Searching unordered List

Linear Sear

Binary Search

93

$$\frac{0+14}{2} = \frac{14}{2} = 7$$

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary Search

33 with 53

bi

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Lo

UB mid

UB

Binary Search

mid: $\frac{0+6}{2} = \underline{3}$

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Lw

High

Binary Search

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Lo

Low

Hi

Binary Search

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

LB UB

Binary Search

$$mid = \frac{6+4}{2} = 5$$

$$UB = mid - 1$$

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

LB

Binary Search

$\frac{3}{2}$ integer

1

Value	6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Lo

UB

mid

$\frac{4+4}{2} = 4$

Binary Search

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Number of Comparisons

$\Omega(1)$ Best case

$O(\log_2 n)$ - worst case

1 comparison — $n/2$

Divide & Conquer

2 - comparison — $n/2^2$

⋮

k^{th} comparison — $n/2^k$

$$\frac{n}{2^k} = 1 \qquad n = 2^k$$

$$k = \log_2 n$$

Time Complexity

Horner's rule

Basic Algorithm

$A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots$

Horner Rule is used for evaluation of polynomial with minimum No. of multiplication.

$$P(x_0) = x_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3$$

How many multiplication Required for evaluating the polynomial?

$$= x_0 + a_1 \cdot x_0 + a_2 \cdot x_0 \cdot x_0 + a_3 \cdot x_0 \cdot x_0 \cdot x_0$$

$$1 + 2 + 3 = 6$$

Horner Rule

$$P(x_0) = x_0 + x_0 (a_1 + a_2 x_0 + a_3 x_0^2)$$

$$= x_0 + \underbrace{x_0}_{3} (a_1 + \underbrace{x_0}_{2} (a_2 + \underbrace{a_3 x_0}_{1})) \leftarrow$$

No. of multiplication Now

Horner's rule

- Horner's rule is a means for evaluating a polynomial at a point x_0 using a minimum number of multiplications. If the polynomial is

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_0$$

- then Horner's rule is

Horner's rule

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_0$$

- then Horner's rule is

$$A(x_0) = (\ldots(a_n x_0 + a_{n-1})x_0 + .. + a_1)x_0 + a_0$$

GATE | GATE-CS-2014-(Set-3) | Question 21

The minimum number of arithmetic operations required to evaluate the polynomial $P(X) = X^5 + 4X^3 + 6X + 5$ for a given value of X using only one temporary variable.

(A) 6

(B) 7 [B]

(C) 8

(D) 9

Gate - 2014
Gate - 2006 } IIT-KGP

Count - Addition & Multiplication

$$P(x) = 5 + 6x + 4x^3 + x^5$$
$$= 5 + x(6 + 4x^2 + x^4)$$
$$= 5 + x(6 + x(4x + x^3))$$
$$= 5 + x(6 + x(x(4 + x^2)))$$

$t = x * x$ (1)
$t = t + 4$ (2)
$t = t * x$ (3)
$\rightarrow t = t * x$ (4)
$t = t + 6$ (5)
$t = t * x$ (6)
$t = t + 5$ (7)

$5 + x(6 + x^2(4 + x^2))$ (3) (1) (7)

GATE | GATE-CS-2006 | Question 1

Consider the polynomial $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, *where* $a_i \neq 0 \; \forall i$. The minimum number of multiplications needed to evaluate p on an input x is:

(A) 3

(B) 4

(C) 6

(D) 9

[A]

Concept

Not Arithmetic operation – 6

Multiplication – 3

$p(x) = a_0 + x(a_1 + a_2x + a_2x^2)$

$= a_0 + \underbrace{x}_{3}(a_1 + \underbrace{x}_{2}(a_2 + \underbrace{a_3x}_{1}))$

Computing x^n

Simple programming

Complexity?

```
Algorithm power(x, n) {
    power = 1;
    for i = 1 to n
        power = power * x
}
```

$\Theta(n)$

$T(n) = T(n-1) + 2$, ~~$T(0) = 1$~~

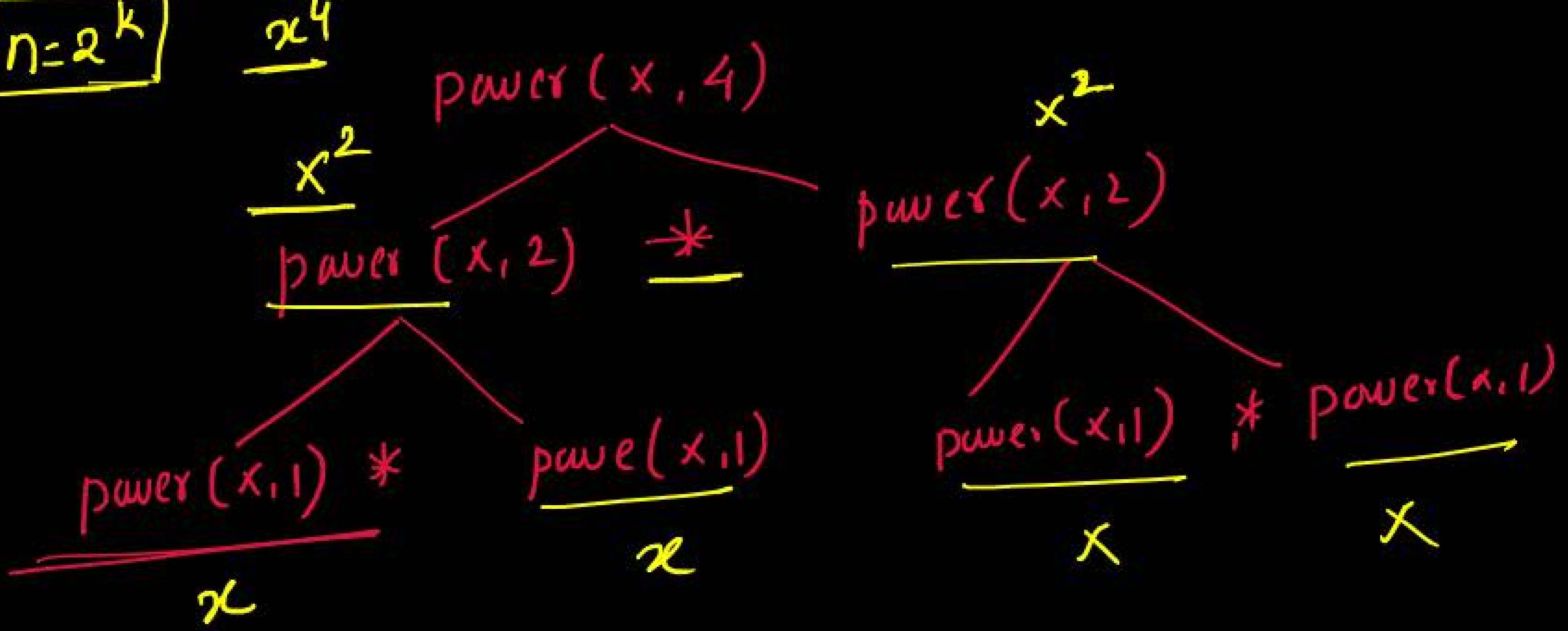
$x \cdot x \cdot x = x^3$ $T(1) = 2$

```
Algorithm power(x, n) {
    if (n == 1)
        return x;
    else
        return x * power(x, n-1)
}
```

n is some 2^k

```
Algorithm power(x, n) {
    if (n == 1)
        return x
    else
        return power(x, n/2) * power(x, n/2)
}
```

$\boxed{n = 2^k}$



$T(n) = 2T(n/2) + 2$

$T(1) = 2$

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

$T(1) = 1$

$\boxed{\Theta(n)}$

Computing x^n

compute x^n for any real number x and integer $n \geq 0$. A naive algorithm for solving this problem

Computing x^n

Computing x^n

compute x^n for any real number x and integer $n \geq 0$. A naive algorithm for solving this problem

Naive

```
power := x
for i := 1 to n - 1 do
    power := power * x;
```

Computing x^n

The number of multiplication performed by algorithm is

Linear time

$n-1$

Better

Repeated squaring

x

x^2 $\quad x^2 * x^2 = x^4$

x^3

x^4 $\quad x^4 * x^4 = x^8$

⋮

x^n $\quad x^8 * x^8 = x^{16}$

power = x

for i = 1 to n - 1

power = power * x

Computing x^n

Computing x^n

This algorithm takes $\Theta(n)$ time. A better approach is to employ the "*repeated squaring*" trick. Consider the special case in which n is an integral power of 2 (that is, in which n equals 2^k for some integer k).

Computing x^n

while

Computing x^n

The following algorithm computes x^n.

```
power := x;
for i := 1 to k do
      power := power*power;
```

$i=1$ x^1

$i=2$ x^{2^1}

$x^4 (= 2^2)$

$i=3 = x^8$

$i=k - x^{2^k}$

while (k != n)

power = powe * power

k = k+k; - 2, 4, 8, 16

}

Logarithmic Algorithm

$x^{2^k} = x^n$

$2^k = n$

$k = \log_2 n$

Time complexity for Computing x^n

The following algorithm computes x^n.

```
power := x;
for i := 1 to k do
     power := power²;
```

$\Theta(\log n)$

Time Complexity

```
int exp(int X,  int Y)  {
          int res = 1, a = X,  b  = Y;
          while ( b != 0  ){
               if ( b%2 == 0) {
                    a = a*a; b  = b/2;
               }
               else {
                    res = res*a; b  = b-1; }
               }
               return res;
          }
```

GATE | GATE-CS-~~2014~~-(Set-2) | Question 45

2016

$\boxed{X^Y}$

The following function computes XY for positive integers X and Y.

$\boxed{2^k}$ $\Theta(\log_2 y)$

```
int exp (int X, int Y) {
    int res =1, a = X, b = Y;
   while (b != 0) {
        if (b % 2 == 0) {a = a * a; b = b/2; }
        else {res = res * a; b = b - 1; }
    }
    return res;
}
```

Completly under

H.W quection

Which one of the following conditions is TRUE before every iteration of the loop?

a) $X^Y = a^b$

b) $(res*a)^Y = (res*X)^b$

c) $X^Y = res*a^b$

d) $X^Y = (res*a)^b$

$n^2 < n^{2-.001}$

n^2 is $O(n^{\log_b a - \epsilon})$

$\boxed{\epsilon}$

Equate time with complexity

30 sec - Solve Input Size 64

$n\log n$ -

$64 \cdot \log 64 = 30$ sec $\propto$ wrong

Some Constant

$\Theta(n\log n) = k.n\log n$

$k.n\log n = 30$

$k.64.\log 64 = 30 \Rightarrow \boxed{k = \frac{30}{64 \times 6}}$

What Size of problem 360 sec

(I) $256 \times \log 256 \times \frac{30}{64 \times 6}$

$\frac{2^8 \times 2^3 \times 3\cancel{0}\,5}{2^6 \times \cancel{6}}$

$= 32$ sec $\times 5 = 160$

II $512 \times \log 512 \times \frac{30}{64 \times 6}$

$= \frac{2^9 \times 9 \times 5}{2^6} = 8 \times 9 \times 5$

$= 72 \times 5$

$= 360$

512

GATE | GATE-CS-2014-(Set-2) | Question 45

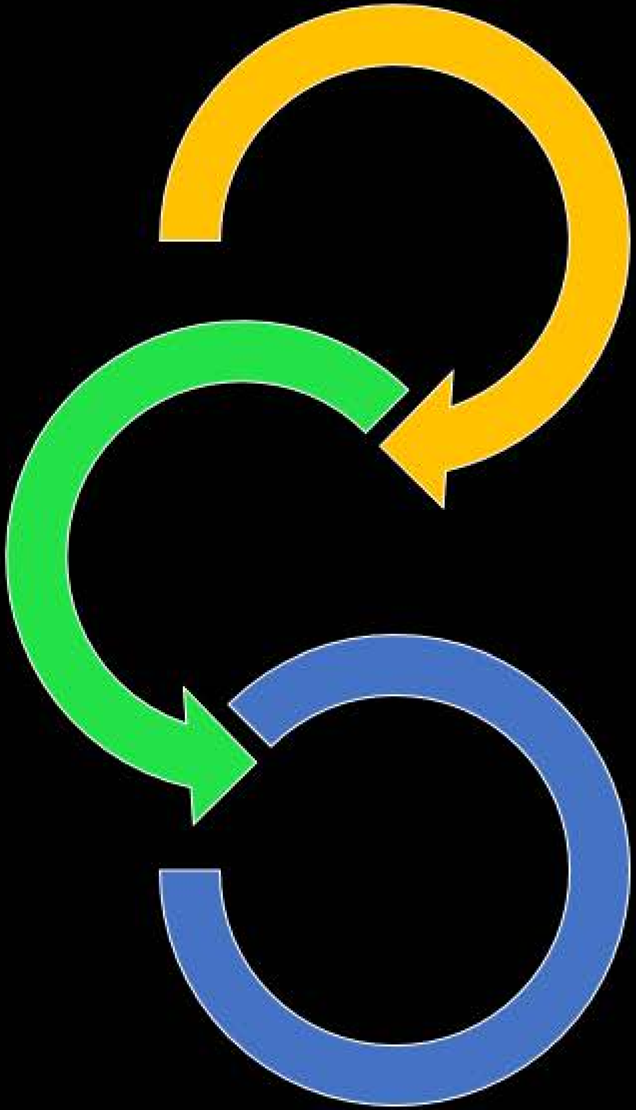
Which one of the following conditions is TRUE before every iteration of the loop?

a) $X^Y = a^b$

b) $(res * a)^Y = (res * X)^b$

c) $X^Y = res * a^b$

d) $X^Y = (res * a)^b$



Divide & Conquer

Design – D&C
– Greedy
– Dynamic
– Graph/Heap
– Sorting – }

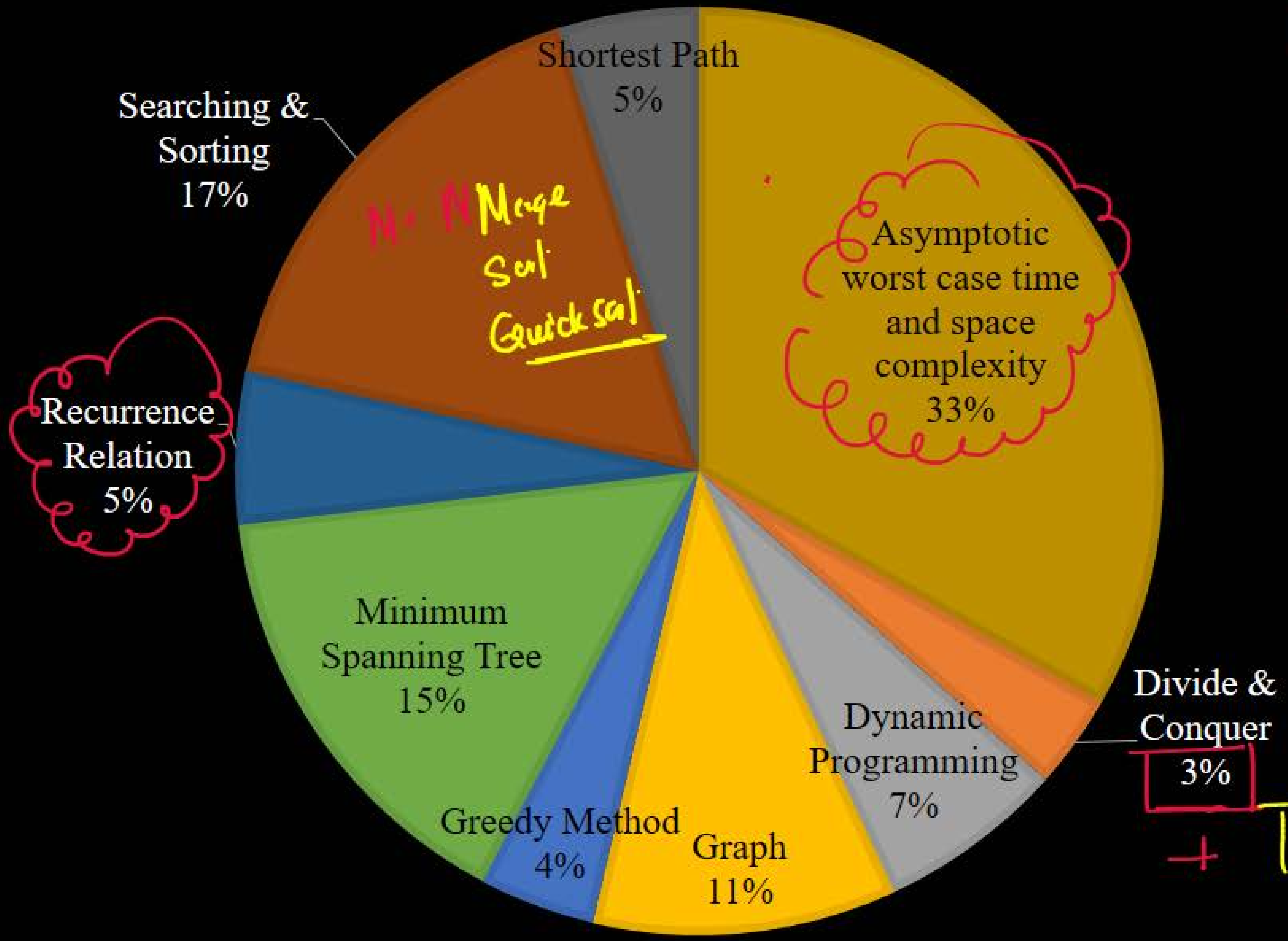
Workbook/Text book

Sorting in Divide & Conquer

Master Method
+
understandin Algorithm

1. Max-Min problem
2. Quick Sort Algorithm
3. Merge Sort
4. Binary Search
5. Stressen's Matrix Multiplication

Divide & Conquer Weightage Analysis



Divide & Conquer ⇒ Master Method

Greedy – Enjoy ⇒

Dynamic programming – interesting + Lengthy ⇐

Sorting – Smooth

Graph – Moderate

+ 10–12%

Divide & Conquer

power(x, n)

power(x, n/2) * power(x, n/2)

each of size n/2

2 subproblem each of size n/2

(x, 1)

return x

- Divide & Conquer : take a problem of size (n) divides it into Subproblem (A smaller instance of problem).
- We Solve the Subproblem using same divide & conquer strategy.
- We keep on dividing in Subproblem until it becomes a smaller instance which can be solved in constant time
- We must find a way to combine Solution of subproblem.

Divide & Conquer

Given a function to compute on n inputs the *divide-and-conquer* strategy suggests splitting the inputs into k distinct subsets, $1 < k \leq n$, yielding k subproblems. These subproblems must be solved, and then a method must be found *to combine subsolutions* into a solution of the whole.

Divide & Conquer

- If the sub-problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied.
- Often the subproblems resulting from a divide-and-conquer design are of the *same* type as the original problem.

Divide & Conquer

- For those cases the application of the divide-and-conquer principle is naturally expressed by a *recursive algorithm*.
- Now smaller and smaller subproblems of the same kind are generated until eventually subproblems that are small enough to be solved without splitting are produced.

same type

Subproblem

power(x, n)

powe(x, n/2)

Divide & Conquer Algorithm

Control Abstraction of Divide & Conquer

```
Algorithm D&C ( P ) {
    if small (P)
        return S(P)

    else {
        divide P in k sub problem. P1 , P2 . . . PK
        Apply D&C to each problem ( D&C(P1) , D&C(P2) . . D&C (Pk)
        return ( combining Solution ( D&C(P1) , D&C(P2) . . D&C(Pk)
    }
}
```

Divide & Conquer Algorithm

```
Algorithm DAndC(P){
 if Small(P)
 then return S(P);
 else{
     divide P into smaller instances P_1,P_2,……,P_k, k ≥ 1;
     Apply DAndC to each of these subproblems;
     return Combine(DAndC(P_1),DAndC(P_2),...,DAndC(P_k));
     }
}
```

Complexity of Divide & Conquer

$$T(n) = \begin{cases} aT(\underline{n/b}) + f(n) & n > 1 \\ \text{Constant time} & n = 1 \end{cases}$$

there are a ~~Subto~~ Sub problem each of size $\underline{n/b}$

at least 1 subprouble – $\underline{a \geq 1}$

$\underline{b > 1}$, a, b are constant.

$f(n)$ is monotonically

funchm

Complexity of Divide & Conquer

- $$T(n) = \begin{cases} 1 & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$
- where a and b are known constants. We assume that T(l) is known and n is a power of b (i.e., $n = b^k$).

One of the methods for solving any such recurrence relation is called the substitution method.

Master Method

Master Method

- The master method provides a “cookbook” method for solving recurrences of the form
- $T(n) = aT(n/b) + f(n)$,

$a \geq 1$, $b > 1$

$f(n)$ - positive function

a :

b

$n^{\log_b a}$

$f(n)$

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

(I) $f(n)$ is $O(n^{\log_b a - \epsilon})$

strictly lower

n^2 $O(n^{2-.0001})$

Subsitt

$3T(n/2)$

3^k

$3^k T(n/2^k)$

$n = 2^k$

$k = \log_2 n$

$3^{\log_2 n} = n^{\log_2 3}$

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

$1/2$ sec

Compare the function -

(Asymptotically) - $n^{\log_b a}$ $f(n)$

Base condition

Compare them

$$\begin{cases} f(n) \text{ smaller function } n^{\log_b a} \\ f(n) \text{ is equal function } n^{\log_b a} \\ f(n) \text{ greater function } n^{\log_b a} \end{cases}$$

ACE Live Class 1

Bound

or

* Series that we get bigger

* Equal function

Base condition

$$n^{\log_4 3} + 4n\left[1 - \frac{3^k}{4^k}\right]$$

$$n^{\log_4 3} + 4n\left[1 - \frac{n^{\log_4 3}}{n}\right]$$

$$n^{\log_4 3} + 4n - 4 * n^{\log_4 3}$$

$4n - n^{\log_4 3}$ $\Theta(n)$ ← Bound

IX $T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$ —— (I)

$T(n/2) = 2T(n/4) + \sqrt{\frac{n}{2}}$ —— (II)

put (II) in (I)

$$T(n) = 2\left[2T\left(n/2^2\right) + \frac{\sqrt{n}}{\sqrt{2}}\right] + \sqrt{n}$$

$$= 2^2 T\left(n/2^2\right) + \frac{2\sqrt{n}}{\sqrt{2}} + \sqrt{n}$$

$$T(n/2^2) = 2T\left(n/2^3\right) + \sqrt{n/2^2}$$

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

T

$T(n) = 3T(n/4) + n$

$a = 3$

$b = 4$

$f(n) = n$

$n^{\log_4 3}$

Series form

Base condition

$$3^k T\left(n/4^k\right) = 3^k T(1)$$

$n = 4^k \Rightarrow k = \log_4 n$

$$3^k = 3^k = 3^{\log_4 n} = n^{\log_4 3}$$

Master Method

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. To use the master method, you will need to memorize three cases.

The Master Theorem

We interpret n/b to mean either $T(\lfloor n/b \rfloor)$ *or* $T(\lceil n/b \rceil)$ Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $\boxed{T(n) = \Theta(n^{\log_b a}).}$

Not $\forall$

very small fraction

f(n) is lesser function

$n^{\log_b a} - \varepsilon$

The Master Theorem

We interpret n/b to mean either $T(\lfloor n/b \rfloor)$ *or* $T(\lceil n/b \rceil)$ Then $T(n)$ has

$n^{.0001}$ the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant ε > 0, then

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

Example: $T(n) = 4.T(n/2) + n$ — $\Theta(n^2)$

$n^{\log_2 4} = n^2 \quad n$

Answer $a - 4$

$b - 2$

$n^{\log_b a} - n^{\log_2 4} = n^2$

$f(n) - n$

$T(n) = (n^{\log_2 4})$

$= n^2$

$f(n) - n$

$n^{\log_b a} = n^2$

$f(n)$ is $O(n^{2 - .001})$

ε

$n < n^{1.999}$

The Master Theorem Case-I

Example: $T(n) = 4.T\left(n/2\right) + n$

$\Theta(n)$

power(α in)

$T(n) = 2T(n/2) + 1$ ← Is case I is applicable?

$a = 2$

$b = 2$

Compare 1 (constant function

& f(n)

Bigger function

$n^{\log_b a} = n^{\log_2 2} = n$

$f(n) = n$

$f(n)$ is $O(n^{1-.001})$

Example: $T(n) = 4.T(n/2) + n$ — (I)

$T(n/2) = 4T(n/2^2) + n/2$ — (II)

put (II) in (I)

$T(n) = 4\left[4T(n/2^2) + n/2\right] + n$

$= \underline{4^2 T(n/2^2)} + 4 * \frac{n}{2} + n$

$T(n/2^2) = 4T(n/2^3) + n/2^2$

$4^2\left[4T(n/2^2) + n/2^2\right] + 4 \times \frac{n}{2} + n$

$4^3 T(n/2^3) + 4^2 \cdot \frac{n}{2^2} + 4 * \frac{n}{2} + n$

k^{th} ter

$4^k \underline{T(n/2^k)} + 4^{k-1} \frac{\cdot n}{2^{k-1}} + \cdots + 4^2 \frac{\cdot n}{2^2} + \frac{4n}{2} + n$

Reduce to base condition

$n/2^k = 1 \Rightarrow \underline{n = 2^k}$

$4^k = (2^2)^k = (2^k)^2 = \underline{n^2}$

$\underline{n^2 T(1)} + n + 2n + 4n + \ldots 2^{k-1} n$

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

$n^2 + n\,\underline{(2^k - 1)}$

$n^2 + n(n-1)^{2-1}$

$= n^2 + n^2 - n = \underline{2n^2 - n}$ $-\Theta(n^2)$

$f(n)$ is $O(n^{\log_b a - \epsilon})$

$n^2 \le n^{2-.0001}$

The Master Theorem- Case-I

For example the equation $T(n) = 2T(n/2) + 1$ falls under the category of case 1 and we can clearly see from it's tree below that at each level the children nodes perform twice as much work as the parent node.

(I) $T(n) = 4T(n/2) + n^2$ — $n^{\log_2 4}$, n^2, $n^{2.001}$

(II) $T(n) = 9T(n/3) + n$ — n^2, $n^{\log_3 9}$, $\theta(n^2)$

(III) $T(n) = 16T(n/4) + n^3$

(IV) $T(n) = 7T(n/3) + n$ — n^2, n^3, n, n, $n^{\log_3 7}$

(V) $T(n) = 5T(n/2) + n^2$ — $\theta(n^{2.32})$

$2.32 = n^{\log_2 5}$, n^2

$n^{2.32}$

$\frac{\log 7}{\log 2}$

(2, 4, 5) apply Master Method case - I