DAA

Asymptotic Notations

1) Big Oh $O(n)$
2) Omega (Ω)
3) Theta (Θ)
4) Small oh (o)
5) Small omega (ω)

Ex: $f(n)$ ↓ Represent time complexity (or) Over stepcount function

$g(n)$ ↓ One which we are going to compare.

~~1)~~ ~~$f(n) = O(g(n))$~~ iff ~~$f(n) \leq c \cdot g(n)$~~

~~$c > 0$, $n \geq n_1 \geq 1$~~

Example: Sum of Two numbers

①
```
int sum(a,b){
    return a+b;      ---- step    T(n) = O(1)
}
```

Sum of elements in an Array;

②
```
int sum(a,n){                          cost    frequency
    total = 0                           1        1
    for(i=0 to i<n ; i++)               1        n+1
    (n+1) checking
    {  total = total + a[i];            1        n
    }                                   1        n
    return total;                       1        1
}
                                                 3n+4
```

1) $f(n) = O\,g(n)$ iff $f(n)$ is lesser than or equal to $g(n)$

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

$c > 0$, $n \geq n_0 \geq 1$

$3n+4 \leq c.g(n)$ $\quad g(n) = n$

$3n+4 \leq c.n$ $\quad c = 1$

$3n+4 \leq n$ ✗

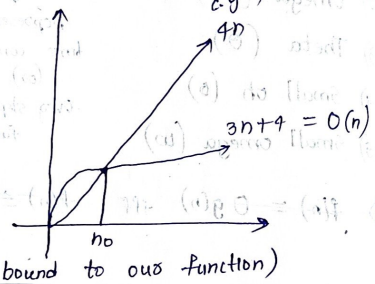
$3n+4 \leq 3n$ ✗

$3n+4 \leq 4n$ ✓

$4 \leq n$

$n_0 = 4$

Graph



(Upper bound to our function)

2) Omega (Ω) $f(n)$ greater than or equal $g(n)$

$f(n) = \Omega\, g(n)$

$f(n) \geq c.g(n)$, $c > 0$, $n \geq n_0 \geq 1$

$3n+4 \geq c.n$

Assume that $c = 1$

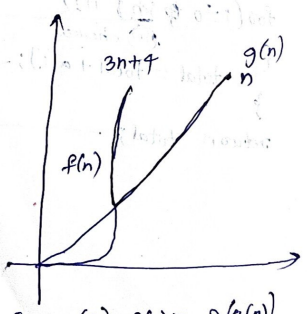
$3n+4 \geq n$

$2n+4 \geq 0$

$n+2 \geq 0$

$n \geq -2$ ✗ $\quad \geq 1$

Graph



Omega (Ω) - $f(n) = \Omega(g(n))$

$f(n) = 3n+4$, $g(n) = n$

(Lower bound to our function)

3) The

• c_1

• $3n$

4) Small

$f(n) =$

$f($

c

f

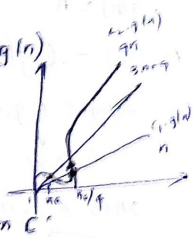
3) Theta (Θ)

$f(n) = \Theta(g(n))$ iff

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$f(n) = \Theta g(n)$

$c_1, c_2 > 0$, $n \geq n_0 \geq 1$



- $c_1 \cdot n \leq 3n+4$

  $n \leq 3n+4$

  $n \geq 1$

  $n_0 = 1$

- $3n+4 \leq c_2 \cdot n$

  $3n+4 \leq 4n$

  $4 \leq n$

  $n_0 = 4$

Lowest bound taken c value from omega ~~function~~ notation.

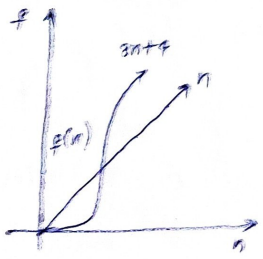
Upper bound taken c value from BigO notation.

4) Small oh (o)

$f(n) = o(g(n))$ iff

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

$c > 0$, $n \geq n_0 \geq 1$



5) Small omega (ω)

$f(n) = \omega(g(n))$

$f(n) \geq c \cdot g(n)$

$c > 0$, $n \geq n_0 \geq 1$

$f(n)$ greater strictly $g(n)$

Q: $f(n) = 2n+5$

$g(n) = n$

1) $f(n) = O(g(n))$

$f(n) \le c \cdot g(n)$

$\exists c > 0,\ n \ge n_0 \ge 1$

$2n+5 \le c \cdot n$

$c = 1$

$2n+5 \le n$ ✗

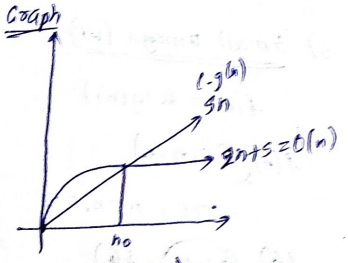
$2n+5 \le 2n$ ✗

$2n+5 \le 3n$ ✓

$5 \le n$

$n_0 = 5$

Graph



2) $f(n) = \Omega(g(n))$

$f(n) \ge c \cdot g(n)$

$c > 0,\ n \ge n_0 \ge 1$

$2n+5 \ge c \cdot n$

$2n+5 \ge n$

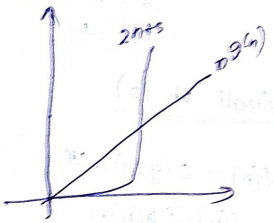
$n+5 \ge 0$

$n \ge -5$

$n \ge 1$

$n_0 = 1$

Graph



3) $f(n) = \Theta(g(n))$

$c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$

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

  $n \le 2n+5$

  $n \ge 1$

  $n_0 = 1$

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

  $2n+5 \le 3n$

  $n \ge 5$

  $n = 5$

Time

1) Fibo

frb(

Iterative

$n^2 = O($

2) Add

Add

based on

T(n)

3mn

3mn

Time Complexicity Analysis :

1) Fibonacci Series

```
fib(n) {
    fib1 = 0
    fib2 = 1
    for (i = 2 to n) do {
        fib3 = fib1 + fib2;
        fib1 = fib2;
        fib2 = fib3;
    }
}
```

Iteration part $n^2 = O(n)$

Cost of operation	Frequency
1	1
1	1
1	1
1	n+1-2
1	n-2
1	n-2
1	n-2
1	n-2
	5n-6

$T(n) = 5n-6 = O(n)$

2) Addition of matrices

```
Add(A, B, C, m, n) {
    for (i = 0 to m) {          --> m times
        for (j = 0 to n) {      --> n times
            C(i,j) = A(i,j) + B(i,j);
        }
    }
}
```

Based on loops

$T(n) = O(mn)$ — i: m times, j: n times

Cost	Frequency
1	1
1	m+1
1	m
1	m
1	m(n+1)
1	mn
1	mn
	3mn+4m+2

$T(n) = 3mn + 4m + 2$

$= O(mn)$

$3mn + 4m + 2 \le g(n)$ → To prove

→ Assume $g(n) = mn$

$3mn + 4m + 2 \le C\,mn$

③ Linear Search:

```
Linear search (A, n, key){                cost    frequency
    for (int i=0 to n){                    1        1
        if (key == A[i]){                  1        n+1
            return i;                      1        n
        }                                  1        n
    }                                                3n+2
    return -1;
}
```

Three cases:
- Best case ⇒ $T(n) = O(1)$
- Average case ⇒ $T(n) = O(n)$
- Worst case ⇒ $T(n) = O(n)$

Middle element → $n/2 + c$ → some constant

$3n/2 + 2 \leq c \cdot g(n)$

$g(n) = n^1$

① last element is key ② key not present

④ Binary Search

```
Binary search (A, s, e, key) {
    if (s > e) return -1;
    mid = (s+e)/2;
    if (key == A[mid]) return mid;
    else if (key < A[mid])
        return Binary search (A, s, mid-1, key);
    return Binary search (A, mid+1, e, key)
}
```

$$T(n) = \begin{cases} 2 & ; n=0 \\ 4 & ; n=1 \\ 4 + T(n/2) & ; n>1 \end{cases}$$

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

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

$= T(n/8) + 4 + 4 + 4$

$= T(n/10) + 4 + 4 + 4$

k steps

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

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

$n = 2^k$

$k = \log n$

$T(n) = T(1) + \log n(4)$

$= 4 \log n + 4$

$= O(\log n)$

worst case.

* Best case

Middle element

$T(n) = O(1)$

* Average case

$T(n) = O(\log n)$

Ascending order of Time complexity

$1, n, n^2, \log n, n^n, 2^n, n^n$ → polynomial; 2^n → exponential.

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

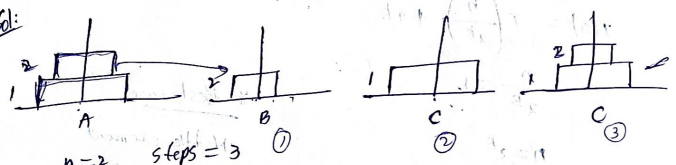
Then $f(n), g(n)$ → Arrangement

$1, \log n, n, n^2, n^3, 2^n, n^n$

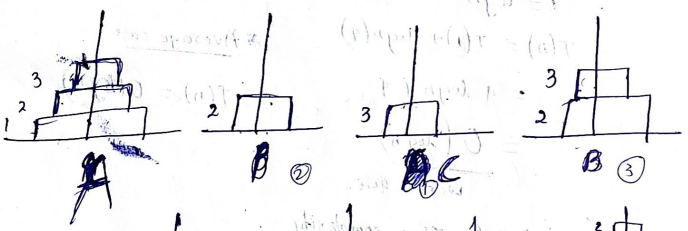
Towers of Honoi:

Q: Move all the D's from set A to C using B. The conditions are 1) You can move at a time only one D.
2) Always small-D's should be top on big D's.

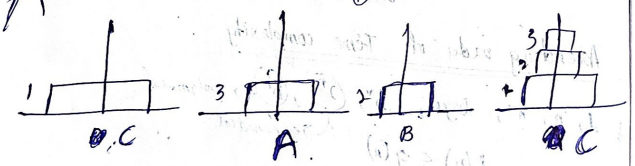
Sol:



n=2 steps = 3 ① ② ③

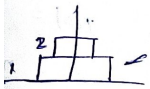


① ② ③

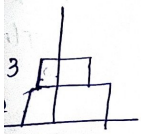


n=3 Steps = 7

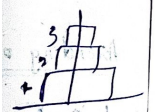
B. The conditio
D.
on Big O's.



C ③



B ③



C

03 March April 2023

TOH (n , A , C , B) { (Source, Destination, Help)

T(n) ↓

If (n == 1) → Base condition.

move (A, C)

If (n > 1) {

T(n-1) ← TOH (n-1, A, B, C)

1 ← TOH (1, A, C, B)

T(n-1) ← TOH (n-1, B, C, A)

}

$$T(n) = \begin{cases} 1 & n=1 \\ 2T(n-1)+1 & n>1 \end{cases}$$

By substitution method.

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

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

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

after for k steps

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

Now make $n-k = 1$

$k = n-1$

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

$= 2^{n-1} + 2^{n-1} + 1$

$= 2^n + 1$

$\boxed{T(n) = O(2^n)}$

Masters Theorem: • For dividing recurrence relation.

• Another method for time complexity ~~For~~ ~~TOPM~~

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta\left(n^k \log^p n\right)$$

Ex: Binary search $T(n) = T(n/2) + 1$ (1 → constant)

There will be three conditions

if $a \geq 1$, $b \geq 1$, $k \geq 0$

① $a > b^k$ $\quad T(n) = \Theta\left(n^{\log_b a}\right)$

② $a = b^k$

ⓐ $p > -1$; $T(n) = \Theta\left(n^{\log_b a} \log^{p+1} n\right)$

ⓑ $p = -1$; $T(n) = \Theta\left(n^{\log_b a} \log \log n\right)$

ⓒ $p < -1$; $T(n) = \Theta\left(n^{\log_b a}\right)$

③ $a < b^k$

a) $p >= 0$; $T(n) = \Theta\left(n^k \log^p n\right)$

b) $p < 0$; $T(n) = \Theta\left(n^k\right)$

Ex: $T(n) = 1.T(n/2) + 2 \cdot n^k \log^p n$

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

$b^k \cdot 2^0 = 1$

Find the

Ex(1) $T(n) =$

Sl $a = 2$

If

Now

② $T(n) =$

Sol: $a =$

If

rence relation.

Ex:
Binary search
$T(n) = T(n/2) + c$
constant

$\log^{p+1} n)$

$^a \log\log n)$

$^a)$

$\log^p n)$

Find the time complexity for following.

Ex: ① $T(n) = 2T\left(\frac{n}{2}\right) + n$

Sol: $a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

It is second case $a = b^k \Rightarrow 2 = 2^1$

Now $p = 0$ means $p > -1$

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

$$= \theta\left(n^{\log_2 2} \log^{0+1} n\right)$$

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

② $T(n) = 4T\left(\frac{n}{3}\right) + n \log n$

Sol: $a = 4 \quad b = 3 \quad k = 1 \quad p = 1$

It is first case $a > b^k \Rightarrow 4 > 3^1$

So $T(n) = \theta\left(n^{\log_b a}\right)$

$$= \theta\left(n^{\log_3 4}\right)$$

$$\boxed{T(n) = \theta\, n^{\log_3 4}}$$

Master Theorem :

- For decreasing recurrence relation

$$T(n) = a\,T(n-b) + f(n)$$

$$f(n) = O(n^k), \quad a>0,\ b>0,\ k\geq 0$$

i) $a<1$; $T(n) = f(n)$

ii) $a=1$; $T(n) = O(n \cdot f(n))$

iii) $a>1$; $T(n) = O\left(a^{\frac{n}{b}} \cdot f(n)\right)$

Ex: ① TOH

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

$a=2$ $b=1$ $f(n)=1$ constant function

$1 = n^0$

$k=0$

3rd Condition $a>1$

$$T(n) = O\left(a^{\frac{n}{b}} \cdot f(n)\right)$$

$$= O\left(2^{\frac{n}{1}} \cdot 1\right)$$

$$= O(2^n)$$

② $T(n) = 2T(n-1) + \log n$

$a>1$ $T(n) = O\left(a^{\frac{n}{b}} \cdot f(n)\right)$

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

③ $T(n) = T(n-1) + n$

$a=1$ 2nd condition $T(n) = O(n \cdot f(n))$

$= O(n \cdot n)$

$= O(n^2)$

Divide

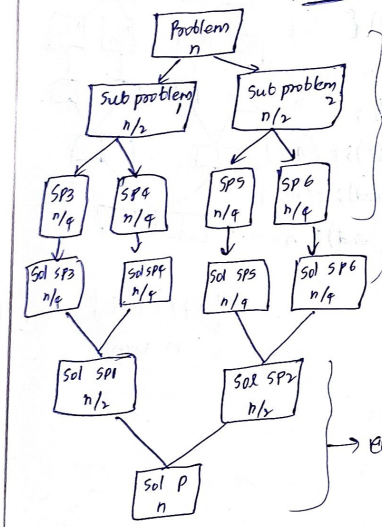
Applicat

1) Bina
2) Mer
3) Quic
4) Tree
5) Sto

11 April 2023
Tuesday

Divide and conquer:

→ combine



Three steps
1) Divide
2) solution finding
3) Merging / combining.

* We use recursion because we are using same property/logic but size is different

→ Conquer strategy.

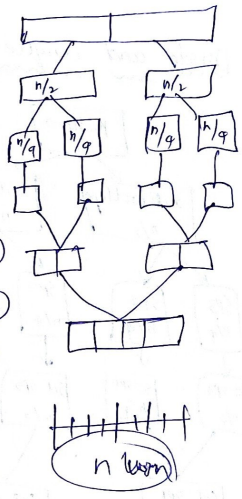
Applications of divide and conquer :

1) Binary search
2) Merge sort
3) Quick sort
4) Tree traversals
5) Strassens matrix multiplication.

Merge sort:

(Pseudocode)

```
mergesort(A, start, end) {
    if (start < end) {
        mid = (start + end)/2;
    mergesort(A, start, mid); T(n/2)
    mergesort(A, mid+1, end); T(n/2)
    merge(A, start, mid, end); (n)
}
merge
```



T(n) =

T

p >

Qu

$$T(n) = \begin{cases} 2T(n/2) + \Theta(n) & , n>1 \\ 1 & , n=1 \end{cases}$$

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

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

$2=2^1$

$p>-1 \ ;\ T(n) = \Theta(n^{\log_2 2^1} \log^{0+1} n)$

$= \Theta(n \log n)$

Quick sort:

10	90	20	5	6	7

p

10 5 20 90 6 7

10 5 6 90 20 7

10 5 6 7 20 90

7 5 6 10 20 90

12 April 2023
Wednesday.

Strassenes Matrix Multiplication:

If size = 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times 2} \qquad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

operators used

$8 - *$
$4 +$
$a * b$

$A \times B$

$$C = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Naive Matrix Multiplication

If size > 2

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]_{q\times q} \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]_{q\times q}$$

$$C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] \qquad C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$

- This algorithm is applicable only for size of $2^k \times 2^k$

2×2, 4×4, 8×8, 16×16 ---

$$T(n) = \underbrace{8}_{\text{multiplication operators}} T\left(\frac{n}{2}\right) + \theta(n)$$

$a = 8$, $b = 2$, $k = 0$

$a > b^k$

$$T(n) = \theta(n^{\log_b a})$$
$$= \theta(n^{\log_2 8})$$
$$= \theta(n^3)$$

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
        for(k=0; k<n; k++)
```

3 loops

n^3

Strassenes Matrix Multiplication:

12 April 2023
Wednesday.

If size = 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times 2} \qquad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

A×B

$$C = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

operations used

8 - * ↕
4 + ↓
a * b +

Naive Matrix Multiplication

If size > 2

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]_{4\times 4} \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]_{4\times 4}$$

$$C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] \qquad C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$

- This algorithm is applicable only for size of $2^k \times 2^k$.

  2×2, 4×4, 8×8, 16×16 ...

$$T(n) = \underset{\text{multiplication operators}}{8}\, T\left(\frac{n}{2}\right) + \theta(n)$$

$a = 8, \; b = 2, \; k = 0$

$a > b^k$

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

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

$$= \theta(n^3)$$

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
        for(k=0; k<n; k++)
```

3 loops

n^3

Strass

A=

A×B

7 Matrices

P

Q
R
S
T
U
V

Strassens Matrix Multiplication:

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]_{4\times4} \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

$$A \times B = C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

7 Matrices

$P = (A_{11} + A_{22}) * (B_{11} + B_{22})$ (Diagonal Adding)

Matrix addition

$Q = (A_{21} + A_{22}) * B_{11}$

$R = A_{11} * (B_{12} - B_{22})$

$S = A_{22} * (B_{21} - B_{11})$

$T = (A_{11} + A_{12}) * B_{22}$

$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$

first column A, first row B

$V = (A_{12} - A_{22}) * (B_{21} + B_{22})$

second column A, second row B

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + V$

- 7 multiplication operators are using.

$B_{11}\ A_{11}\ A_{22}\ B_{22}$ sequence always A first then B

- Row elements addition operation
- Same column elements subtraction operation

$2^k \times 2^k$

Pseudocode:

```
Stressens (A, B, n) {
    C = n x n
    if (n == 1) {
        C[0][0] = A[0][0] + B[0][0];
    }
    else {
        A11, A12, A21, A22 ← A.split();
        B11, B12, B21, B22 ← B.split();
        P = stressens(A11 + A22, B11 + B22, n/2);
        Q = stressens(A21 + A22, B11, n/2);
        R = stressens(A11, B12 - B22, n/2);
        S = stressens(A22, B21 - B11, n/2);
        T = stressens(A11 + A12, B22, n/2);
        U = stressens(A21 - A11, B11 + B12, n/2);
        V = stressens(A12 - A22, B21 + B22, n/2);
        C11 = P + S - T + V;
        C12 = R + T;
        C21 = Q + S
        C22 = P + R - Q + U
        { C ← combine-quadrats(C11, C12, C21, C22);
        }
        return C;
    }
}
```