Date:

ALGORITHMS

① INTRODUCTION & ANALYSIS

Definition: step by step process of solving a computational problem

Program: step by step process of solving a problem.

Algorithm	Program
☆ Design	☆ Implementation
☆ Domain knowledge	☆ programmer
☆ Any Language	☆ Programming Language
☆ Hardware and OS independent	☆ Dependent on hardware OS
☆ Analyze	☆ Testing

Priori Analysis	Posteriori Testing
① Algorithm	① program
② Independent of Language	② Language Dependent
③ Hardware Independent	③ Hardware Dependent
④ Time and space Function	④ watch Time and Bytes

Characteristics of Algorithm :

① **Input**
takes 0 (or) more inputs

② **output**
at least one output must be generated

③ **Definiteness**
every statement shall be unambiguous and must have a single and exact meaning.
$\sqrt{-1} \rightarrow \times$

④ **Finiteness**
algorithm must terminate at some point → it must have a finite number of statements.

⑤ **Effectiveness**
Unnecessary statements must not be written.
Procedure must be effective.

These are the 5 important characteristics of every algorithm.

apsara

Date: __.__.____

How to write an algorithm :

Algorithm swap (a, b)
{ Begin
temp = a ;
a = b ;
b = temp ;
} End

The reader must be able to understand what is written.
no fixed syntax

① Time
time effeciency → must be fast

② Space
space efficiency

[TWO MAJOR FACTORS]

③ Network consumption
↳ how much data transfer is done

④ Power consumption
↳ how much power is consumed

⑤ CPU registers
↳ how many cpu registers are consumed in the memory.

Algorithm swap (a, b)
Begin
temp ← a ⇒ 1
a ← b ⇒ 1
b ← temp ⇒ 1
End

Note : Every simple statement in an algorithm consumes one unit of time

$\underline{3}$

$\boxed{f(n) = 3}$ constant Time
↳ O(1)

$x = (5 * a) + (6 * b) \rightarrow 1$
(generalised)

space : a → 1 b → 1 temp → 1

$\boxed{S(n) = 3}$ (3 words) constant space
↳ O(1)

Frequency count Method

[sum of elements of an array]

Algorithm sum (A, n)
{
s = 0 ; ⟶ 1
for (i=0 ; i< n ; i++) ⟶ 2n+2 ≡ n+1
s = s+ A[i] ; ⟶ n
return s; ⟶ 1
}

	0	1	2	3	4
A	8	3	9	7	2

$i=0$ ✓
$i=1$ ✓
$i=2$ ✓
$i=3$ ✓
$i=4$ ✓
$i=5$ ✓ ✗

condition is checked
$(n+1)$ times

Time function

$f(n) = 1 + (n+1) + \cancel{(n+1)} + (n)$

$f(n) = 2n+3$

$\boxed{O(n)}$

Space complexity

$A \rightarrow n \quad n \rightarrow 1 \quad s \rightarrow 1 \quad i \rightarrow 1$

$s(n) = n+3 \longrightarrow \boxed{O(n)}$

sum of two matrices

Algorithm Add (A, B, n)

{

for ($i=0$; $i<n$; $i++$) $\rightarrow n+1$

{

for ($j=0$; $j<n$; $j++$) $\rightarrow n(n+1)$

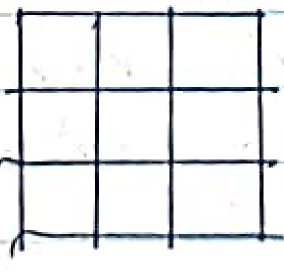
{

$c[i,j] = A[i,j] + B[i,j]$ $\rightarrow n(n)$

}

}

}



Time

$f(n) = 2n^2 + 2n + 1$

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

apsara

Date:

Space : $A \to n^2$ $B \to n^2$ $C \to n^2$

$n \to 1$ $i \to 1$ $j \to 1$

$\Downarrow$

$S(n) = 3n^2 + 3 \to \boxed{O(n^2)}$

Multiplication of two matrices

Algorithm Multiply (A, B, C, n)

{

$n+1 \leftarrow$ for (i = 0; i < n; i++)

{

$(n+1)n \leftarrow$ for (j = 0; j < n; j++)

{

$(n)(n) \leftarrow$ C [i, j] = 0

$(n+1)(n)(n) \leftarrow$ for (k = 0; k < n; k++)

$(n)(n)(n) \leftarrow$ C[i, j] += A[i,k] * B[k,j]

}

}

}

time function

$f(n) = 2n^3 + 3n^2 + 2n + 1$

$\hookrightarrow \boxed{O(n^3)} \to$ time

Space function

$A \to n^2$ $B \to n^2$ $C \to n^2$ $n \to 1, i \to 1, j \to 1, k \to 1$

$S(n) = 3n^2 + 4$ $\boxed{O(n^2)} \to$ space

apsara

Date: __.__.__

Analysis of time complexity

① for (i=0; i<n; i++) → n+1
  stmt; → n   O(n)

② for (i=n; i>0; i--) → n+1
  stmt; → n   O(n)

③ for (i=1; i<n; i=i+2) doesn't affect O(n)
  stmt; → n/2
  f(n) = n/2 → O(n)

④ for (i=0; i<n; i++) → (n+1)
{
  for (j=0; j<n; j++) → n(n+1)
  {
    stmt; → n^2
  } $O(n^2)$
}

⑤ for (i=0; i<n; i++)
{
  for (j=0; j<i; j++)
  { stmt;
  }
}

Date:

$i=0 \rightarrow$ j not executed $\rightarrow$ 0 times

$i=1 \rightarrow j=0$ executed $\rightarrow$ 1 time

$i=2 \rightarrow j=0,1$ executed $\rightarrow$ 2 times

$i=3 \rightarrow j=0,1,2 \rightarrow$ 3 times

$\vdots$

$i=n-1 \rightarrow j=0,1,\ldots n-2 \rightarrow n-1$ times

total

$$\text{total} = 0+1+2+\cdots+(n-1)$$

$$f(n) = \frac{n(n-1)}{2} \rightarrow O(n^2)$$

---

⑥

```
p = 0;
for (i=1; p <= n; i++)
{
    p = p+i;
}
```

i	p
1	$0+1=1$
2	$1+2=3$
3	$1+2+3=6$
4	$1+2+3+4=10$
$\vdots$	
k	$1+2+\cdots+k = \frac{k(k+1)}{2}$

loop repeats k times

apsara

Date: __.__.__

Assume $p > n \rightarrow \frac{k(k+1)}{2} > n$

$k(k+1) > 2n \rightarrow$ stopping condition

$\downarrow$

approx $k^2 > n \rightarrow k > \sqrt{n}$

Time complexity is $O(\sqrt{n})$

(7) for (i=1; i<n; i = i*2)

{

stmt;

}

i

$1 \rightarrow 1\times 2 = 2 \rightarrow 2^2 \rightarrow 2^3 \dots \rightarrow 2^k$

k times

Assume $i \geq n$ $\quad i = 2^k$

$2^k \geq n \rightarrow 2^k = n \rightarrow k = \log_2 n$

Time complexity $O(\log_2 n)$

$i = 1 \times 2 \times 2 \times \dots \times 2 = n$

$2^k = n$

$\hookrightarrow k = \log_2 n$

(Logarithmic complexity)

apsara

Date

eg $n = 8$ $i = \underbrace{1, 2, 4,}_{3\ \text{times}} 8,$

$n = 10$ $i = \underbrace{1, 2, 4, 8,}_{4\ \text{time}} 16$

$\log_2 8 = 3$ $\log_2 10 = 3.2$ 4 times

⇓

clearly $O\lceil \log n \rceil$ ceil function

---

(8) for (i = n; i >= 1; i = i/2)
{
stmt;
}

$i \to n, \frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3} \dots \frac{n}{2^k}$

Assume $i < 1 \to \frac{n}{2^k} < 1 \to 2^k > n$

$2^k = n \to k = \log_2 n \to O(\log_2 n)$

---

(9) for (i = 0; i * i < n; i++)
{
stmt;
}

Assuming $k^2 > n \to k = \sqrt{n}$ $O(\sqrt{n})$

apsara

Date: __.__.____

(10)

```
for (i=0; i<n; i++)
{
    stmt; → n
}
for (j=0; j<n; j++)
{
    stmt; → n
}
```

Independent loops

$f(n) = 2n$

$O(n)$

(11)

```
p = 0
for (i=1; i<n; i=i*2)
{
    p++; → p = log2 n
}
for (j=1; j<p; j=j*2)
{
    stmt; → log2 p
}
```

$O(\log_2(\log_2 n))$ → time complexity

(12)

```
for (i=0; i<n; i++) → n+1
{
    for (j=1; j<n; j=j*2) → n log2 n
    {
        stmt; → n log2 n
    }
}
```

$f(n) = 2n\log_2 n$

$\boxed{O(n \log_2 n)}$

© Prajwal Sundar

Summary

for (i = 0; i < n; i++) → $O(n)$

for (i = 0; i < n ; i = i+2) → $O(n)$

for (i = n; i > 1; i--) → $O(n)$

for (i = 1; i < n; i = i*2) → $O(\log_2 n)$

for (i = 1; i < n; i = i*3) → $O(\log_3 n)$

for (i = n; i > 1; i = i/2) → $O(\log_2 n)$

<u>Analysis of if & while</u>

for i = 1 to n do step 2
{
  stmt ;
}

step 2 ↓ 1, 3, 5 . .

step 1   for → n+1

stmt → n

while (condn)
{
  stmt ; → executes as long as condition is true
}

do

{ stmt; → executed at least once even

} while (condn); if condition is initially false

repeat

{

stmt; → executes as long as

} condition is $\boxed{\text{FALSE}}$

until (condition);

---

① $i = 0;$ → 1

while $(i<n)$ → $n+1$ $f(n) = 3n+2$

{

stmt; → n ⟹

$i++;$ → n $O(n)$

}

(for ①)

for $(i=0; i<n; i++)$ [1, $n+1$, n] $f(n) = 3n+2$

{

stmt; → n ⟹ $O(n)$

}

---

② $a=1;$

while $(a<b)$ $a \to 1,\ 1\times 2 = 2,\ 2\times 2 = 2^2.$

{ stmt; $2^2 \times 2 = 2^3 \cdots 2^{k-1}\times 2 = 2^k$

$a = a*2;$

} $2^k \to a \geq b$ terminate

apsara

Date: ____

Terminates @ $a = 2^K = b$

$2^K = b \qquad K = \log_2 b$

$O(\log_2 b) = O(\log_2 n)$

$b = n$

```
for (i=1; i<n; i = i*2)
{   stmt;
}
```

$\hookrightarrow O(\log_2 n)$

---

③
```
i = n ;
while (i>1)
{       stmt;
        i = i/2;
}
```

```
for (i=n; i>1; i=i/2)
{     stmt;
}
```

$\hookrightarrow O(\log_2 n)$

---

④
```
i=1;
k=1;
while (k<n)
{
    stmt;
    k = k+i;
    i++;
}
```

i	K
$i=1$	$K=1$
2	$1+1=2$
3	$2+2=4$
4	$2+2+3$
5	$2+2+3+4= \dots$
⋮	
K	$\underbrace{2+2+3 \dots + K}_{\frac{K(K+1)}{2}+1}$

apsara

Date: __.__.__

Terminating condition

$k \geq n \equiv k = n$

$\frac{k(k+1)}{2} + 1 = n$ approximately

$k^2 = n, \; k = \sqrt{n}$

$O(\sqrt{n})$

```
for (k = 1, i = 1; k < n; i++)
{
    stmt;
    k = k + i;
}
```

$\rightarrow O(\sqrt{n})$

---

5) Finding GCD of (m, n)

```
while (m != n)
{
    if (m > n)   m = m - n;
    else   n = n - m;
}
```

m	n		m	n		m	n
6	3		5	5		16	2
3	3					14	2
						12	2
						10	2
						8	2
						6	2
						4	2
						2	2

1 time

0 times

7 times

Approx $n/2$ times

apsara

Date:

min O(1) , max O(n)

```
for ( ; m!=n; )
{    if (m>n)    m = m-n;
     else        m = n-m;
}
```

---

(6)

```
Algorithm Test (n)
{
    if (n<5)
        printf ("%d", n); → 1
    else
        for (i=0; i<n; i++)
            printf ("%d", i);    n times
}
```

if → O(1) time (best case)

else → O(n) time (worst case)

```
Algorithm test (n)
{
    if (n<5)
        for (i=0; i<n; i++)
            printf ("%d", i); → n
```

O(n)

apsara

Date: __.__.__

Types of Time Functions

$O(1) \rightarrow$ constant

$O(\log n) \rightarrow$ logarithmic

$O(n) \rightarrow$ linear

$O(n^2) \rightarrow$ quadratic

$O(n^3) \rightarrow$ cubic

$O(2^n) \rightarrow$ exponential

$O(3^n)$, $O(n^n)$

$\left\{ \begin{array}{l} f(n) = 2 \\ f(n) = 5 \\ f(n) = 5000 \end{array} \right.$

$\left\{ \begin{array}{l} f(n) = 2n+3 \\ f(n) = 500n+700 \\ f(n) = n/5000 + 6 \end{array} \right.$

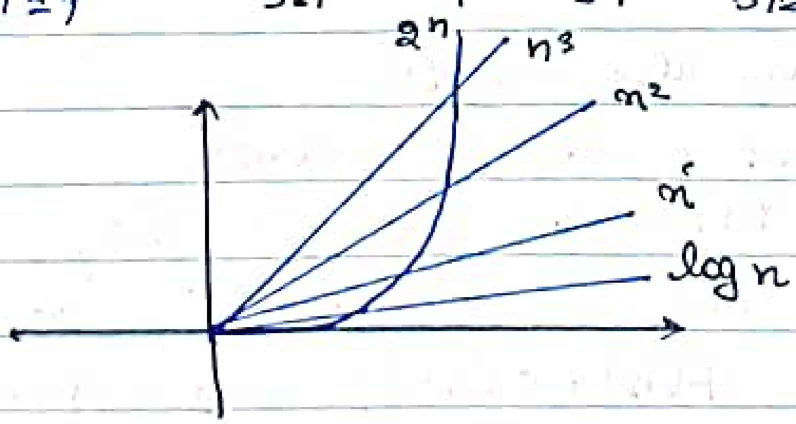
Comparison of classes of Functions

$$\boxed{1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n}$$

Eg	$\log n$	n	n^2	2^n
$n=1$	0	1	1	2
$n=2$	1 <	2 <	4 <	4
$n=4$	2	4	16	16
$n=8$	3	8	64	256
$n=9$	3.1	9	81	512

$n^{100} < 2^n$

$n^k < 2^n$.



(Approximate graph)

Asymptotic Notations

O Big-Oh → Upper Bound
Ω Big-omega → Lower Bound
θ Theta → Average Bound

Big-Oh

↓

The function $f(n) = O(g(n))$ iff $\exists$ +ve constants c and n_0 such that
$$f(n) \leq c\,g(n) \ \forall \ n \geq n_0$$

eg $f(n) = \underbrace{2n+3}_{f(n)} \leq \underbrace{10}_{c}\,\underbrace{n}_{g(n)} \ \forall \ n \geq \underbrace{1}_{n_0}$

↓

$f(n) = O(n)$

(or)

$$2n+3 \leq 2n+3n$$
$$2n+3 \leq 5n \rightarrow n \geq 1$$

(n_0 → positive starting value)

we can also write

$$2n+3 \leq 2n^2+3n^2$$
$$\underbrace{2n+3}_{f(n)} \leq \underbrace{5}_{c}\,\underbrace{n^2}_{g(n)} \ \forall \ n \geq \underbrace{1}_{n_0}$$

$f(n) = O(n^2)$ also accepted

For given $f(n) = 2n+3$

$$\underbrace{1 < \log n < \sqrt{n} <}_{\text{lower bound}} \underbrace{\boxed{n}}_{\text{average bound}} \underbrace{< n\log n < n^2 < n^3 < \dots 2^n < 3^n < \dots < n^n}_{\text{upper bound}}$$

$f(n) = O(n)$ ✓, $f(n) = O(n^2)$ ✓, $f(n) = O(\log n)$ ✗

But while writing Big-Oh, try to express as a function closest to average bound.
$O(n)$ is best for $f(n) = 2n+3$

omega

The function $f(n) = \Omega(g(n))$ iff $\exists$ +ve constants c and n_0 such that
$f(n) \geq c\, g(n) \;\forall\; n \geq n_0$

eg $\underbrace{f(n) = 2n+3}_{f(n)} \geq \underbrace{1}_{c}\underbrace{n}_{g(n)} \;\forall\; n \geq \underbrace{1}_{n_0}$

so $f(n) = \Omega(n)$ → nearest one is useful

$f(n) = \Omega(\log n)$ ✓ $f(n) = \Omega(\sqrt{n})$ ✓

$f(n) = \Omega(n^2)$ ✗

apsara

Date: __.__.__

Theta

↓

The function $f(n) = \theta(g(n))$ iff $\exists$ +ve constants c_1, c_2 and n_0 such that

$$c_1\, g(n) \le f(n) \le c_2\, g(n) \;\forall\; n \ge n_0$$

↓

eg $f(n) : 2n + 3$

↓

$$\underbrace{1}_{c_1}\underbrace{n}_{g(n)} \le \underbrace{2n+3}_{f(n)} \le \underbrace{5}_{c_2}\underbrace{n}_{g(n)} \;\forall\; n \ge \underbrace{1}_{n_0}$$

↓

$f(n) = \theta(n)$ is the only possible solution

☆ $f(n) = 2n^2 + 3n + 4$

$$2n^2 + 3n + 4 \le 2n^2 + 3n^2 + 4n^2$$

$$2n^2 + 3n + 4 \le 9n^2 \;\forall\; n \ge 1$$

↓

$$f(n) = O(n^2)$$

$$2n^2 + 3n + 4 \ge 1\, n^2$$

↳ $f(n) = \Omega(n^2)$

$$1\, n^2 \le 2n^2 + 3n + 4 \le 9n^2 \rightarrow \theta(n^2)$$

apsara

Date: __.__.__

☆ $f(n) = n^2 \log n + n$

$1 \cdot n^2 \log n \le n^2 \log n + n \le 10 n^2 \log n$

$f(n) = O(n^2 \log n) = \Omega(n^2 \log n)$
$= \theta(n^2 \log n)$

☆ $f(n) = n!$

$f(n) = n(n-1)(n-2) \ldots (3)(2)(1)$

$1 \times 1 \times \ldots \times 1 \le n! \le n \times n \ldots \times n$

$1 \le n! \le n^n$

$\Omega(1)$ and $O(n^n)$

θ (average bound) is not possible

For smaller value bounds, n! is closer to left side
For larger value, n! is closer to right side
$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots < 2^n < 3^n < \ldots < n^n$

n! cannot be fixed in any location

Here O and Ω are useful.

apsara

Date: __.__.__

$f(n) = \log n!$

$\log(1 \times 1 \times \cdots \times 1) \leq \log(1 * 2 * \cdots * n) \leq \log(n * n \ldots)$

$\log(1) \leq \log n! \leq \log n^n$

$0 \leq \log n! \leq n \log n$

↳ $\Omega(1)$ and $O(n \log n)$

Properties of Asymptotic notations

General properties :

① If $f(n) = O(g(n))$, then $af(n) = O(g(n))$ also

eg ↳ $f(n) = 2n^2 + 5 = O(n^2)$

$7f(n) = 14n^2 + 35 \overset{also}{=} O(n^2)$

↓

True for $\Theta(g(n))$ and $\Omega(g(n))$ also

② Reflexive property

If $f(n)$ is given, $f(n) = O(f(n))$

↓

$f(n) = n^2$, $f(n) = O(n^2)$ also

③ **Transitive Property**

If $f(n) = O(g(n))$ and $g(n) = O(h(n))$
then $f(n) = O(h(n))$

eg $f(n) = n$, $g(n) = n^2$, $h(n) = n^3$
$n = O(n^2)$, $n^2 = O(n^3)$
so $n = O(n^3)$ θ Ω

④ **Symmetric Property**

If $f(n) = \theta(g(n))$, then $g(n) = \theta(f(n))$
eg $f(n) = n^2$ $g(n) = n^2$
$f(n) = \theta(n^2)$ $g(n) = \theta(n^2)$

⑤ **Transpose symmetric property**

If $f(n) = O(g(n))$, then $g(n) = \Omega(f(n))$
eg $f(n) = n$, $g(n) = n^2$
$n = O(n^2)$ and $n^2 = \Omega(n)$

⑥ If $f(n) = O(g(n))$ and
$f(n) = \Omega(g(n))$ $\downarrow \le f(n) \le \downarrow$
then $f(n) = \theta(g(n))$

apsara

Date:

other properties

① If $f(n) = O(g(n))$ and $d(n) = O(e(n))$ — summation property

$\Downarrow$

then $f(n) + d(n) = ?$

$f(n) = n = O(n)$

$d(n) = n^2 = O(n^2)$

$\Downarrow$

$f(n) + d(n) = n + n^2 = O(n^2)$

$$\boxed{f(n) + d(n) = O[\max[O(f(n)), O(d(n))]]}$$

② If $f(n) = O(g(n))$

$d(n) = O(e(n))$

$\Downarrow$

then $\boxed{f(n) * d(n) = O(g(n) * e(n))}$

$n * n^2 = O(n * n^2) = O(n^3)$

comparision of functions

method-1

n	n^2 < n^3
2	$2^2 = 4 < 2^3 = 8$
3	$3^2 = 9 < 3^3 = 27$
4	$4^2 = 16 < 4^3 = 64$

$n^2 < n^3$ ✓

Page No.

apsara

Date: __.__.____

Method-2 Apply log on both sides

$n^2 \quad n^3 \rightarrow \log n^2$ vs $\log n^3$

$2 \log n < 3 \log n$ obviously

Logarithmic Formulae

$\log(ab) = \log a + \log b$

$\log(a/b) = \log a - \log b$

$\log(a^b) = b \log a$

$a^{\log_c b} = b^{\log_c a}$

$a^b = n \longrightarrow b = \log_a n$

① compare $f(n) = n^2 \log n$ and $g(n) = n(\log n)^{10}$

Applying Log,

$\log f(n) = 2\log n + \log \log n$

$\log g(n) = \log n + 10 \log \log n$

log log is smaller, log dominates

$\log f(n) > \log g(n) \rightarrow f(n) > g(n)$

② compare $f(n) = 3n^{\sqrt{n}}$ and $g(n) = 2^{\sqrt{n} \log n}$

$f(n) = 3n^{\sqrt{n}}$ and $g(n) = n^{\sqrt{n}}$

clearly $f(n) > g(n)$, asymptotically =

apsara

Date:

(3) compare $f(n) = n^{\log n}$ and $g(n) = 2^{\sqrt{n}}$

$\downarrow$

Applying log on both sides

$\log f(n) = (\log n)^2$, $\log g(n) = \sqrt{n}\ \cancel{\log 2}$

$\downarrow$ log again

$\log\log f(n) = 2\log\log n$ — log domination

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

$f(n) < g(n)$

(4) $f(n) = 2^{\log n}$ and $g(n) = n^{\sqrt{n}}$

$\downarrow$

$\log f(n) = \log n$, $\log g(n) = \sqrt{n}\log n$

clearly $f(n) < g(n)$

(5) $f(n) = 2n$ and $g(n) = 3n$

$f(n) < g(n)$ in terms of value

But asymptotically they are equal.

(6) $f(n) = 2^n$ and $g(n) = 2^{2n}$

$\log f(n) = n$ and $\log g(n) = 2n$

$\downarrow$

clearly $f(n) < g(n)$

[Even asymptotically $f(n) < g(n)$ as after log we can't equalise]

(7)

$$g_1(n) = \begin{cases} n^3 & \forall \; n < 100 \\ n^2 & \forall \; n \geq 100 \end{cases}$$

$$g_2(n) = \begin{cases} n^2 & \forall \; n < 10000 \\ n^3 & \forall \; n \geq 10000 \end{cases}$$

$g_1(n)$	n^3		n^2		n^2
		100		10000	
$g_2(n)$	n^2		n^2		n^3

we can write $g_2(n) > g_1(n) \;\forall\; \underbrace{n \geq 10000}_{\text{starting point}}$

so in general $g_2(n) > g_1(n)$

check if the asymptotic notations are correct:

(1) $(n+k)^m = \Theta(n^m)$ ✓

Highest term $= n^m = \Theta(n^m)$

(2) $2^{n+1} = O(2^n)$ $\quad 2 \cdot 2^n$

$O(2^n) = \Theta(2^n) = \Omega(2^n)$ ✓

(3) $2^{2n} = O(2^n)$ ✗ $\quad 2^{2n} = O(4^n)$

2^n isn't an upper Bound

as $4^n > 2^n$

④ $\sqrt{\log n} = O(\log \log n)$ ↳ cannot be upper bound ✗

⑤ $n^{\log n} = O(e^n)$ ✓

log → $\log \log n, < n$ ↳ upper Bound

Best, worst and Average case Analysis

1. Linear search
2. Binary search Tree

Linear search

A

8	6	12	5	9	7	4	8	16	18
0	1	2	3	4	5	6	7	8	9

searching key = 7 → 6 comparisions ✓ Yes
key = 20 → 10 comparisions ✗ No

* Best case key = 8 ⇒ present at the first index → $O(1)$ time (constant)

$B(n) = O(1)$ (Best case time)

☆ worst case key = 18 ⇒ present at the last index → O(n) time (linear)

$w(n) = O(n)$

☆ Average case key → $\dfrac{\text{all possible case times}}{\text{no. of cases}}$

↓ difficult to compute in general

$$\text{Avg time} = \frac{1+2+3+\cdots+n}{n} = \frac{n+1}{2}$$

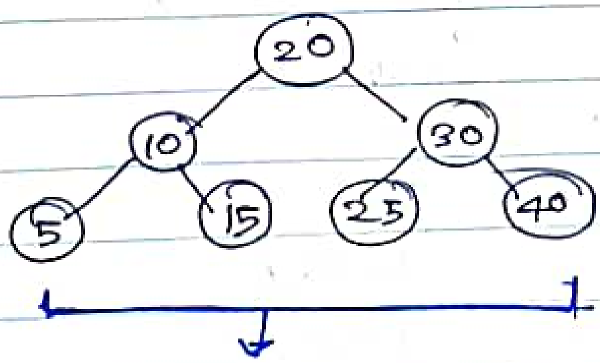
$$A(n) = (n+1)/2 = O(n)$$

$B(n) = O(1) = \Omega(1) = \Theta(1)$

$w(n) = O(n) = \Theta(n) = \Omega(n)$

$A(n) = O(n)$

Binary search Tree



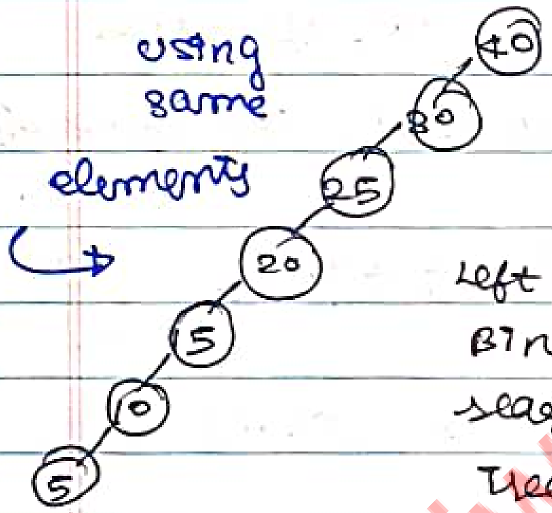
searching 15 ↪ log n comparisions

left < root < right

✲ Best case → searching root
= constant time → $B(n) = 1$

✲ worst case → searching for leaf element
= height → $w(n) = \log n$

using same elements ↪



Left skewed Binary search Tree

Best case ↳ $O(1)$

worst case ↳ $O(n)$

$$\boxed{\begin{array}{l} \min\ w(n) = \log n \\ \max\ w(n) = n \end{array}}$$

minimum worst case time

maximum worst case time

$w(n) = h$ in general

min $\log n$ → height balanced BST

max n → skewed BST