

TIME COMPLEXITY AND ASYMPTOTIC NOTATION

$O \rightarrow f(n) \rightarrow$ upper bound $(n) f.g \geq (n)$

$\Omega \rightarrow f(n) \rightarrow$ lower bound

$\Theta \rightarrow f(n) \rightarrow$ upper + lower bound $((n) f.g = (n))$

Big $O \rightarrow$ If $f(n)$ and $g(n)$ are two functions,
then $f(n) = O(g(n))$, If there exists constants C
and n_0 such that $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$.

Example —

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

Least upper bound.

$f(n)$ is least upper bound, $g(n)$ is big O.

Big O.

* If $f(n) = 4n+3$. Is $f(n) = O(g(n))$?

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

$$\Rightarrow 4n+3 \leq 5n$$

c $\leq n$ $\forall n \in \mathbb{N}$

$c \cdot g(n) \leq 5n \leq n^2$

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

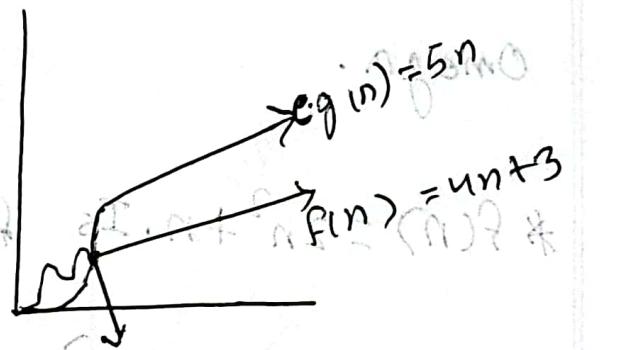
Now, $f(n) = 4n+3$ bounded below by $g(n) = n^2$

$$4n+3 \leq 5n$$

$$\Rightarrow 3 \leq 5n - 4n$$

$$\Rightarrow 3 \leq n$$

n_0



$n_0 = 3 \leq n$

$[n_0 \text{ point } \Rightarrow n_0^2 < 4n_0 + 3]$

* $f(n) = 2n^2 + n$. Find $O(g(n))$.

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

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

$$c \leq \frac{n}{n^2} = \frac{1}{n}$$

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

Find $\sum_{n=1}^{\infty} \frac{1}{n^2}$

$$\frac{1}{n} \leq 1 \leq n$$

$\Omega(g(n))$ — If $f(n) = g(n)$ and $g(n)$ are two functions then $f(n) = \Omega(g(n))$

there exists constants c and n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

Example -

$$f(n) = 2n^2 \quad \text{greatest lower bound } g(n) = n^2$$

$f(n)$ is greatest lower bound, $g(n)$ is ~~not~~ ω

$\Omega(g(n))$

* $f(n) = 2n^2 + n$. Is $f(n) \geq \Omega(g(n))$?

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

$$\Rightarrow 2n^2 + n \geq 2n^2$$

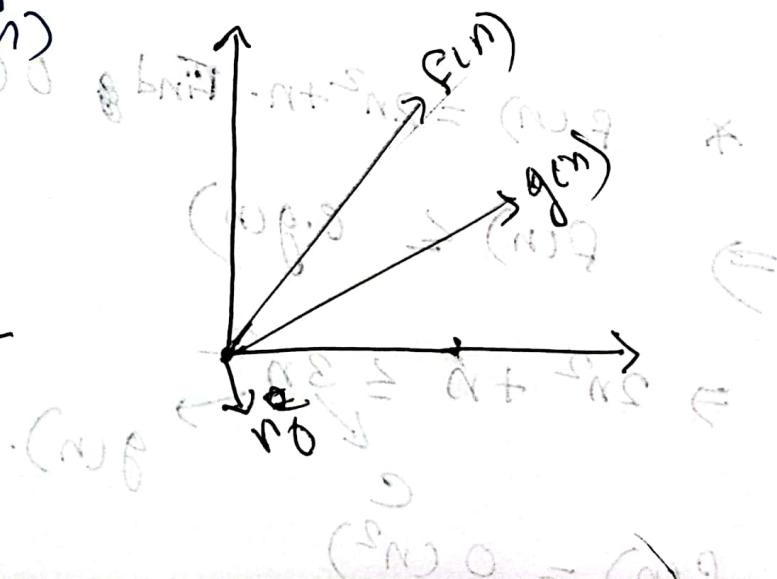
$$c \cdot (n^2)$$

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

Now,

$$2n^2 + n \geq 2n^2$$

$$\Rightarrow n \geq 0$$



$\Theta(\theta) \Rightarrow$ If $f(n)$ and $g(n)$ are two functions then $f(n) = \Theta(g(n))$, if there exists constants c and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0.$$

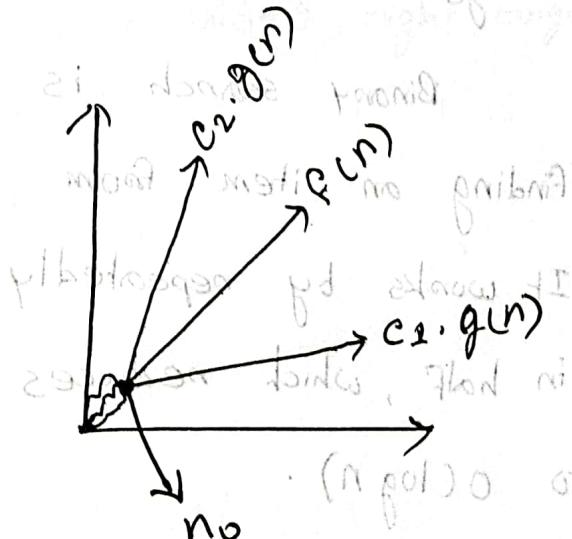
* $f(n) = 2n^2 + n$, $f(n) = \Theta(g(n)) = ?$

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

$$c_1 \frac{2n^2}{g(n)} \leq 2n^2 + n \leq c_2 \frac{3n^2}{g(n)}$$

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

$$= \Theta(n^2)$$



out size = $O(\log n)$ $\in \Theta(n \log n)$

smaller $\Theta(n \log n) \leq O(n^2)$ result another

3 steps of D & C approach -

(1) Divide (2) Conquer (3) Combine

↓
Creating subproblems

Solving
subproblems

combining solutions
to get the final

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

BINARY SEARCH ALGORITHM -

*Algorithm —

Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing the search interval in half, which reduces the search time complexity to $O(\log n)$.

#include <iostream>
using namespace std;

```
int binarySearch (int arr[], int left, int right, int target)
```

```
{ if (left <= right)
```

```
{ mid = left + (right - left) / 2;
```

```
if (arr[mid] == target) return mid;
```

```
return mid;
```

```
if (arr[mid] > target) l = [bin]arr
```

```
return binarySearch (arr, left, mid-1, target);
```

```
else
```

```
return binarySearch (arr, mid+1, right, target);
```

```
}
```

```
return -1;
```

```
}
```

F	d	E	N	S	S	L
08	05	21	21	01	8	1

```
21 = target, n = bin, p = tfloor, m = fceil
```

```
target = 21 = [bin]arr
```

```
p modified to bin < target, 08
```

Simulation - (Top part) Normal search

1	2	3	4	5	6	7
1	8	10	15	18	20	30

left = 0, right = 7, mid = [left + right] / 2 = 3, target = 15

Here, ; bin search

arr[mid] = 15 < target.

Now, left = mid + 1 = 3 + 1 = 4

1	2	3	4	5	6	7
1	8	10	15	18	20	30

left = 4, right = 7, mid = 5, target = 15

arr[mid] = 18 > target.

Now, right = mid - 1 = 5 - 1 = 4

1	2	3	4	5	6	7
1	8	10	15	18	20	30

left = 4, right = 4, mid = 4, target = 15

arr[mid] = 15 = target.

So, target found at position 4.

B

→ ~~multistep~~ Normal Process

Time Complexity -

In each iteration, the search range is divided in half.

From the figure, it is observed that size of base

$$\text{Itr} = 1, \text{ search range size} = \frac{n}{2}$$

$$\text{Itr} = 2, \text{ " " } = \frac{n}{4} = \frac{n}{2^2}$$

$$\text{Itr} = 3, \text{ " " } = \frac{n}{8} = \frac{n}{2^3}$$

$$\text{Itr} = k, \text{ " " } = \frac{n}{2^k}$$

→ multistep

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

$$\Rightarrow n = 2^k : \exists (T_{\text{last}} - T_{\text{first}}) + T_{\text{last}} = 1 \text{ bin}$$

$$\Rightarrow \log_2 n = \log_2 2^k : \exists (T_{\text{last}} - T_{\text{first}}) - T_{\text{first}} = 5 \text{ bin}$$

$$\Rightarrow k \log_2 n = \log_2 n : \text{if } T_{\text{first}} = [1 \text{ bin}]_{\text{last}}$$

$$\therefore k = \log_2 n : \text{if } T_{\text{first}} = [1 \text{ bin}]_{\text{last}} \rightarrow T_{\text{last}}$$

Best case: element at middle index. Complexity = $\Theta(1)$.

Average " : " found after the middle of search.

Complexity $\Rightarrow \Theta(\log_2 n)$

Worst " : element found at end of the search or not found. complexity $\Rightarrow \Theta(\log_2 n)$ L. - ~~multistep~~

■ Ternary Search Algorithm -

The ternary search algorithm is a divide and conquer search technique which is used to find the position of a target element in a sorted array by repeated dividing the search space into three equal parts and then narrowing the search to one of these three regions.

Algorithm —

```
# TernarySearch (int arr[], int left, int right, int target)
{
    if (left <= right) {
        int mid1 = left + (right - left) / 3;
        int mid2 = right - (right - left) / 3;

        if (arr[mid1] == target) return mid1;
        if (arr[mid2] == target) return mid2;
        if (target < arr[mid1])
            return ternarySearch(arr, left, mid1 - 1, target);
        else if (target > arr[mid2])
            return ternarySearch(arr, mid2 + 1, right, target);
        else
            return ternarySearch(arr, mid1 + 1, mid2 - 1, target);
    }
    return -1;
}
```

Simulation — ~~random~~ ~~sliding~~ ~~self~~ ~~to~~ ~~translates~~; ~~send back~~

0	1	2	3	4	5	6	7	8
6	8	14	16	18	20	23	32	45

~~left = 0, mid1 = 2, mid2 = 6, right = 8, target = 45~~

Here, *band* is formed by adding the suffix *-er* to the verb *band*.

$\text{arr}[\text{mid1}] = 14 < \text{arr}[\text{mid2}] = 23 < 45$

As $\text{target} > \text{arr}[\text{mid2}]$, then $\text{left} = \text{mid2} + 1 = 6 + 1 = 7$

0	1	2	3	4	5	6	7	8
6	8	14	16	18	20	23	32	45

left = 7, mid1 = 7, mid2 = 8, right = 8, target = 45

Now, arr[mid2] = target.

So, target found at position 8.

Time complexity:

Int = 1, search range size = $\frac{n}{3}$

$$It \text{ is } 2, \quad " \quad " \quad " \quad " = \frac{n}{9} = \frac{81}{3^2}$$

$$\text{If } n=3, \quad " \quad " \quad " \quad " = \frac{n}{27} = \frac{n}{3^3}$$

$$\text{If } n=k, \quad " \quad " \quad " = \frac{n}{3^k}$$

$$\frac{n}{3^k} = 1$$

$$\Rightarrow n = 3^k \quad \therefore k = \log_3^n$$

Best Case : element at the middle index. middle

$\Theta(1)$.

Average Case : element found in the middle of the

$\Sigma P = \text{fogart}$, $\Sigma = \text{Infixa}$, $\Sigma = \text{Sbin}$, $\Sigma = \text{Lbin}$, $\Sigma = \text{Rbin}$

Worst case : element found at either end of the

$\Sigma P > \Sigma$ search not found.

$\Sigma = L + R = L + S_{\text{bin}}$, $\Sigma = \Theta(\log_3^n)$, $S_{\text{bin}} < \text{fogart}$

2	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63	65	67	69	71	73	75	77	79	81	83	85	87	89	91	93	95	97	99	101	103	105	107	109	111	113	115	117	119	121	123	125	127	129	131	133	135	137	139	141	143	145	147	149	151	153	155	157	159	161	163	165	167	169	171	173	175	177	179	181	183	185	187	189	191	193	195	197	199	201	203	205	207	209	211	213	215	217	219	221	223	225	227	229	231	233	235	237	239	241	243	245	247	249	251	253	255	257	259	261	263	265	267	269	271	273	275	277	279	281	283	285	287	289	291	293	295	297	299	301	303	305	307	309	311	313	315	317	319	321	323	325	327	329	331	333	335	337	339	341	343	345	347	349	351	353	355	357	359	361	363	365	367	369	371	373	375	377	379	381	383	385	387	389	391	393	395	397	399	401	403	405	407	409	411	413	415	417	419	421	423	425	427	429	431	433	435	437	439	441	443	445	447	449	451	453	455	457	459	461	463	465	467	469	471	473	475	477	479	481	483	485	487	489	491	493	495	497	499	501	503	505	507	509	511	513	515	517	519	521	523	525	527	529	531	533	535	537	539	541	543	545	547	549	551	553	555	557	559	561	563	565	567	569	571	573	575	577	579	581	583	585	587	589	591	593	595	597	599	601	603	605	607	609	611	613	615	617	619	621	623	625	627	629	631	633	635	637	639	641	643	645	647	649	651	653	655	657	659	661	663	665	667	669	671	673	675	677	679	681	683	685	687	689	691	693	695	697	699	701	703	705	707	709	711	713	715	717	719	721	723	725	727	729	731	733	735	737	739	741	743	745	747	749	751	753	755	757	759	761	763	765	767	769	771	773	775	777	779	781	783	785	787	789	791	793	795	797	799	801	803	805	807	809	811	813	815	817	819	821	823	825	827	829	831	833	835	837	839	841	843	845	847	849	851	853	855	857	859	861	863	865	867	869	871	873	875	877	879	881	883	885	887	889	891	893	895	897	899	901	903	905	907	909	911	913	915	917	919	921	923	925	927	929	931	933	935	937	939	941	943	945	947	949	951	953	955	957	959	961	963	965	967	969	971	973	975	977	979	981	983	985	987	989	991	993	995	997	999	1001	1003	1005	1007	1009	1011	1013	1015	1017	1019	1021	1023	1025	1027	1029	1031	1033	1035	1037	1039	1041	1043	1045	1047	1049	1051	1053	1055	1057	1059	1061	1063	1065	1067	1069	1071	1073	1075	1077	1079	1081	1083	1085	1087	1089	1091	1093	1095	1097	1099	1101	1103	1105	1107	1109	1111	1113	1115	1117	1119	1121	1123	1125	1127	1129	1131	1133	1135	1137	1139	1141	1143	1145	1147	1149	1151	1153	1155	1157	1159	1161	1163	1165	1167	1169	1171	1173	1175	1177	1179	1181	1183	1185	1187	1189	1191	1193	1195	1197	1199	1201	1203	1205	1207	1209	1211	1213	1215	1217	1219	1221	1223	1225	1227	1229	1231	1233	1235	1237	1239	1241	1243	1245	1247	1249	1251	1253	1255	1257	1259	1261	1263	1265	1267	1269	1271	1273	1275	1277	1279	1281	1283	1285	1287	1289	1291	1293	1295	1297	1299	1301	1303	1305	1307	1309	1311	1313	1315	1317	1319	1321	1323	1325	1327	1329	1331	1333	1335	1337	1339	1341	1343	1345	1347	1349	1351	1353	1355	1357	1359	1361	1363	1365	1367	1369	1371	1373	1375	1377	1379	1381	1383	1385	1387	1389	1391	1393	1395	1397	1399	1401	1403	1405	1407	1409	1411	1413	1415	1417	1419	1421	1423	1425	1427	1429	1431	1433	1435	1437	1439	1441	1443	1445	1447	1449	1451	1453	1455	1457	1459	1461	1463	1465	1467	1469	1471	1473	1475	1477	1479	1481	1483	1485	1487	1489	1491	1493	1495	1497	1499	1501	1503	1505	1507	1509	1511	1513	1515	1517	1519	1521	1523	1525	1527	1529	1531	1533	1535	1537	1539	1541	1543	1545	1547	1549	1551	1553	1555	1557	1559	1561	1563	1565	1567	1569	1571	1573	1575	1577	1579	1581	1583	1585	1587	1589	1591	1593	1595	1597	1599	1601	1603	1605	1607	1609	1611	1613	1615	1617	1619	1621	1623	1625	1627	1629	1631	1633	1635	1637	1639	1641	1643	1645	1647	1649	1651	1653	1655	1657	1659	1661	1663	1665	1667	1669	1671	1673	1675	1677	1679	1681	1683	1685	1687	1689	1691	1693	1695	1697	1699	1701	1703	1705	1707	1709	1711	1713	1715	1717	1719	1721	1723	1725	1727	1729	1731	1733	1735	1737	1739	1741	1743	1745	1747	1749	1751	1753	1755	1757	1759	1761	1763	1765	1767	1769	1771	1773	1775	1777	1779	1781	1783	1785	1787	1789	1791	1793	1795	1797	1799	1801	1803	1805	1807	1809	1811	1813	1815	1817	1819	1821	1823	1825	1827	1829	1831	1833	1835	1837	1839	1841	1843	1845	1847	1849	1851	1853	1855	1857	1859	1861	1863	1865	1867	1869	1871	1873	1875	1877	1879	1881	1883	1885	1887	1889	1891	1893	1895	1897	1899	1901	1903	1905	1907	1909	1911	1913	1915	1917	1919	1921	1923	1925	1927	1929	1931	1933	1935	1937	1939	1941	1943	1945	1947	1949	1951	1953	1955	1957	1959	1961	1963	1965	1967	1969	1971	1973	1975	1977	1979	1981	1983	1985	1987	1989	1991	1993	1995	1997	1999	2001	2003	2005	2007	2009	2011	2013	2015	2017	2019	2021	2023	2025	2027	2029	2031	2033	2035	2037	2039	2041	2043	2045	2047	2049	2051	2053	2055	2057	2059	2061	2063	2065	2067	2069	2071	2073	2075	2077	2079	2081	2083	2085	2087	2089	2091	2093	2095	2097	2099	2101	2103	2105	2107	2109	2111	2113	2115	2117	2119	2121	2123	2125	2127	2129	2131	2133	2135	2137	2139	2141	2143	2145	2147	2149	2151	2153	2155	2157	2159	2161	2163	2165	2167	2169	2171	2173	2175	2177	2179	2181	2183	2185	2187	2189	2191	2193	2195	2197	2199	2201	2203	2205	2207	2209	2211	2213	2215	2217	2219	2221	2223	2225	2227	2229	2231	2233	2235	2237	2239	2241	2243	2245	2247	2249	2251	2253	2255	2257	2259	2261	2263	2265	2267	2269	2271	2273	2275	2277	2279	2281	2283	2285	2287	2289	2291	2293	2295	2297	2299	2301	2303	2305	2307	2309	2311	2313	2315	2317	2319	2321	2323	2325	2327	2329	2331	2333	2335	2337	2339	2341	2343	2345	2347	2349	2351	2353	2355	2357	2359	2361	2363	2365	2367	2369	2371	2373	2375	2377	2379	2381	2383	2385	2387	2389	2391	2393	2395	2397	2399	2401	2403	2405	2407	2409	2411	2413	2415	2417	2419	2421	2423	2425	2427	2429	2431	2433	2435	2437	2439	2441	2443	2445	2447	2449	2451	2453	2455	2457	2459	2461	2463	2465	2467	2469	2471	2473	2475	2477	2479	2481	2483	2485	2487	2489	2491	2493	2495	2497	2499	2501	2503	2505	2507	2509	251

Recurrence Relation -

* Substitution Method -

→ void Test (int n) → $T(n)$

{

if ($n > 0$)

{ printf ("%d", n); }

Test (n-1);

}

$T(n) =$

$$\begin{cases} 1, & n = 0 \\ T(n-1) + 1, & n > 0 \end{cases}$$

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

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

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

$$= T(n-3) + 3$$

$$= T(n-k) + k$$

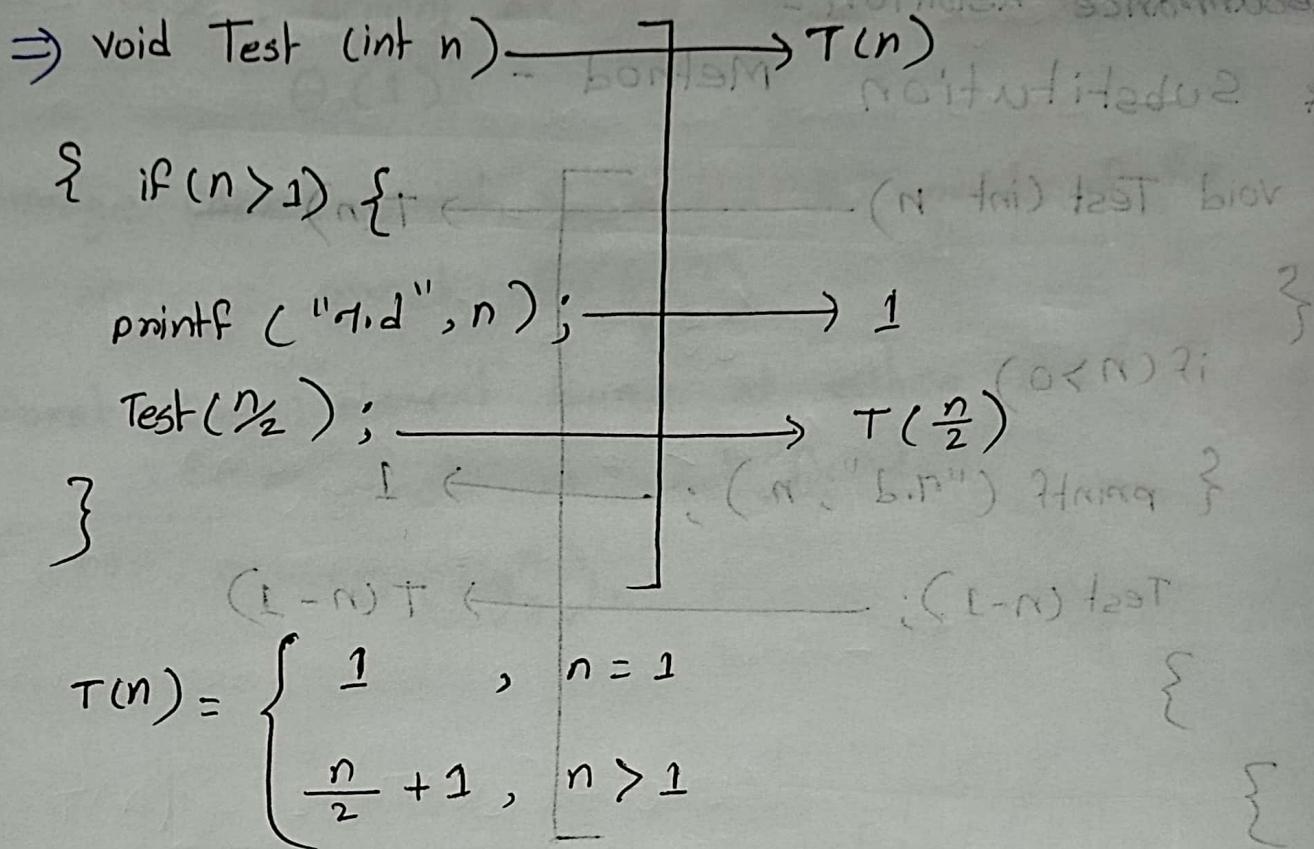
If $k = n$

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

$$= T(0) + n$$

$$= 1 + n$$

Time complexity = $O(n)$



$$\begin{aligned}
 T(n) &= T\left(\frac{n}{2}\right) + 1 & 0 = n & , \quad 1 \\
 &= \left[T\left(\frac{n}{4}\right) + 1 \right] + 1 & 0 < n & , 1 + (L-N)T \\
 &= \left[\left[T\left(\frac{n}{8}\right) + 1 \right] + 1 \right] + 1 & L + (L-N)T & = (n)T \\
 &= T\left(\frac{n}{2^3}\right) + 3 & L + [L + (L-N)T] & = \\
 &= T\left(\frac{n}{2^k}\right) + k & \cancel{\text{when}} & = \\
 &= T\left(\frac{n}{2^k}\right) + \left[1 + [1 + [1 + (1-N)T]] \right] & =
 \end{aligned}$$

When,

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

$$\Rightarrow 2^k = n$$

$$\Rightarrow \log_2 2^k = \log_2 n$$

$$\Rightarrow k \log_2 2 = \log_2 n$$

$$k + (k-N)T =$$

$$n = 4 \quad ?$$

$$n + (n-N)T =$$

$$n + (0)T =$$

$$n + 1 =$$

No. of multiplications = $\log_2 n$

$$\therefore k = \log_2 n$$

~~$T(n)$~~ = $T(1) + k$

$$= 1 + \log_2 n$$

$\therefore f(n) < b \quad \forall n \quad (1)$

$$\therefore f(n) = b \quad \forall n \quad (2)$$

$$\therefore f(n) > b \quad \forall n \quad (3)$$

Master Method \Rightarrow

\Rightarrow Master theorem - ~~multiplication doesn't pyramid~~ not pyramid not

$$T(n) = a^k T\left(\frac{n}{b}\right) + O(n^d)$$

Here $a \geq 1, b > 1$

Here,

$T(n)$ = Time complexity of the algorithm on an input of size n .

a = number of recursive subproblems in each division steps.

$\frac{n}{b}$ = size of each subproblem.

$O(n^d)$ = The time complexity of combining the results of all subproblems and any additional work done outside the recursive calls.

⇒ Time complexity ($T(n)$) of the algorithm can be determined based on three cases -

(1) $O(n^d)$ if $d > \log_b a$

(2) $O(n^d \log_b^n)$ if $d = \log_b a$

(3) $O(n \log_b^a)$ if $d < \log_b a$

For binary search algorithm -

$$T(n) = 1 * T\left(\frac{n}{2}\right) + O(1)$$

Hence, $a=1$ [मात्राकारी विकल्प का समानांक $\frac{1}{2}$ के बराबर है। अतः एक विकल्प का समानांक $\frac{1}{2}$ है।]

$$\frac{n}{b} = \frac{n}{2}, b=2$$

$$nd = 1, \text{ so, } d = 0$$

$$\text{Now, } \log_b^a = \log_2^1 = 0 = d \text{ जो मध्यम } = 0$$

$$\therefore T(n) = O(\log_2^n) [\text{if } d = \log_b^a, T(n) = O(n^d \log_b^n)]$$

इसका अर्थ यह है कि यदि विकल्पों की संख्या n हो तो इसका विकल्प का समानांक $\frac{1}{2}$ हो जाता है। अतः शर्करा का विकल्प का समानांक $\frac{1}{2}$ हो जाता है।

For Ternary search algorithm -

HOIT 2009 TS

$$T(n) = 1 * T\left(\frac{n}{3}\right) + O(1)$$

Hence, $a=1$ [1 तिरा फिरा 2009 अप्पे]
 $b=3$, $n^d = 1$, so, $d=0$

Now, $\log_b^a = \log_3^1 = 0 = d$

$$\therefore T(n) = O(\log_3^n)$$

For merge sort algorithm -

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

Hence, $a=2$ [2 तिरा फिरा 2009]

$$b=2$$

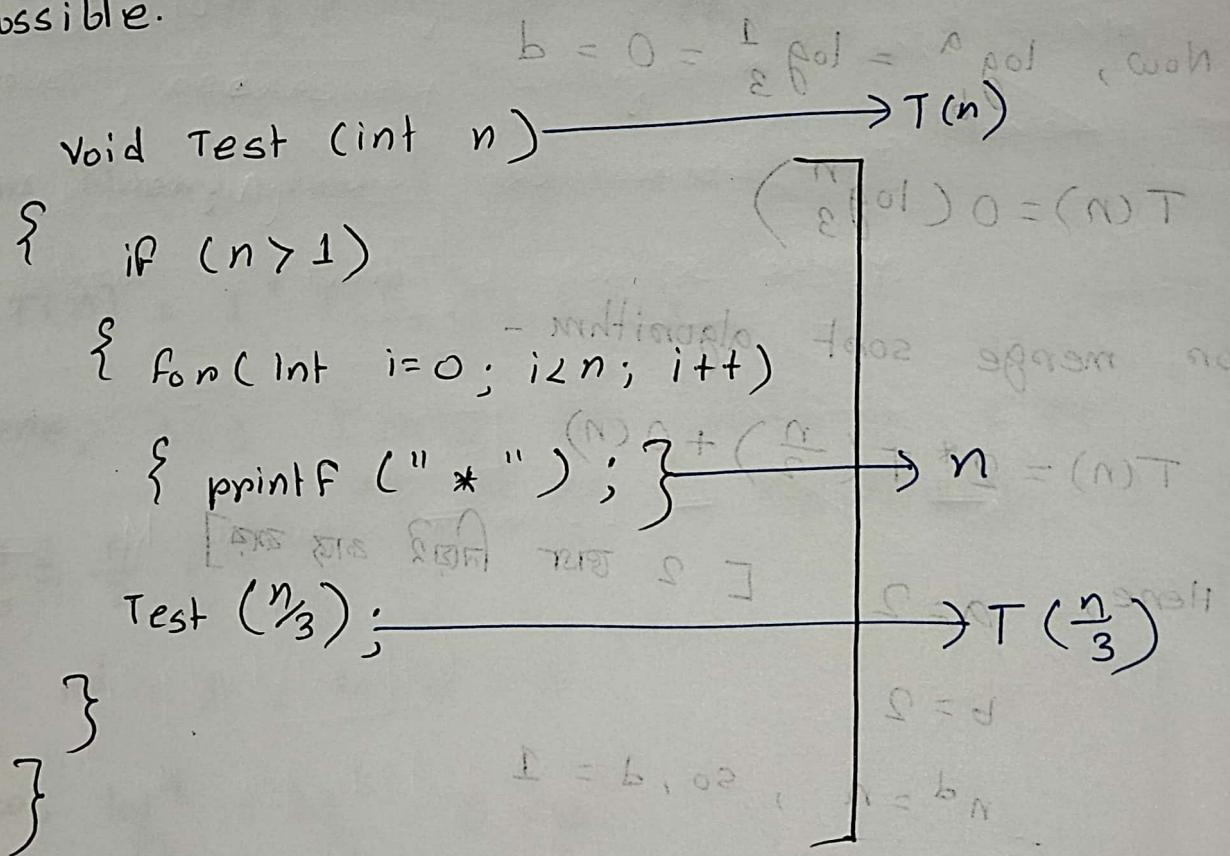
$$n^d = n, \text{ so, } d = 1$$

Now, $\log_b^a = \log_2^2 = 1 = d$

$$T(n) = O\left(n \log_2^n\right)$$

CT QUESTION

→ Derive recursive relation on equation
 from the following function and find the
 time complexity using master method if
 possible.



Recursive relation \Rightarrow

$$T(n) = \begin{cases} 1, & \text{when } n \leq 1 \\ T\left(\frac{n}{3}\right) + n, & \text{when } n > 1 \end{cases}$$

$$T(n) = \cancel{1} \times T\left(\frac{n}{3}\right) + n$$

$$= 1 \times T\left(\frac{n}{3}\right) + n$$

$a = 1$, $b = 3$, $n^d = n$, $\log_b d = 1$

Now, $\log_b d = \log \frac{1}{3} = 0 < d$

Hence, $d > \log_b a$.

$$\text{So, } T(n) = O(n^d) = O(n)$$

(using 1, 2, 3, 4) face of A

(using 1, 2) 3

$$\{(1)(f_{21} - f_{11}) + f_{11} = b_{11}\}$$

;(b_{11}, f_{11}, a_{11}) face of A

;(f_{11}, 1 + b_{11}, a_{11}) face of A

;(f_{11}, b_{11}, f_{21}, a_{11}) face of A

{

{

Merge Sort Algorithm

Merge sort is an efficient, stable and comparison-based sorting algorithm that follows the divide and conquer technique. The algorithm works by dividing an input array into two halves, recursively sorting each half, and then merging the two sorted halves back together to form a sorted array.

Algorithm -

~~void~~ MergeSort (~~int~~ arr[], ~~int~~ left, ~~int~~ right)

{

{ if (left < right)

{ int mid = left + (right - left) / 2;

mergeSort (arr, left, mid);

mergeSort (arr, mid + 1, right);

merge (arr, left, mid, right);

}

}

Merge (arr[], left, mid, right)

- not balanced

{ # Dividing the array into two parts.

{ n₁ = mid - left + 1; // length of left-half

n₂ = right - mid; // length of right-half.

left_half [n₁] = arr [left ----- mid] .

right_half [n₂] = arr [mid+1 ----- right].

i=0 # left-half pointer

j=0 # right-half pointer

k=left # main array pointer.

while i < n₁ and j < n₂:

if left_half [i] ≤ right_half [j]:

arr [k] = left_half [i]

i = i + 1

else

arr [k] = right_half [j]

j = j + 1

k = k + 1 .

while i < n₁:

arr [k] = left_half [i]

i = i + 1

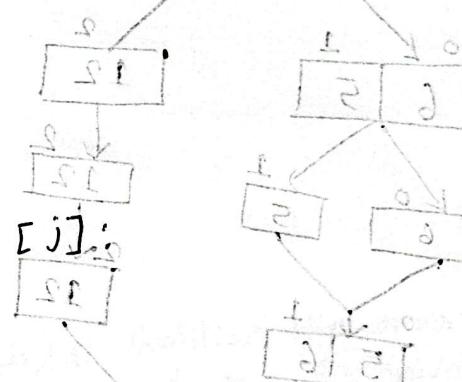
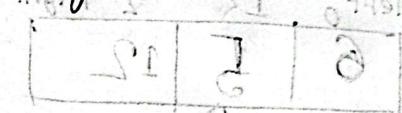
k = k + 1

while j < n₂:

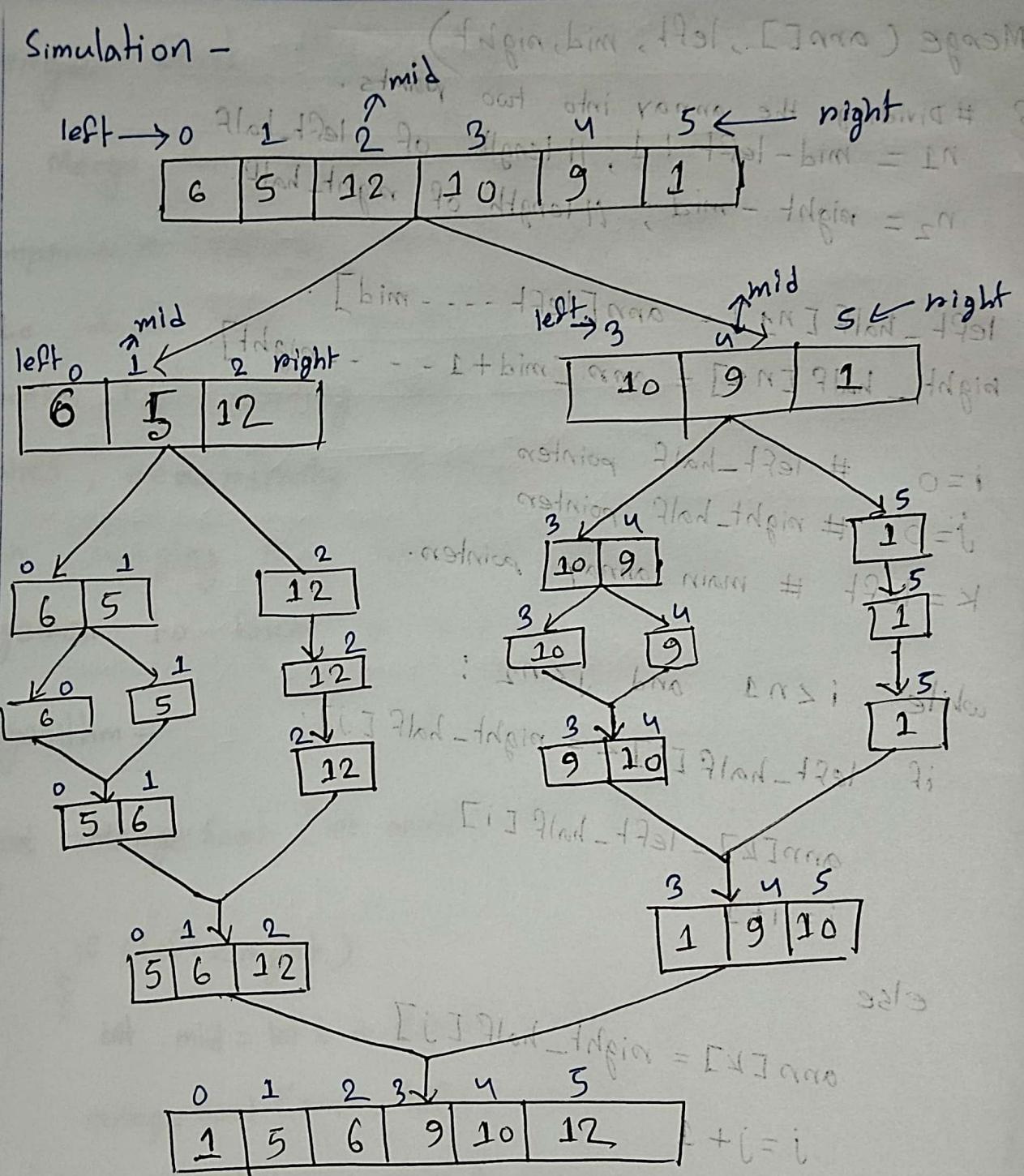
arr [k] = right_half [j]

j = j + 1 . k = k + 1

3



Simulation -



Time Complexity, $T(n) = O(n \log_2 n)$: Insert : slide

Quick SORT ALGORITHM : (Initial, [] array) partition

Quick sort is another efficient, comparison-based divide and conquer sorting algorithm which works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays; those less than pivot and those greater than the pivot. The sub-arrays are then recursively sorted.

Algorithm :

Quicksort (arr[], low, high) {

if (low < high)

 pivot = Partition (arr, low, high) # Partition the array and get the pivot index.

Recursively sort the two halves

 Quicksort (arr, low, pivot - 1);

 Quicksort (arr, pivot + 1, high)

}

partition, first, second, ... => [] array

Partition (arr[], low, high):

{ pivot = arr[low] # Select pivot as 1st element

i = low + 1 # initialize i to start after pivot
j = high # initialize j to the end of the array

while i <= j:

Find element > pivot on the left

while i <= high and arr[i] <= pivot:

i = i + 1

Find element <

while j >= low and arr[j] > pivot:

j = j - 1

swap (arr[i], arr[j])

i = i + 1

j = j - 1

swap pivot with arr[j]

swap (arr[low], arr[j])

return j # return pivot's final position.

Simulation :

Pivot	0	1	2	3	4	5	6	7	8	9	10
	35	50	15	25	80	20	90	45			
	i		j								

Now, $i < j$, so swap [50, 20]

Pivot	0	1	2	3	4	5	6	7	8	9	10
	35	20	15	25	80	50	90	45			
	i		i		j						

i > j [overlap], swap (pivot) \rightarrow swap [35, 25]

0	1	2	3	4	5	6	7	8	9	10
25	20	15	35	80	50	90	45			
j	i									

Now, Partition function will return $j=3$ to quick sort function.

Then -

0	1	2	3	4	5	6	7	8	9	10
25	20	15	35	80	50	90	45			

0	1	2	i	3	4	5	6	7	8	9	10
25	20	15	35	80	50	90	45				
	i	j									

i, j overlap. swap (25, 15)

0	1	2	3	4	5	6	7	8	9	10
15	20	25	35	80	50	90	45			
4	i	j								

$i < j$, swap (90, 45).

0	1	2	3	4 ↑	5 ↓	6 ↓	7 ↓
25	20	25	35	80	50	45	90

pivot

$i \rightarrow j$

i, j overlap. swap(80, 45)

0	1	2	3	4 ↑	5 ↓	6 ↓	7 ↓
25	20	25	35	45	50	80	90

which is the sorted array.

Time Complexity $\Theta(n \log n)$

Best case - Pivot at mean position

and two balanced part $= O(n \log_2 n)$

Worst case - pivot is the smallest or largest element - $O(n^2)$.

Average case - selects pivot at uniformly

at random position - $O(n \log_2 n)$.

25	20	25	35	80	50	45	90
25	20	25	35	80	50	45	90

(25, 25) form qdarray (i, j)

25	20	25	35	80	50	45	90
25	20	25	35	80	50	45	90

DYNAMIC PROGRAMMING

→ Problematic

Divide and Conquer — Disjoint Subproblems.

Dynamic Programming — Overlapping subproblems.

Fibonacci Series —

Recursive Approach —

$$\text{fib}(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ \text{fib}(n-1) + \text{fib}(n-2) & n > 1 \end{cases}$$

```
int fib (int n) {
```

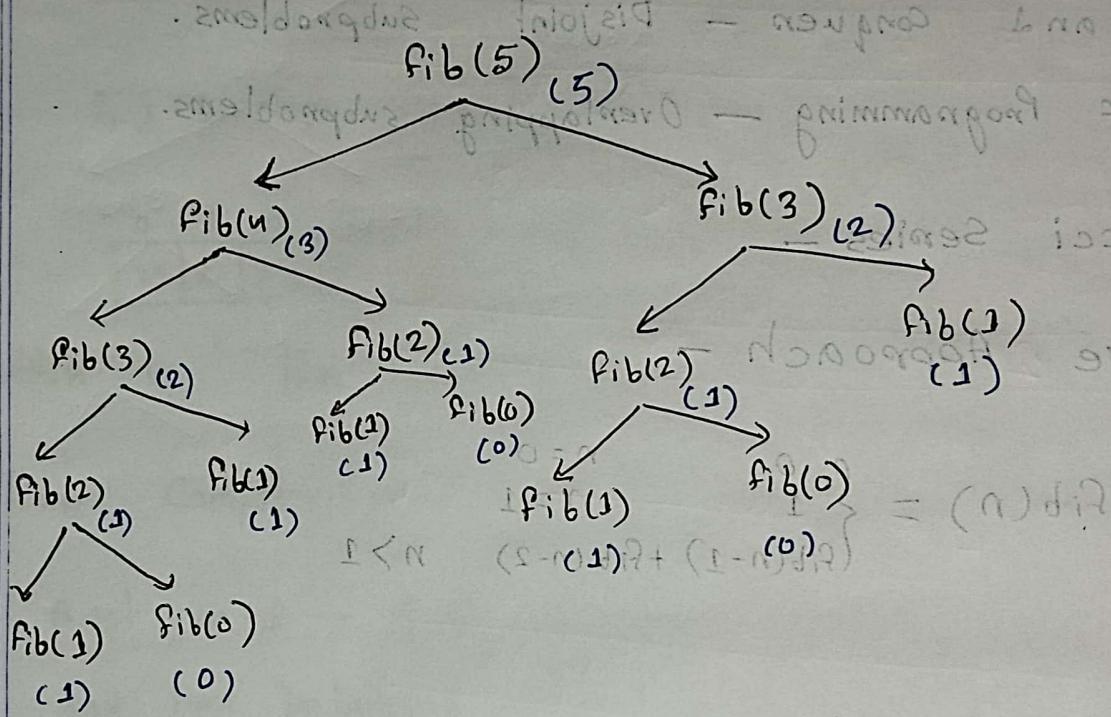
 if ($n \leq 1$)

 return n;

 return fib(n-1) + fib(n-2);

}

Simulation —



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

$$\approx 2T(n-1) + 1$$

$$a=2, b=1; a \geq 1, b \neq 1$$

$T(n) = O(2^n) \rightarrow$ Exponential complexity.

DP Approach -

```
int dp[n+1];
```

```
int fibonacci(int n)
```

```
{ if (n == 0)
```

```
{ return 0;
```

```
}
```

```
else if (n == 1)
```

```
{ return 1;
```

```
}
```

```
else if (dp[n] == 0) —————— Recursion Tree
```

```
{
```

```
dp[n] = fibonacci(n-1) + fibonacci(n-2);
```

$\vdash \text{if } n \neq 0 \quad \text{so} \quad 0 = n \quad \text{if}$

```
}
```

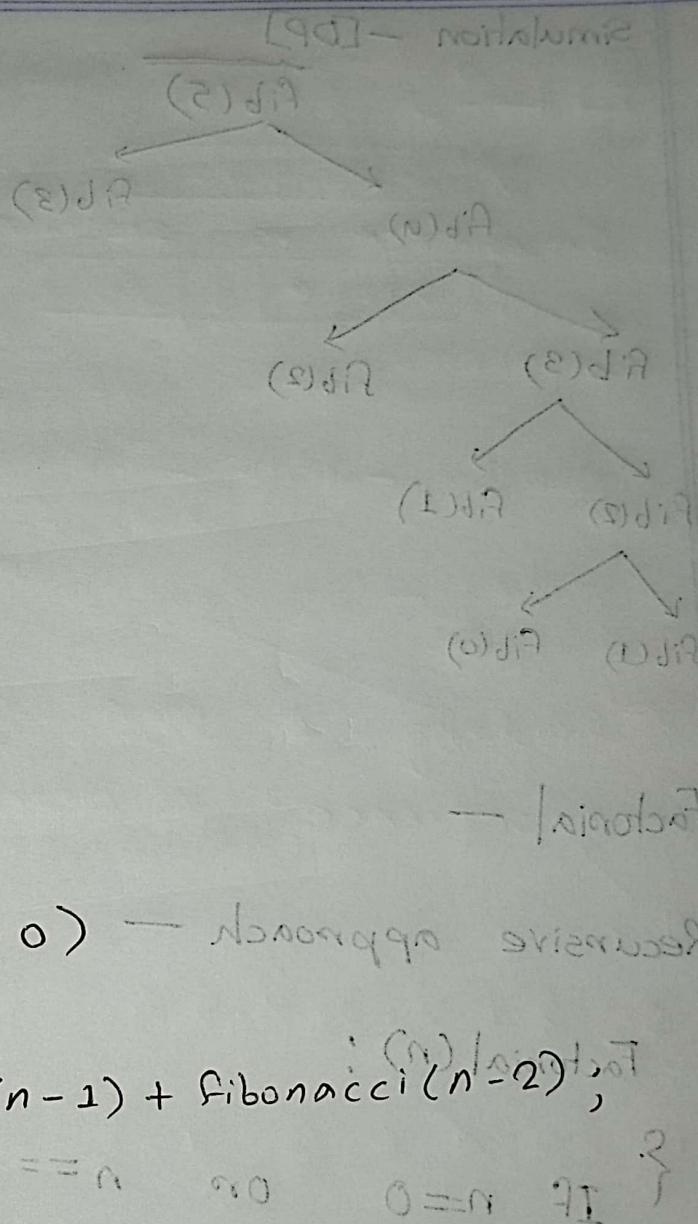
```
return dp[n];
```

↓ number

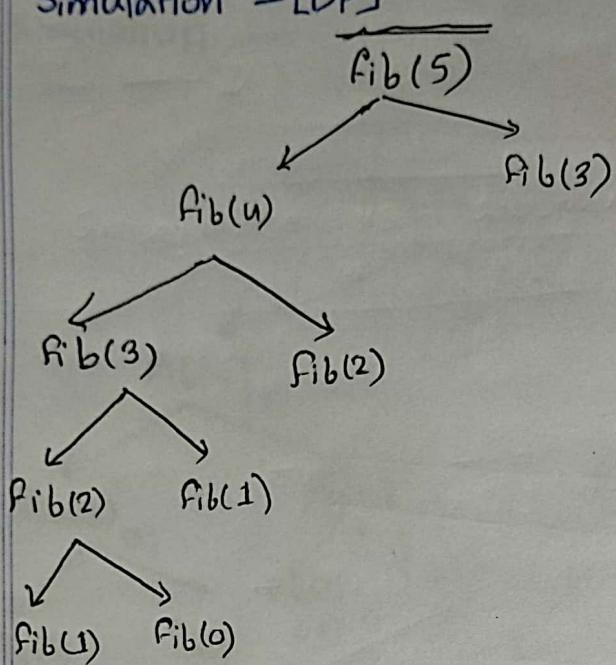
```
}
```

($n-1$) is also a number

{



Simulation - [DP]



Factorial —

Recursive approach —

Factorial(n) :

{ If $n=0$ or $n=1$

return 1

return $n * \text{Factorial}(n-1)$

}

DP Approach - ~~reduces common subproblem~~ - 25 L

dp[n+1] # An array of size n+1 to store the results
initially all elements are 0

Factorial(n)

{ if $n=0$ on size $n=1$ add - minnig
return 1

else if $dp[n]=0$ name \rightarrow printed \leftarrow minnig

$dp[n] = n * \text{Factorial}(n-1)$

return $dp[n]$.

}

$\{OK, 8, d, N, S\} = E$

common subproblems

3 base cases

$\{N, S\} = X$

$\{d, N, S\} = S$

base case

$\{OK, 8, d, N, S\} = X$

LCS - Longest Common Subsequence

Subsequence vs substring.

"programming"

grammm - both subsequence and substring

gammg - subsequence, not substring.

* Substring is main sequence

same sequence

Now,

$$S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$E = \{2, 4, 6, 8, 10\}$$

$$X_1 = \{2, 4\}$$

$$X_2 = \{2, 4, 6\}$$

$$X_3 = \{2, 4, 6, 8, 10\}$$

Common subsequence
of S and E

Longest.

Simulation -

i → X [0 1 2
b d 10]

j → Y [0 1 2 3 4
a b c d 10]

LCS(0,0)
x[0], y[0]
b a

$\max \{ \text{LCS}(1,0), \text{LCS}(0,1) \}$

$\max \{ \text{LCS}(2,0), \text{LCS}(1,1) \}$

$1 + \text{LCS}(1,2)$

$\max \{ \text{LCS}(2,1), \text{LCS}(1,2) \}$

$\max \{ \text{LCS}(2,2), \text{LCS}(1,3) \}$

$\max \{ \text{LCS}(2,2), \text{LCS}(1,3) \}$

$1 + \text{LCS}(2,4)$

$0 = \text{LCS}(2,4)$

$\text{LCS} = 2$

LCS [DP Approach] -

→ ~~Algorithm~~

LCS(i, j):

To find the ~~seq~~ LCS of 2 sequences, we need to follow the steps given below -

- ① Create a ~~dp~~ matrix dp of $m+1$ by $n+1$ size where m and n are sizes of the given sequences.
[$dp[i, j]$ = LCS of x_i and y_j] and traverse each cell now-wise from column-wise.
- ② Traverse each cell of ~~dp~~ matrix dp row-wise from bottom to top.
- ③ If $i=0$ or $j=0$, then set $dp[i, 0] = dp[0, j] = 0$.
- ④ If $x_i = y_j$, set $dp[i, j] = dp[i-1, j-1] + 1$.
- ⑤ If $x_i \neq y_j$, set $dp[i, j] = \max(dp[i-1, j], dp[i, j-1])$.

$$O = [0, 0]_{qb} = [0, i]_{qb} + \text{for } 0 \leq i \leq l (s)$$

below numbers are given after $[0, 0]$ for segment (s)

with ~~l~~ \rightarrow max i values for $[0, i]$ for (s)

with previous minimum

$$[0, l-s]_{qb} = [0, i]_{qb} + \text{for}$$

from s to l select max frequency of i for $[0, i]$

Knapsack -

- [Lecture 9] 20.1

Algorithm —

To solved knapsack problem of been
To solved the 0/1 knapsack problem

with n items, each with a weight and a value, and a knapsack with a maximum capacity, w , we need to follow the steps given below:

(1) Create a matrix dp of size $n+1$ by $w+1$ [where $dp[i][w]$ = the maximum value of items $1 \dots i$ with weight limit w].

(2) If $i=0$ or $w=0$, set $dp[i,0] = dp[0,w] = 0$.

(3) Traverse each cell row-wise or column-wise.

(4) If weight of current item i exceeds the remaining capacity w ,

set $dp[i,w] = dp[i-1,w]$.

[skip the current item and take value of previous row].

(5) If the current item can fit ($\text{weight}[i-1] \leq w$):
set $dp[i, w] = \max(dp[i-1, w], dp[i-1, w - \text{weight}[i]] + v(i))$

[choose the maximum between skipping the item or taking it]

(6) The value at $dp[n, w]$ will be the maximum value ~~to~~ that can be achieved with the given capacity.