

Searching & Hashing

Searching :-

searching is a process of finding an element within the list of elements (sorted in any order / randomly)

① Linear search / Sequential search :-

Elements are searched one by one sequentially

Procedure SEQSEARCH (i, n, k)

// i = variable

// n = no of elements in list

// k = we have to search it

$k_0 \leftarrow k$

$i \leftarrow n$

while $k_i \neq k$ do

$i \leftarrow i-1$

end

end SEQSEARCH

complexity :- $O(n)$

for $i \leftarrow n$ to 0
~~if $k_i \neq k$~~

→ 2

2

② Binary Search complexity :- $O(\log n)$

Searching is done in ordered / sorted sequential file

Procedure BINSEARCH (i, n, k)

$l \leftarrow 1$
 $u \leftarrow n$

while $(l \leq u)$ do

$m \leftarrow [(l+u)/2]$ // index of middle record

case

: $k > k_m$; $l \leftarrow m+1$ // look in upper half

: $k = k_m$; $i \leftarrow m$; return

: $k < k_m$; $u \leftarrow m-1$

// look in lower half
// no record is present

end end $i \leftarrow 0$

Procedure

①

②

③

④

⑤

⑥ exit

Hashing

Sortin

① Bubble

→ 11

11

11

→ 2

2

→ 2

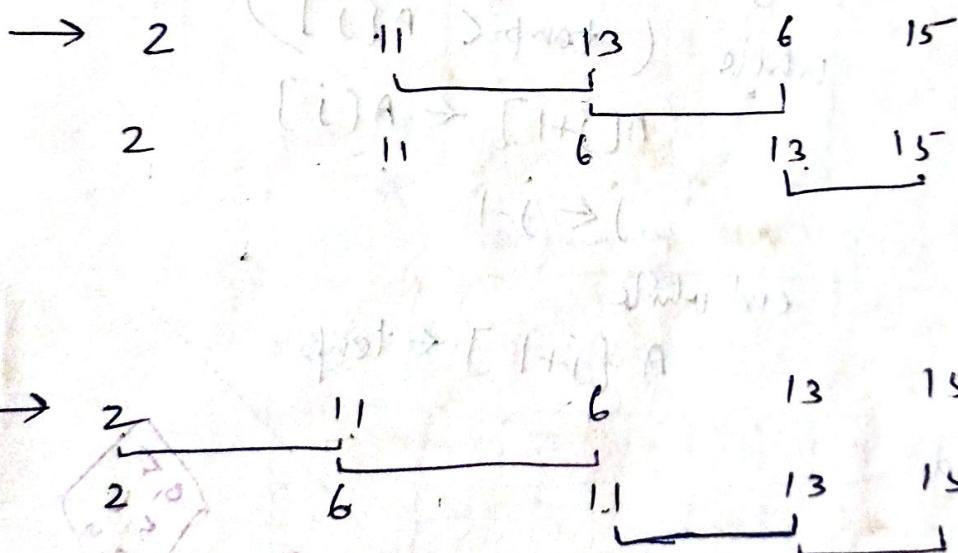
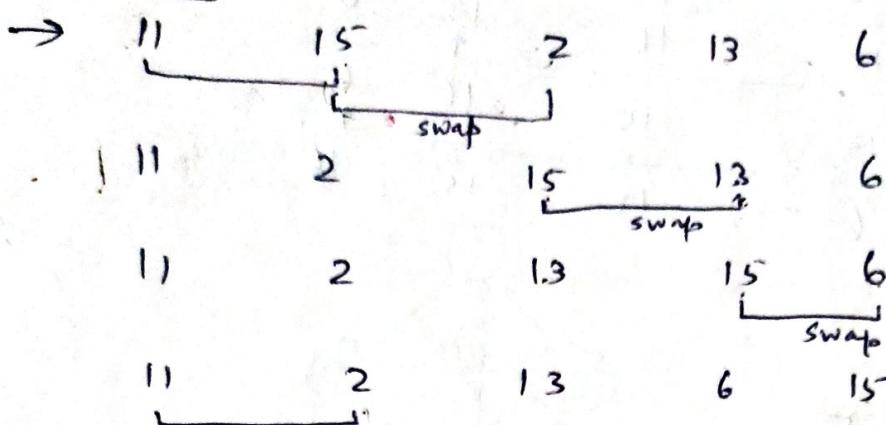
2

arranging of data items is possible in linear and one by one sequentially

Hashing:-

Sorting

① Bubble :-



Procedure Bubble-sort

- ① exit
- ② Repeat step 3 to 5 until $i < (n-1)$
- ③ $j \leftarrow 0$
- ④ Repeat step ⑤ until $j < (n-i-1)$
- ⑤ if $A[j] > A[j+1]$
 temp $\leftarrow A[j]$
 $A[j] \leftarrow A[j+1]$
 $A[j+1] \leftarrow temp$

end if

- ⑥ exit

Inception sort-

I	16	15	2	13	6
II	15	16	2	13	6
III	2	15	16	13	6
IV	2	13	16	15	6

avg n -
Basic

Graph R
Adj

Inception sort

for ($i \leftarrow 1$) to ($n-1$)

temp $\leftarrow A[i]$
 $j \leftarrow i-1$

while (temp < $A[j]$)

$A[j+1] \leftarrow A[j]$
 $j \leftarrow j-1$

end while

$A[j+1] \leftarrow \text{temp}$

end.

Street chalking
open Addressing
Linear sorting
Quadratic sorting
Double Refilling

4 (Clearance)
forward
idle 2

18 min 27 52, 55, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 256, 258, 260, 262, 264, 266, 268, 270, 272, 274, 276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 298, 300, 302, 304, 306, 308, 310, 312, 314, 316, 318, 320, 322, 324, 326, 328, 330, 332, 334, 336, 338, 340, 342, 344, 346, 348, 350, 352, 354, 356, 358, 360, 362, 364, 366, 368, 370, 372, 374, 376, 378, 380, 382, 384, 386, 388, 390, 392, 394, 396, 398, 400, 402, 404, 406, 408, 410, 412, 414, 416, 418, 420, 422, 424, 426, 428, 430, 432, 434, 436, 438, 440, 442, 444, 446, 448, 450, 452, 454, 456, 458, 460, 462, 464, 466, 468, 470, 472, 474, 476, 478, 480, 482, 484, 486, 488, 490, 492, 494, 496, 498, 500, 502, 504, 506, 508, 510, 512, 514, 516, 518, 520, 522, 524, 526, 528, 530, 532, 534, 536, 538, 540, 542, 544, 546, 548, 550, 552, 554, 556, 558, 560, 562, 564, 566, 568, 570, 572, 574, 576, 578, 580, 582, 584, 586, 588, 590, 592, 594, 596, 598, 600, 602, 604, 606, 608, 610, 612, 614, 616, 618, 620, 622, 624, 626, 628, 630, 632, 634, 636, 638, 640, 642, 644, 646, 648, 650, 652, 654, 656, 658, 660, 662, 664, 666, 668, 670, 672, 674, 676, 678, 680, 682, 684, 686, 688, 690, 692, 694, 696, 698, 700, 702, 704, 706, 708, 710, 712, 714, 716, 718, 720, 722, 724, 726, 728, 730, 732, 734, 736, 738, 740, 742, 744, 746, 748, 750, 752, 754, 756, 758, 760, 762, 764, 766, 768, 770, 772, 774, 776, 778, 780, 782, 784, 786, 788, 790, 792, 794, 796, 798, 800, 802, 804, 806, 808, 810, 812, 814, 816, 818, 820, 822, 824, 826, 828, 830, 832, 834, 836, 838, 840, 842, 844, 846, 848, 850, 852, 854, 856, 858, 860, 862, 864, 866, 868, 870, 872, 874, 876, 878, 880, 882, 884, 886, 888, 890, 892, 894, 896, 898, 900, 902, 904, 906, 908, 910, 912, 914, 916, 918, 920, 922, 924, 926, 928, 930, 932, 934, 936, 938, 940, 942, 944, 946, 948, 950, 952, 954, 956, 958, 960, 962, 964, 966, 968, 970, 972, 974, 976, 978, 980, 982, 984, 986, 988, 990, 992, 994, 996, 998, 1000, 1002, 1004, 1006, 1008, 1010, 1012, 1014, 1016, 1018, 1020, 1022, 1024, 1026, 1028, 1030, 1032, 1034, 1036, 1038, 1040, 1042, 1044, 1046, 1048, 1050, 1052, 1054, 1056, 1058, 1060, 1062, 1064, 1066, 1068, 1070, 1072, 1074, 1076, 1078, 1080, 1082, 1084, 1086, 1088, 1090, 1092, 1094, 1096, 1098, 1100, 1102, 1104, 1106, 1108, 1110, 1112, 1114, 1116, 1118, 1120, 1122, 1124, 1126, 1128, 1130, 1132, 1134, 1136, 1138, 1140, 1142, 1144, 1146, 1148, 1150, 1152, 1154, 1156, 1158, 1160, 1162, 1164, 1166, 1168, 1170, 1172, 1174, 1176, 1178, 1180, 1182, 1184, 1186, 1188, 1190, 1192, 1194, 1196, 1198, 1200, 1202, 1204, 1206, 1208, 1210, 1212, 1214, 1216, 1218, 1220, 1222, 1224, 1226, 1228, 1230, 1232, 1234, 1236, 1238, 1240, 1242, 1244, 1246, 1248, 1250, 1252, 1254, 1256, 1258, 1260, 1262, 1264, 1266, 1268, 1270, 1272, 1274, 1276, 1278, 1280, 1282, 1284, 1286, 1288, 1290, 1292, 1294, 1296, 1298, 1300, 1302, 1304, 1306, 1308, 1310, 1312, 1314, 1316, 1318, 1320, 1322, 1324, 1326, 1328, 1330, 1332, 1334, 1336, 1338, 1340, 1342, 1344, 1346, 1348, 1350, 1352, 1354, 1356, 1358, 1360, 1362, 1364, 1366, 1368, 1370, 1372, 1374, 1376, 1378, 1380, 1382, 1384, 1386, 1388, 1390, 1392, 1394, 1396, 1398, 1400, 1402, 1404, 1406, 1408, 1410, 1412, 1414, 1416, 1418, 1420, 1422, 1424, 1426, 1428, 1430, 1432, 1434, 1436, 1438, 1440, 1442, 1444, 1446, 1448, 1450, 1452, 1454, 1456, 1458, 1460, 1462, 1464, 1466, 1468, 1470, 1472, 1474, 1476, 1478, 1480, 1482, 1484, 1486, 1488, 1490, 1492, 1494, 1496, 1498, 1500, 1502, 1504, 1506, 1508, 1510, 1512, 1514, 1516, 1518, 1520, 1522, 1524, 1526, 1528, 1530, 1532, 1534, 1536, 1538, 1540, 1542, 1544, 1546, 1548, 1550, 1552, 1554, 1556, 1558, 1560, 1562, 1564, 1566, 1568, 1570, 1572, 1574, 1576, 1578, 1580, 1582, 1584, 1586, 1588, 1590, 1592, 1594, 1596, 1598, 1600, 1602, 1604, 1606, 1608, 1610, 1612, 1614, 1616, 1618, 1620, 1622, 1624, 1626, 1628, 1630, 1632, 1634, 1636, 1638, 1640, 1642, 1644, 1646, 1648, 1650, 1652, 1654, 1656, 1658, 1660, 1662, 1664, 1666, 1668, 1670, 1672, 1674, 1676, 1678, 1680, 1682, 1684, 1686, 1688, 1690, 1692, 1694, 1696, 1698, 1700, 1702, 1704, 1706, 1708, 1710, 1712, 1714, 1716, 1718, 1720, 1722, 1724, 1726, 1728, 1730, 1732, 1734, 1736, 1738, 1740, 1742, 1744, 1746, 1748, 1750, 1752, 1754, 1756, 1758, 1760, 1762, 1764, 1766, 1768, 1770, 1772, 1774, 1776, 1778, 1780, 1782, 1784, 1786, 1788, 1790, 1792, 1794, 1796, 1798, 1800, 1802, 1804, 1806, 1808, 1810, 1812, 1814, 1816, 1818, 1820, 1822, 1824, 1826, 1828, 1830, 1832, 1834, 1836, 1838, 1840, 1842, 1844, 1846, 1848, 1850, 1852, 1854, 1856, 1858, 1860, 1862, 1864, 1866, 1868, 1870, 1872, 1874, 1876, 1878, 1880, 1882, 1884, 1886, 1888, 1890, 1892, 1894, 1896, 1898, 1900, 1902, 1904, 1906, 1908, 1910, 1912, 1914, 1916, 1918, 1920, 1922, 1924, 1926, 1928, 1930, 1932, 1934, 1936, 1938, 1940, 1942, 1944, 1946, 1948, 1950, 1952, 1954, 1956, 1958, 1960, 1962, 1964, 1966, 1968, 1970, 1972, 1974, 1976, 1978, 1980, 1982, 1984, 1986, 1988, 1990, 1992, 1994, 1996, 1998, 2000, 2002, 2004, 2006, 2008, 2010, 2012, 2014, 2016, 2018, 2020, 2022, 2024, 2026, 2028, 2030, 2032, 2034, 2036, 2038, 2040, 2042, 2044, 2046, 2048, 2050, 2052, 2054, 2056, 2058, 2060, 2062, 2064, 2066, 2068, 2070, 2072, 2074, 2076, 2078, 2080, 2082, 2084, 2086, 2088, 2090, 2092, 2094, 2096, 2098, 2100, 2102, 2104, 2106, 2108, 2110, 2112, 2114, 2116, 2118, 2120, 2122, 2124, 2126, 2128, 2130, 2132, 2134, 2136, 2138, 2140, 2142, 2144, 2146, 2148, 2150, 2152, 2154, 2156, 2158, 2160, 2162, 2164, 2166, 2168, 2170, 2172, 2174, 2176, 2178, 2180, 2182, 2184, 2186, 2188, 2190, 2192, 2194, 2196, 2198, 2200, 2202, 2204, 2206, 2208, 2210, 2212, 2214, 2216, 2218, 2220, 2222, 2224, 2226, 2228, 2230, 2232, 2234, 2236, 2238, 2240, 2242, 2244, 2246, 2248, 2250, 2252, 2254, 2256, 2258, 2260, 2262, 2264, 2266, 2268, 2270, 2272, 2274, 2276, 2278, 2280, 2282, 2284, 2286, 2288, 2290, 2292, 2294, 2296, 2298, 2300, 2302, 2304, 2306, 2308, 2310, 2312, 2314, 2316, 2318, 2320, 2322, 2324, 2326, 2328, 2330, 2332, 2334, 2336, 2338, 2340, 2342, 2344, 2346, 2348, 2350, 2352, 2354, 2356, 2358, 2360, 2362, 2364, 2366, 2368, 2370, 2372, 2374, 2376, 2378, 2380, 2382, 2384, 2386, 2388, 2390, 2392, 2394, 2396, 2398, 2400, 2402, 2404, 2406, 2408, 2410, 2412, 2414, 2416, 2418, 2420, 2422, 2424, 2426, 2428, 2430, 2432, 2434, 2436, 2438, 2440, 2442, 2444, 2446, 2448, 2450, 2452, 2454, 2456, 2458, 2460, 2462, 2464, 2466, 2468, 2470, 2472, 2474, 2476, 2478, 2480, 2482, 2484, 2486, 2488, 2490, 2492, 2494, 2496, 2498, 2500, 2502, 2504, 2506, 2508, 2510, 2512, 2514, 2516, 2518, 2520, 2522, 2524, 2526, 2528, 2530, 2532, 2534, 2536, 2538, 2540, 2542, 2544, 2546, 2548, 2550, 2552, 2554, 2556, 2558, 2560, 2562, 2564, 2566, 2568, 2570, 2572, 2574, 2576, 2578, 2580, 2582, 2584, 2586, 2588, 2590, 2592, 2594, 2596, 2598, 2600, 2602, 2604, 2606, 2608, 2610, 2612, 2614, 2616, 2618, 2620, 2622, 2624, 2626, 2628, 2630, 2632, 2634, 2636, 2638, 2640, 2642, 2644, 2646, 2648, 2650, 2652, 2654, 2656, 2658, 2660, 2662, 2664, 2666, 2668, 2670, 2672, 2674, 2676, 2678, 2680, 2682, 2684, 2686, 2688, 2690, 2692, 2694, 2696, 2698, 2700, 2702, 2704, 2706, 2708, 2710, 2712, 2714, 2716, 2718, 2720, 2722, 2724, 2726, 2728, 2730, 2732, 2734, 2736, 2738, 2740, 2742, 2744, 2746, 2748, 2750, 2752, 2754, 2756, 2758, 2760, 2762, 2764, 2766, 2768, 2770, 2772, 2774, 2776, 2778, 2780, 2782, 2784, 2786, 2788, 2790, 2792, 2794, 2796, 2798, 2800, 2802, 2804, 2806, 2808, 2810, 2812, 2814, 2816, 2818, 2820, 2822, 2824, 2826, 2828, 2830, 2832, 2834, 2836, 2838, 2840, 2842, 2844, 2846, 2848, 2850, 2852, 2854, 2856, 2858, 2860, 2862, 2864, 2866, 2868, 2870, 2872, 2874, 2876, 2878, 2880, 2882, 2884, 2886, 2888, 2890, 2892, 2894, 2896, 2898, 2900, 2902, 2904, 2906, 2908, 2910, 2912, 2914, 2916, 2918, 2920, 2922, 2924, 2926, 2928, 2930, 2932, 2934, 2936, 2938, 2940, 2942, 2944, 2946, 2948, 2950, 2952, 2954, 2956, 2958, 2960, 2962, 2964, 2966, 2968, 2970, 2972, 2974, 2976, 2978, 2980, 2982, 2984, 2986, 2988, 2990, 2992, 2994, 2996, 2998, 3000, 3002, 3004, 3006, 3008, 3010, 3012, 3014, 3016, 3018, 3020, 3022, 3024, 3026, 3028, 3030, 3032, 3034, 3036, 3038, 3040, 3042, 3044, 3046, 3048, 3050, 3052, 3054, 3056, 3058, 3060, 3062, 3064, 3066, 3068, 3070, 3072, 3074, 3076, 3078, 3080, 3082, 3084, 3086, 3088, 3090, 3092, 3094, 3096, 3098, 3100, 3102, 3104, 3106, 3108, 3110, 3112, 3114, 3116, 3118, 3120, 3122, 3124, 3126, 3128, 3130, 3132, 3134, 3136, 3138, 3140, 3142, 3144, 3146, 3148, 3150, 3152, 3154, 3156, 3158, 3160, 3162, 3164, 3166, 3168, 3170, 3172, 3174, 3176, 3178, 3180, 3182, 3184, 3186, 3188, 3190, 3192, 3194, 3196, 3198, 3200, 3202, 3204, 3206, 3208, 3210, 3212, 3214, 3216, 3218, 3220, 3222, 3224, 3226, 3228, 3230, 3232, 3234, 3236, 3238, 3240, 3242, 3244, 3246, 3248, 3250, 3252, 3254, 3256, 3258, 3260, 3262, 3264, 3266, 3268, 3270, 3272, 3274, 3276, 3278, 3280, 3282, 3284, 3286, 3288, 3290, 3292, 3294, 3296, 3298, 3300, 3302, 3304, 3306, 3308, 3310, 3312, 3314, 3316, 3318, 3320, 3322, 3324, 3326, 3328, 3330, 3332, 3334, 3336, 3338, 3340, 3342, 3344, 3346, 3348, 3350, 3352, 3354, 3356, 3358, 3360, 3362, 3364, 3366, 3368, 3370, 3372, 3374, 3376, 3378, 3380, 3382, 3384, 3386, 3388, 3390, 3392, 3394, 3396, 3398, 3400, 3402, 3404, 3406, 3408, 3410, 3412, 3414, 3416, 3418, 3420, 3422, 3424, 3426, 3428, 3430, 3432, 3434, 3436, 3438, 3440, 3442, 3444, 3446, 3448, 3450, 3452, 3454, 3456, 3458, 3460, 3462, 3464, 3466, 3468, 3470, 3472, 3474, 3476, 3478, 3480, 3482, 3484, 3486, 3488, 3490, 3492, 3494, 3496, 3498, 3500, 3502, 3504, 3506, 3508, 3510, 3512, 3514, 3516, 3518, 3520, 3522, 3524, 3526, 3528, 3530, 3532, 3534, 3536, 3538, 3540, 3542, 3544, 3546, 3548, 3550, 3552, 3554, 3556, 3558, 3560, 3562, 3564, 3566, 3568, 3570, 3572, 3574, 3576, 3578, 3580, 3582, 3584, 3586, 3588, 3590, 3592, 3594, 3596, 3

C
=

Sorting

Px ① [Insertion sort.]

77	33	44	11	88	22	66	55
77	33	44	11	88	22	66	55
33	77	44	11	88	22	66	55
33	44	77	11	88	22	66	55
11	33	44	77	88	22	66	55
11	33	44	77	88	22	66	55
11	22	33	44	77	88	66	55
11	22	33	44	66	77	88	55
11	22	33	44	55	66	77	88

Procedure Insertionsort

```
set k←1
for k←1 to (n-1)
    set temp ← a[k]
    set j ← k+1
    while temp < a[j] and (j>=0) do
        set a[j+1] ← a[j]
        end while; a[j] ← a[j+1]
    Assign the value of temp to a[j+1]
end; end for;
```

k=1

```
→ for(k=1 ; k<=(n-1); k++)
{
    temp = a[k];
    → for(j=k-1; temp < a[j] and j>=0; j-)
        Insert temp at j
```

2. Non-linear DS - Tree, graph

Accessing of data items is possible in linear fashion
..... one by one sequentially
..... is not

② Selection sort -

(77)	33	44	(11)	88	22	66	55
11	(33)	44	77	88	(22)	66	55
11	22	(44)	77	88	(33)	66	55
11	22	33	(77)	88	(44)	66	55
11	22	33	44	(88)	77	66	(55)
11	22	33	44	55	(77)	(66)	88
11	22	33	44	55	66	(77)	88
11	22	33	44	55	66	77	88

Procedure **MIN** (a, k, n, loc) \rightarrow loc of smallest elmt

```

set min  $\leftarrow a[k]$ , loc  $\leftarrow k$ 
for j  $\leftarrow k+1$  to n
  if min  $> a[j]$ 
    set min  $\leftarrow a[j]$ , loc  $\leftarrow j$ 
  end if
end for;
return;
```

Procedure Selection_Sort

```

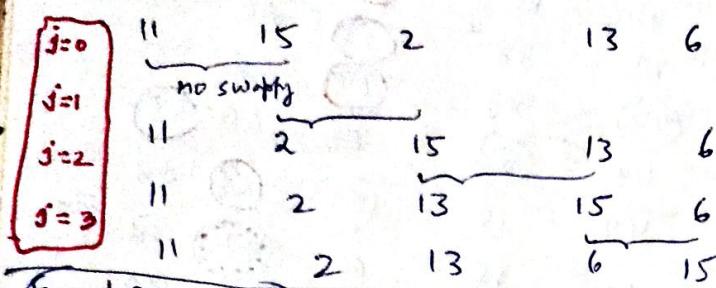
for k  $\leftarrow 1$  to n-1
  call MIN( $a, k, n, loc$ )
  // Now interchange  $a[k]$  and  $a[loc]$ 
  set temp  $\leftarrow a[k]$ 
   $a[k] \leftarrow a[loc]$ 
   $a[loc] \leftarrow temp$ 
end for;
end;
```

③ Bubble sort -

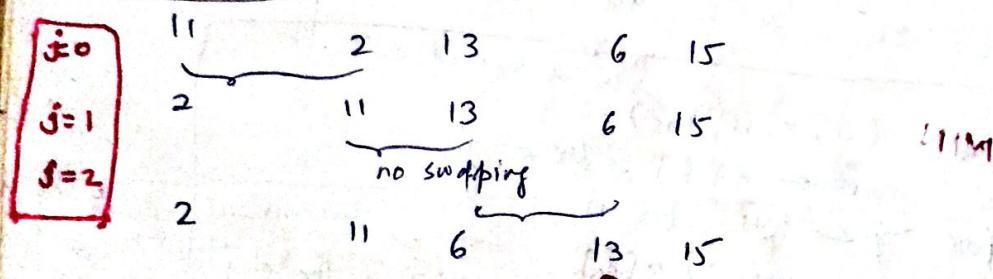
Visited list

11, 15, 2, 13, 6

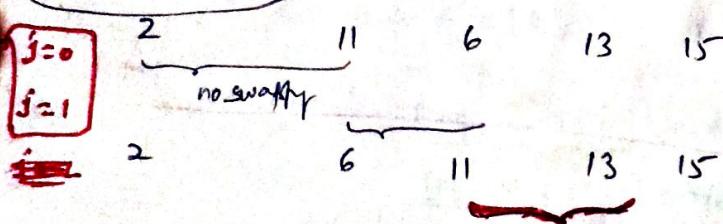
P1. First Pass ($i=0$)



Second Pass ($i=1$)



Third Pass ($i=2$)



(Fourth Pass ($i=3$))



Procedure bubble-sort

for $i \leftarrow 0$ to $j \leq n-1$

~~for~~

~~for $j \leftarrow 0$ to $n-1-i$~~

~~if $A[j] > A[j+1]$~~
~~swap $A[j]$ with $A[j+1]$~~
~~temp $\leftarrow A[j]$~~
 ~~$A[j] \leftarrow A[j+1]$~~
 ~~$A[j+1] \leftarrow temp$~~

~~end if~~

~~end for~~

~~end;~~

$i = n-i$ ~~for~~

$\left\{ \begin{array}{l} n-0 = n \\ n-1 = n-1 \\ n-2 = n-2 \\ \vdots \\ (n-1)-(n-1) = 0 \end{array} \right.$

for($i=0; i \leq n-1; i++$)

{ for($j=$

$\left\{ \begin{array}{l} \text{if } A[j] > A[j+1] \\ \text{swap } A[j] \text{ with } A[j+1] \end{array} \right.$

(4) Merge sort

2. ~~Non comparison sorting~~ It is possible to linearly compare all data items, one by one sequentially. Unsorted list - 85, 76, 46, 92, 30, 41, 12, 19, 93, 3, 50, 11 merge each pair of elements to obtain the list of sorted pairs

76 | 85

46 | 92

30 | 41

12 | 19

3 | 93

11 | 50

merge each pair of pairs to obtain the lists of sorted elements

46 | 76 | 85 | 92

12 | 19 | 30 | 41

3 | 11 | 50 | 93

again merge the two subarrays to get two lists

12 | 19 | 30 | 41 | 46 | 76 | 85 | 92

3 | 11 | 50 | 93

merging the above two lists, we get

3 | 11 | 12 | 19 | 30 | 41 | 46 | 50 | 76 | 85 | 92 | 93

Procedure Merge sort

loop until last merge

 merge each pair of elements to obtain the list
 of sorted elements

end loop;

end;

C) [Heap sort]

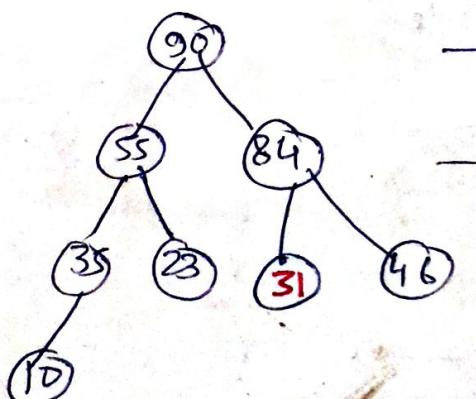
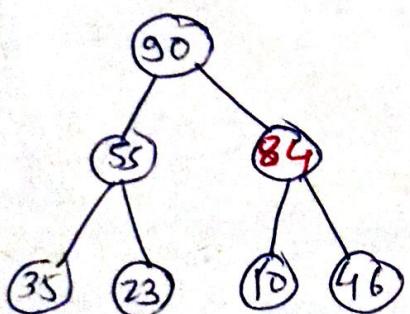
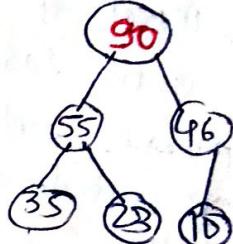
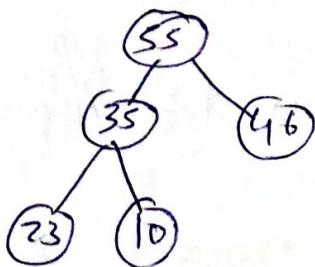
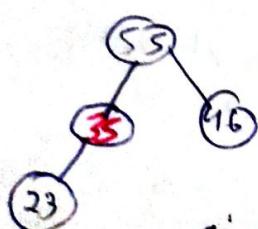
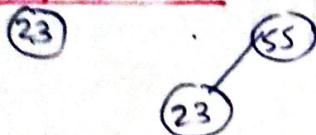
P) min heap (root - min) method
Ascending order list is obtained

max heap (root - max) method
Descending order list is obtained

Eg. max heap method

23, 55, 46, 35, 10, 90, 84, 31

Heap creation



$l=1$
 $l=2$
 $l=3$
 $l=4$

Heap sort. (Delete max of level i)

90	84	55	46	35	31	23	10
----	----	----	----	----	----	----	----

Procedure Heap-Sort

Create heap (max) for descending from max for ascending
Loop $i \leftarrow 1$ to l // l is the no. of levels

Delete max of level i

Loop $j = 1$ to last node at level i

Delete max of level i

end loop 1;

end loop 2;

end;

end on

end DFS

Radix

348, 1

max digit

Part I (for

348

143

36

42

543

128

Part II (for

361

321

143

423

543

366

348

538

128

Part III (for

321

423

128

538

143

543

348

361

366

Radix sort / Bucket sort

tree, graph
can be possible in linear fashion
is not
sequentially

348, 143, 361, 423, 538, 128, 321, 543, 366
max digits = 3, so 3 passes

(so that in each pass we get
sorted list at a particular place value)

Pass I (for ones places)

IP	0	1	2	3	4	5	6	7	8	9
348										348
143										143
361				361						
423										423
538										538
128										128
321			321							
543										543
366										366

Pass II (for tens places)

IP	0	1	2	3	4	5	6	7	8	9
361	348									361
321	143									
143	361									
423	423									
538	538									
543	128									
366	321									
348	543									
366	366									
538										538
128										128

Pass III (for hundreds place)

IP	0	1	2	3	4	5	6	7	8	9
321				321						
423					423					
128										
538										538
143										
543										543
348					348					
366					321					
366						366				

~~Sort with sort~~

Sorted List - 128, 143, 321, 348, 361, 366, 423, 538, 543

Procedure Radix sort
for pause

loop until max no of digits \leq of the no in the list

In Pass I, unit digits are sorted, then collect the nos & reinput
and reinput for next pass

In Pass II, tens digits are sorted, then collect the nos & reinput
and reinput for next pass

In Pass III, Hundreds digits are sorted, then collect the nos & reinput
and reinput for next pass

and so on.

end loop;

end;

Quick Sort

(Partition Exchange Tech-) (an app' of stacks) is not
(divide & conquer meta) is not

- circle first and last no
- Begin with last no, scan the list from right to left,
compare each no with first no and
stop at the no which is less than first no
- Interchange the obtained no with first no
- Now circle these interchanged nos and name as ^{new}last
- Begin with this new first no, scan the list from left to right,
compare each no with ^{new}last no and
stop at the no which is greater than ^{new}last no
- Interchange the obtained no with new last no
- Now circle these interchanged nos and name as ^{new}last
- Repeat the above ~~first~~ steps until you get pivot element
(pivot element would be the element whose left-side elements are lesser than pivot and right-side elements are greater than pivot)
- In this way we get 2 sublists at the both sides of pivot.
- Above reduction steps are repeated with each sublist containing 2 or more elements.
(by solving one sublist at a time)

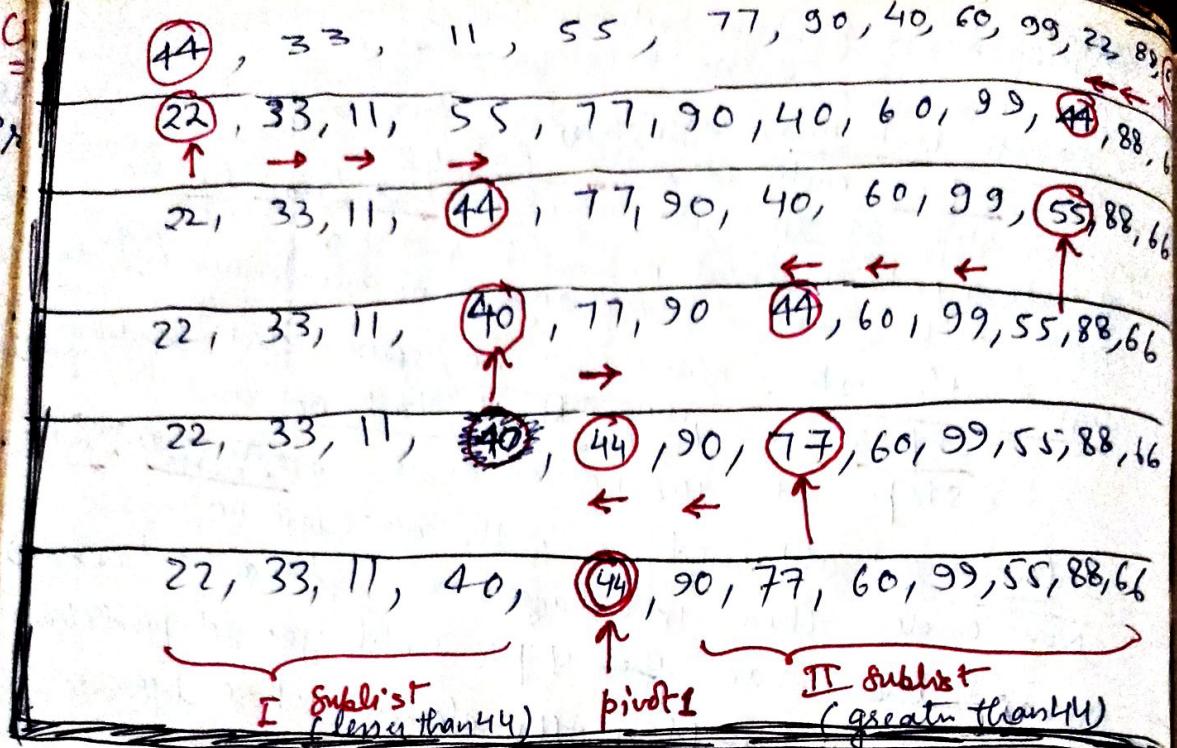
Compare
including itself

Obtained no with
last no

Compare with last no
stop at no $>$ last no
Interchange & circle &
rename 1st & last no

Compare
including itself

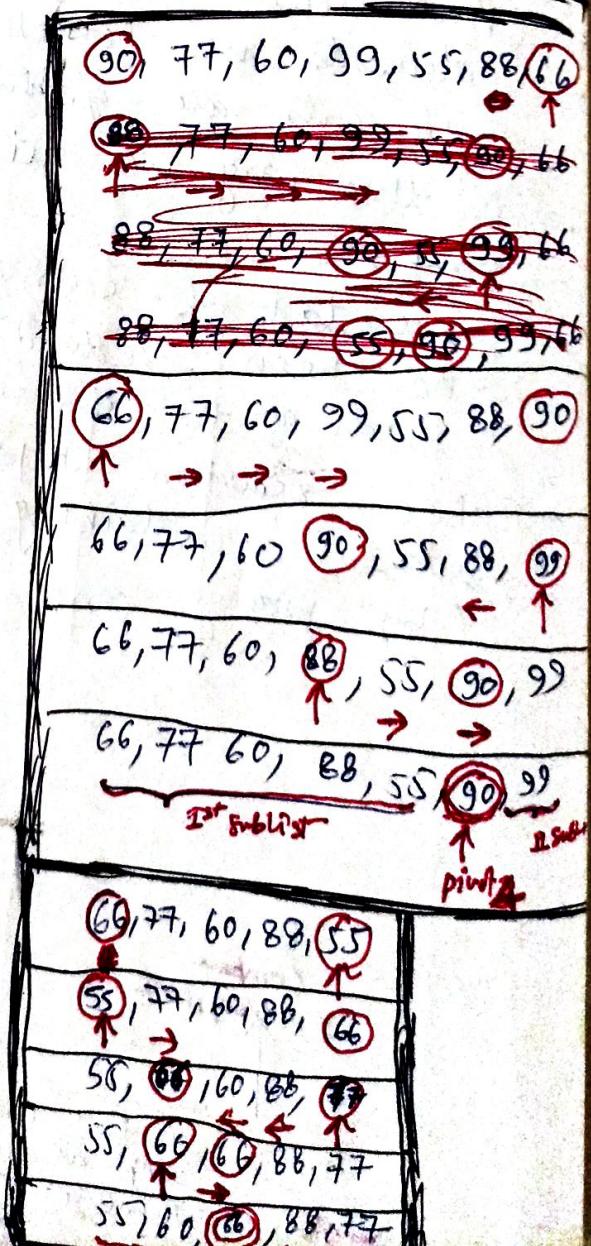
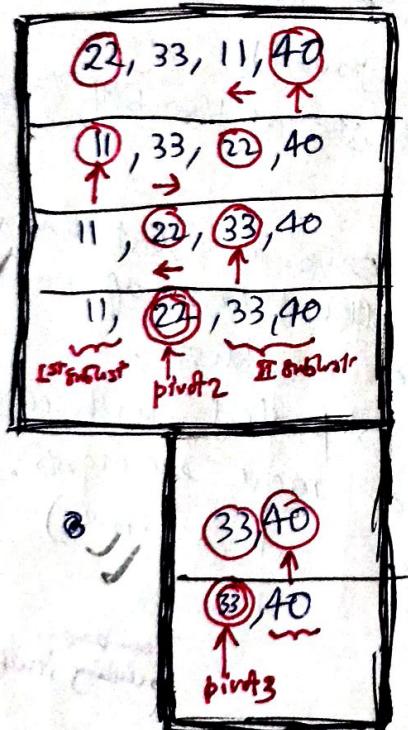
← compare with 1st no
stop at no \leq 1st no
Interchange & circle &
rename 1st & last no
obtained no with 1st no



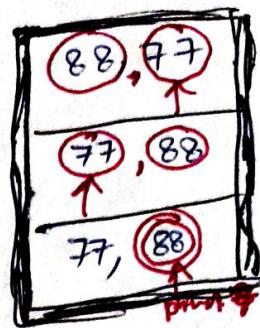
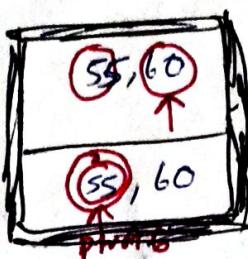
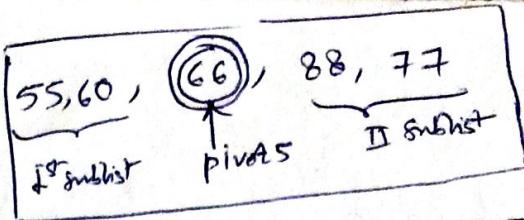
Histogram

sorted

11, 22



~~Warning~~



sorted list \Rightarrow

11, 22, 33, 40, 44, 55, 60, 66, 77, 88, 90, 99

Ref: Schaum's Series
Page 6.14
Section 6.6

2. Non-Linear ~~use, just~~ items. Is possible in linear fashion sequentially

Hashing Searching process of finding element in the list

Linear / Sequential Search - $O(n)$

\Rightarrow elements are searched one by one sequentially.

Procedure SEQ-SEARCH

$c \leftarrow 0$

loop $i \leftarrow l$ to u

if $item = k[i]$

$c \leftarrow 1$

print 'Item found at i position' & exit

end if;

end loop;

if $c = 0$

print 'Item not found'

endif;

end;

Binary search - $O(\log_2 n)$

\Rightarrow elements are searched in a sorted / ordered list.

Procedure BIN-SEARCH

$c \leftarrow 0$

loop $i \leftarrow l$ to u

$m \leftarrow (l+u)/2$

~~case~~

~~if item < k[m]~~

if ($item = k[m]$)

$c \leftarrow 1$

print 'Item found at m position' & exit

else if ($item > k[m]$)

$l \leftarrow m+1$

else

$u \leftarrow m-1$

end loop;

if $c = 0$

print 'Item not found'

endif;

end;

[Hashing] O(1)

Linear & Binary search are based on comparison of key.
③ Hashing is a search tech in which there are no unnecessary comparisons of keys.
Thus, in hashing no. of comparisons are minimized.
Hashing is an approach in which location of desired record is ~~exp~~ obtained in single access, this location depends only on the given key (not on other keys).

[Hash Table]

It is a data structure where we store a key value after applying the hash fn.

[Hash function]

It is a fn that takes key as i/p and forms it into a hash index.

$$H : K \rightarrow M$$

↑ | ↗
Hash fn set of keys set of many addresses

[Hash collision]

Sometimes, Hash fn H may not yield distinct values. It is possible that two different keys k_1 and k_2 will yield the same hash address. This situation is called Hash collision.

[Types of Hash fn]

Division Remainder Method
Mid Square Method
Folding Method

We should consider following things while solving a hash fn.

- ① H should be very easy and quick to compute
- ② H should result ~~with~~ in less no of collisions

Ex: common key mapping

② Mid

Ex: common m=

COR

m=

K

T

f

To prop

1. Linear DS - Array, linked list, stack, Queue
 Non-linear DS - Tree, graph ... linear fashion
 usually not

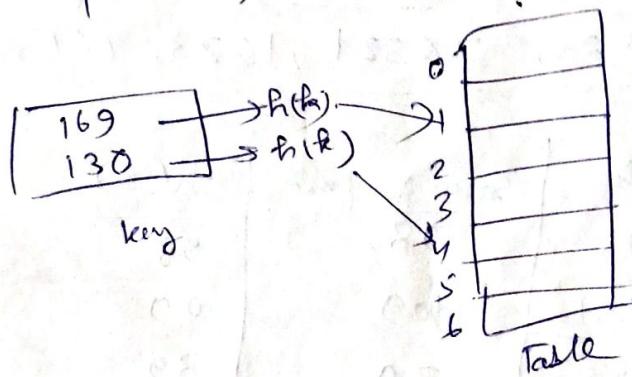
① Division Remainder Method

$$h(k) = k \bmod m$$

↓
key ↓ slot
point no (minimum no of collisions)

Ex: consider a hash table with 7 slots i.e. $m=7$
 then hash fn $h(k) = k \bmod m$ will map the
 key 169 to slot 1 ~~slot 1~~ and key 130 will
 mapped to slot 4.
 $h(169) = 169 \bmod 7 = 1$

$$\text{similar } h(130) = 130 \bmod 7 = 4$$



② Mid Square Method

$$h(k) = f(k)$$

↓
 obtained by deleting digits
 from both sides of k^2

Ex: consider a hash table with 50 slots i.e.

$m=50$ and key values $k = 1632, 1739, 3123$

$$k = 1632$$

$$1739$$

$$3123$$

$$k^2 = 2663424$$

$$3024121$$

$$9753129$$

$$h(k) = \underline{\underline{34}}$$

$$\underline{\underline{41}}$$

$$\underline{\underline{31}}$$

To properly implement this, the same k^2 must be used for all the keys

③ folding Method

key k is partitioned into a no. of parts k_1, k_2, \dots, k_r

where each part (except possibly the last) has the same no. of digits as the required address.
Then the parts are added together, ignoring the last carry.

$$H(k) = k_1 + k_2 + \dots + k_r$$

Ex. Consider a hash table with 100 slots i.e. $m=100$ and key values
 $k = 7325, 76321, 1623, 7613$

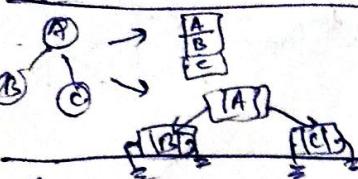
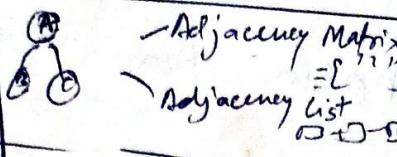
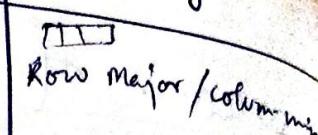
k	Parts	Sum of parts	$h(k)$
7325	73, 25	98	98
76321	76, 32, 1	109	09
1623	16, 23	39	39
7613	76, 13	89	89

complexity of sorting Algo's -

algo	worst case	Avg case
Bubble	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$
Quick	$\frac{n(n+3)}{2} = O(n^2)$	$1 + n \log n = O(n \log n)$
Heap	$3n \log n = O(n \log n)$	$3n \log n = O(n \log n)$
Insertion	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{4} = O(n^2)$
Selection	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$
Merge	$n \log n = O(n \log n)$	$n \log n = O(n \log n)$
Radix (bucket)	$O(n^2)$	$O(n \log n)$

Linear search = $O(n)$

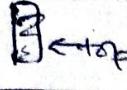
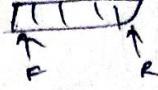
Binary search = $O(\log n)$

CE	Tree	Graph	Array
Pr	 Static	 Adjacency Matrix Adjacency List	 Row Major / Column Major
	DYNAMIC	STATIC	

Trans	Inorder Preorder Postorder	Trans	DFS BFS	Insert Del^n
Heapsort etc.		colorization effect Airth Path Accm Matrix (O.S.)		Tables mathematics matrix

Thread B.T. Heapsort AVL B Tree / B+ Tree / BST	MST (Min Span Tree) Kruskal Alg	Sparse Matrix
	but a graph is not necessarily a tree	

particular orgⁿ of data is called DS.

Linked List	Stack (LIFO)	Queue (FIFO)	
	 static Dynamic	 static Dynamic	Representation Implementation
Insert ⁿ Del ⁿ	PUSH POP	Insert ⁿ Del ⁿ	Operations
Dynamic Representation & Implementation	① Conversion of Infix to Prefix & Postfix ② Evaluation of Postfix	CPU scheduling (OS) Relayway Rec ⁿ Printer	apps
malloc() calloc() realloc() free() structure	top increases on insert & decreases on delete	R increases on insert (no effect on F) F increases on del ⁿ (no effect on R)	(Extra)
	PUSH POP	circular DE-Que Priority Q	
			already studied