A1.2) The sequence of moves required to move 8 disks from source to destination using the above algorithm is given below.

My code is based on recursion where we first take the no. of disks as input from user and then solve it using 2 temporary pegs.

In this we first send (n-2) disks to a temporary peg and using the other pegs send the 2 disks left to the destination and again hit the function where it sends remaining (n-2) disks to the destination.
Base case is when we either reach zero or one disk.

In case input = 8.
It first executes function call of taking 6 disks from T_1 to T_2 which again hits function call taking 4 disks from T_1 to T_4 again function call of 2 disks from T_1 to T_2 which in next function call of zero disks hits base case & returns then gets completed in opposite order moving 2 disks from T_1 to T_2, then 4 disks to T_4 then 6 to T_2 & 8 to T_4 completing the tower of hanoi.

The third peg was used for solving the 2 disks left behind.

Output of the code:-

>> Enter the number of disks : 8

>> The sequence of steps are:-

1. T1 -> T3
2. T1 -> T2
3. T3 -> T2
4. T1 -> T3
5. T1 -> T4
6. T3 -> T4
7. T2 -> T1
8. T2 -> T4
9. T1 -> T4
10. T1 -> T3
11. T1 -> T2

12. T3 -> T2
13. T4 -> T1
14. T4 -> T3
15. T1 -> T3
16. T4 -> T1
17. T4 -> T2
18. T1 -> T2
19. T3 -> T4
20. T3 -> T2
21. T4 -> T2
22. T1 -> T3
23. T1 -> T4
24. T3 -> T4
25. T2 -> T1
26. T2 -> T4
27. T1 -> T4
28. T2 -> T1
29. T2 -> T3
30. T1 -> T3
31. T4 -> T2
32. T4 -> T3
33. T2 -> T3
34. T2 -> T1
35. T2 -> T4
36. T1 -> T4
37. T3 -> T2
38. T3 -> T1
39. T2 -> T1
40. T3 -> T2
41. T3 -> T4
42. T2 -> T4
43. T1 -> T3
44. T1 -> T4
45. T3 -> T4

A.3) The outcome from the typical tower of hanoi is given below.
Calculating time complexity of my code:

$$T(n) = \underbrace{2T(n-2)}_{\text{As 2 times function call for } (n-2).} + \underbrace{c}_{\text{for const. pout operations}}$$

My Code :

$$T(n) = 2T(n-2) + c$$
$$= 2(2T(n-4) + c) + c$$
$$= 4T(n-4) + 3c$$
$$= 4(2T(n-6) + c) + 3c$$
$$= 8T(n-6) + 7c$$
$$\vdots \quad n-2K = 1 \Rightarrow \boxed{K = \frac{n-1}{2}}$$
$$= 2^K\, T(\underbrace{n-2K}) + (2^K - 1)c$$
$$= 2^{\frac{n-1}{2}}\, T(1) + (2^{\frac{n-1}{2}} - 1)c$$
$$= 2^{n/2 - 1/2} + (2^{\frac{n}{2}-\frac{1}{2}})c - c$$
$$= \frac{2^{n/2}}{2^{1/2}}(1+c) - c$$
$$\Downarrow \le 2^{n/2}(1+c)$$
$$\boxed{T(n) = O(2^{n/2})}$$

Time complexity of my code is smaller than that of typical one so it is better in regards of time

Typical Tower of Hanoi

$$T(n) = 2T(n-1) + c$$
$$T(n) = 4T(n-2) + 3c$$
$$T(n) = 8T(n-3) + 7c$$
$$\vdots \quad n-K = 1 \Rightarrow \boxed{K = n-1}$$
$$T_n = 2^K\, T(\underbrace{n-K}) + (2^K - 1)c$$
$$= 2^{n-1}T(1) + (2^{n-1} - 1)c$$
$$= \frac{2^n}{2}(1+c) - c$$
$$\Downarrow \le 2^n(1+c)$$
$$\boxed{T_n = O(2^n)}$$

Typical Tower of Hanoi output with A as source rod and C as destination rod:-

Number of disks : 8

1. A -> B
2. A -> C
3. B -> C
4. A -> B
5. C -> A
6. C -> B
7. A -> B
8. A -> C
9. B -> C
10. B -> A
11. C -> A
12. B -> C
13. A -> B
14. A -> C
15. B -> C
16. A -> B
17. C -> A
18. C -> B
19. A -> B
20. C -> A

21. B -> C
22. B -> A
23. C -> A
24. C -> B
25. A -> B
26. A -> C
27. B -> C
28. A -> B
29. C -> A
30. C -> B
31. A -> B
32. A -> C
33. B -> C
34. B -> A
35. C -> A
36. B -> C
37. A -> B
38. A -> C
39. B -> C
40. B -> A
41. C -> A
42. C -> B
43. A -> B
44. C -> A
45. B -> C
46. B -> A
47. C -> A
48. B -> C
49. A -> B
50. A -> C
51. B -> C
52. A -> B
53. C -> A
54. C -> B
55. A -> B
56. A -> C
57. B -> C
58. B -> A
59. C -> A
60. B -> C
61. A -> B
62. A -> C
63. B -> C
64. A -> B
65. C -> A
66. C -> B
67. A -> B
68. C -> A

69. B -> C
70. B -> A
71. C -> A
72. C -> B
73. A -> B
74. A -> C
75. B -> C
76. A -> B
77. C -> A
78. C -> B
79. A -> B
80. C -> A
81. B -> C
82. B -> A
83. C -> A
84. B -> C
85. A -> B
86. A -> C
87. B -> C
88. B -> A
89. C -> A
90. C -> B
91. A -> B
92. C -> A
93. B -> C
94. B -> A
95. C -> A
96. C -> B
97. A -> B
98. A -> C
99. B -> C
100. A -> B
101. C -> A
102. C -> B
103. A -> B
104. A -> C
105. B -> C
106. B -> A
107. C -> A
108. B -> C
109. A -> B
110. A -> C
111. B -> C
112. A -> B
113. C -> A
114. C -> B
115. A -> B
116. C -> A

117. B -> C
118. B -> A
119. C -> A
120. C -> B
121. A -> B
122. A -> C
123. B -> C
124. A -> B
125. C -> A
126. C -> B
127. A -> B
128. A -> C
129. B -> C
130. B -> A
131. C -> A
132. B -> C
133. A -> B
134. A -> C
135. B -> C
136. B -> A
137. C -> A
138. C -> B
139. A -> B
140. C -> A
141. B -> C
142. B -> A
143. C -> A
144. B -> C
145. A -> B
146. A -> C
147. B -> C
148. A -> B
149. C -> A
150. C -> B
151. A -> B
152. A -> C
153. B -> C
154. B -> A
155. C -> A
156. B -> C
157. A -> B
158. A -> C
159. B -> C
160. B -> A
161. C -> A
162. C -> B
163. A -> B
164. C -> A

165. B -> C
166. B -> A
167. C -> A
168. C -> B
169. A -> B
170. A -> C
171. B -> C
172. A -> B
173. C -> A
174. C -> B
175. A -> B
176. C -> A
177. B -> C
178. B -> A
179. C -> A
180. B -> C
181. A -> B
182. A -> C
183. B -> C
184. B -> A
185. C -> A
186. C -> B
187. A -> B
188. C -> A
189. B -> C
190. B -> A
191. C -> A
192. B -> C
193. A -> B
194. A -> C
195. B -> C
196. A -> B
197. C -> A
198. C -> B
199. A -> B
200. A -> C
201. B -> C
202. B -> A
203. C -> A
204. B -> C
205. A -> B
206. A -> C
207. B -> C
208. A -> B
209. C -> A
210. C -> B
211. A -> B
212. C -> A

213. B -> C
214. B -> A
215. C -> A
216. C -> B
217. A -> B
218. A -> C
219. B -> C
220. A -> B
221. C -> A
222. C -> B
223. A -> B
224. A -> C
225. B -> C
226. B -> A
227. C -> A
228. B -> C
229. A -> B
230. A -> C
231. B -> C
232. B -> A
233. C -> A
234. C -> B
235. A -> B
236. C -> A
237. B -> C
238. B -> A
239. C -> A
240. B -> C
241. A -> B
242. A -> C
243. B -> C
244. A -> B
245. C -> A
246. C -> B
247. A -> B
248. A -> C
249. B -> C
250. B -> A
251. C -> A
252. B -> C
253. A -> B
254. A -> C
255. B -> C

A2.2) In iterative merge sort using same merge function of merge sort.

Lets take input array as [10, 45, 12, 32, 8, 7, 1]

In iterative code:

First length is 1 which is the size of 2arrays to be merged. it starts with 0 index so first the first two numbers are merged ~~which~~ which ~~gives~~ sorts first 2 elements then start changes to next index numbers to be sorted which is third index similarly sorting it.

(8, 7) changes to

Sorting order: 10,45 → 12,32 → 7, 8 → 1 (as only element)

We then double the length and sort 2 arrays

10,45 & 12,32 | 7,8 & 1

resulting in 10, 12, 32, 45 | 1, 7, 8

Then length double again and above 2 sorted arrays merged to form => 1, 7, 8, 10, 12, 32, 45

Output after sorting pair of elements:

Length 1 :-

10 45 12 32 8 7 1	sorted elements: 10,45 (no change right order)
10 45 12 32 8 7 1	sorted elements: 12,32 (no change right order)
10 45 12 32 7 8 1	sorted elements: 8,7 -> 7,8
10 45 12 32 7 8 1	sorted element: 1 (no change)

Length 2 :-

10 12 32 45 7 8 1	sorted elements: 10,45, 12, 32 -> 10,12,32, 45
10 12 32 45 1 7 8	sorted elements: 7, 8,1 -> 1,8,7

Length 4 :-

1 7 8 10 12 32 45	sorted whole list;

A3) The sequence of elements are in the order given below.

$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	$\boxed{4}$	$\boxed{5}$	$\boxed{6}$
n^{-1}	$(\log(n))/n$	$n^{-1/2}$	$(2^2)^{100}$	$\log_{10} n$	$\log_2 n$
$\boxed{7}$	$\boxed{8}$	$\boxed{9}$	$\boxed{10}$	$\boxed{11}$	$\boxed{12}$
$2n$	$3n$	$n2^{100}$	$\log(n!)$	$n \log n$	$\binom{n}{64}$
$\boxed{13}$	$\boxed{14}$	$\boxed{15}$	$\boxed{16}$	$\boxed{17}$	$\boxed{18}$
n^{64}	n^{65}	$2.1^{\sqrt{n}}$	2^n	2^{n+1}	$n2^n$
$\boxed{19}$	$\boxed{20}$	$\boxed{21}$	$\boxed{22}$	$\boxed{23}$	$\boxed{24}$
3^n	2^{2n}	4^n	$n!$	n^n	2^{2^n}

Indexing done by block where below element is of index i