

Size of
each block is k
↓
Sort each block
using insertion
sort

arr

1	2	3	4	8	10	15	20
0	1	2	3	4	5	6	7

Diagram illustrating an array 'arr' with 8 elements. The elements are 1, 2, 3, 4, 8, 10, 15, and 20. The indices are 0 through 7. A blue arrow points from index 0 to index 3, and another blue arrow points from index 4 to index 7. A red circle highlights the elements 15 and 20 at indices 6 and 7.

Steps of Merge

i → ~~0~~ 1 2 3 4

j → ~~4~~ 5 6 7 8

k → ~~0~~ 1 2 3 4 5 6 7 8

result

0	1	2	3	4	5	6	7
1	2	3	4	8	10	15	20

Diagram illustrating the result array. The elements are 1, 2, 3, 4, 8, 10, 15, and 20, stored at indices 0 through 7.

start 1 → 0

end 1 → 3

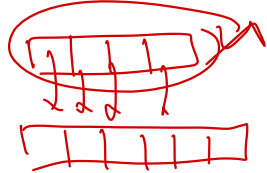
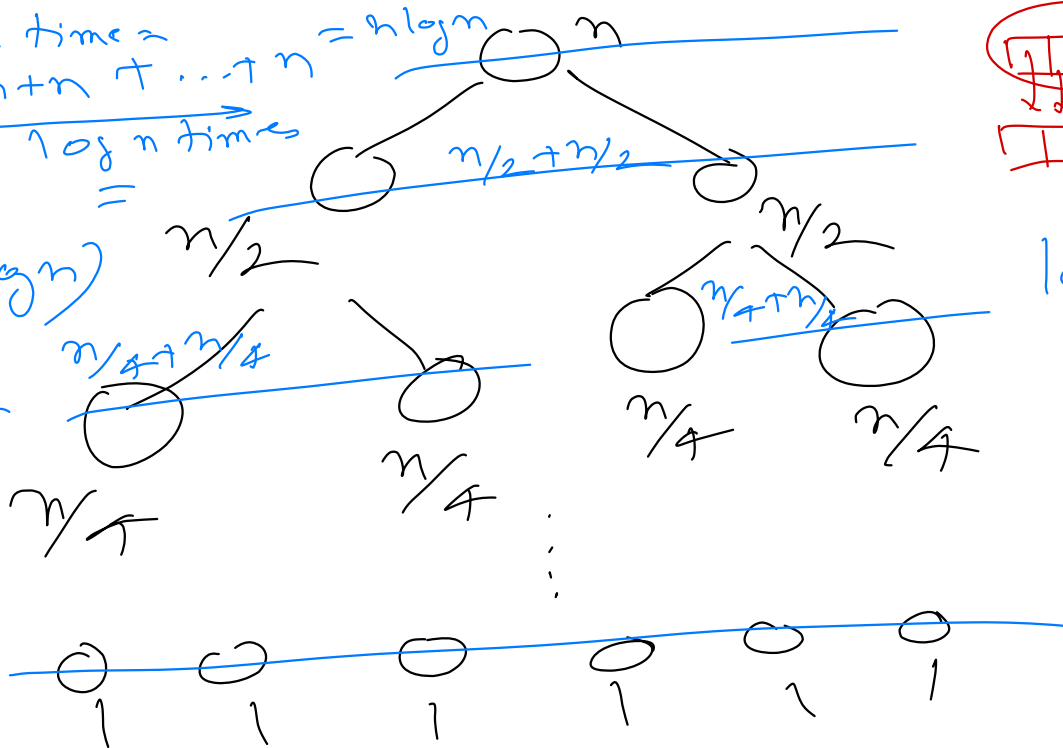
start 2 → 4

end 2 → 7

Total time =
 $n + n + n + \dots + n = n \log n$
 $\xleftarrow{\log n \text{ times}}$
 $n =$

$O(n \log n)$

$n =$



$\log n$
 levels

Space Complexity

Selection Sort \Rightarrow $j, \text{max}, i = 3$
 $\approx O(1)$

Merge $\Rightarrow i, j, k$, result array of size
some elements
of two sorted arrays
of size n & m
 \Downarrow
Size $= n + m$
 $O(n)$

Quick Sort \Rightarrow $\text{pivot}, \text{left}, \text{right}$
 $\approx O(1)$
 $O(\log n)$

Merge Sort $\rightarrow \log n$ for system stack
 for recursive calls
 $n \neq \text{Merge.}$

$$O(n + \log n)$$

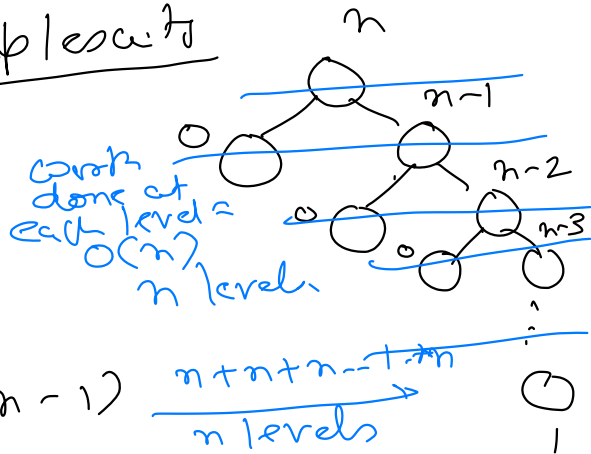
Worse Case time complexity

$[\textcircled{1} \quad 2 \quad 3 \quad 4]$
 $[] \quad 1 \quad [\textcircled{2} \quad 3 \quad 4]$
 $1 \quad [] \quad 2 \quad [3 \quad 4]$

Quick Sort

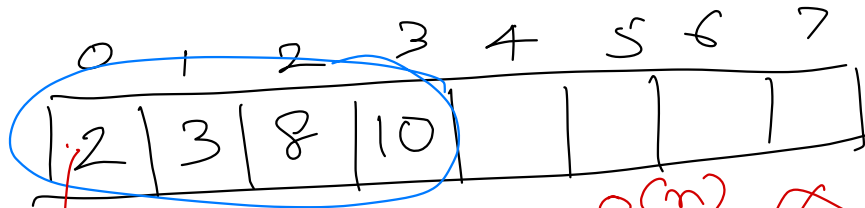
$$O(n^2)$$

$$(n-1) \frac{n + n + n + \dots + n}{n \text{ levels}}$$

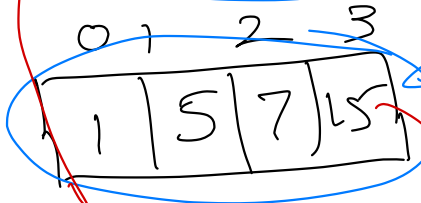


Sort a linked list \Rightarrow Merge Sort

External Merge Sort \Rightarrow Sorting data stored in file.



$n = 8$



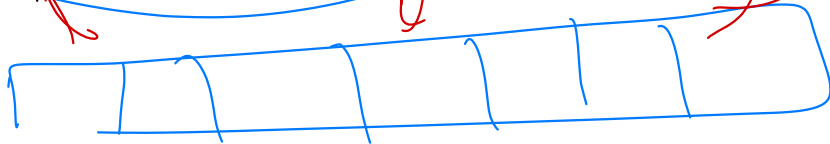
arr

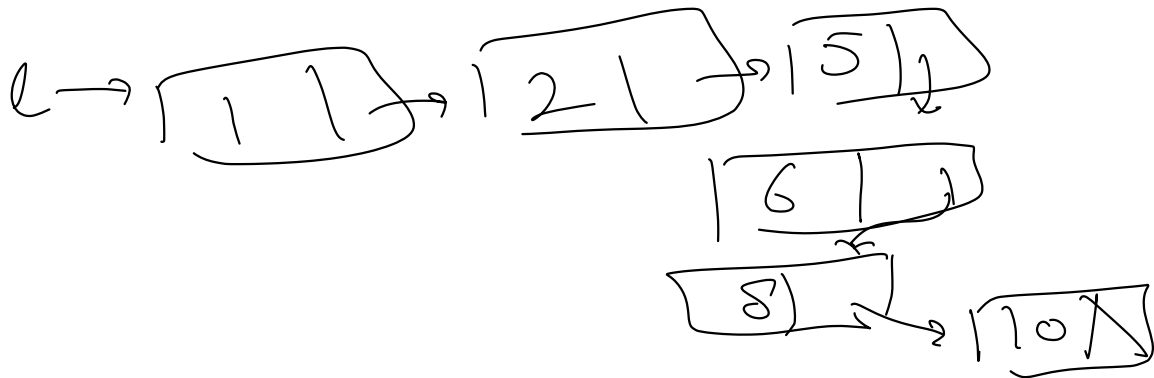
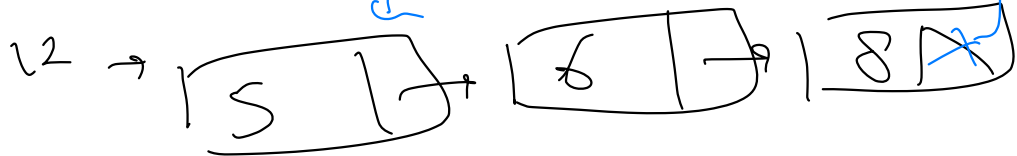
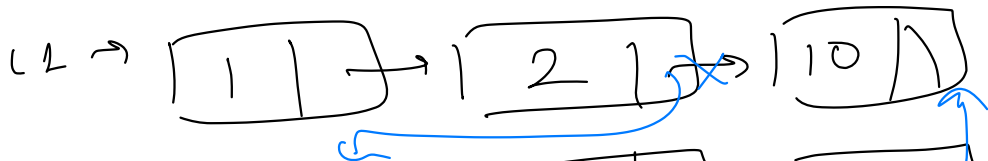
$O(n)$

\leftarrow result

Inplace
Merging
i.e. not using
temp array
S.C. - $O(1)$

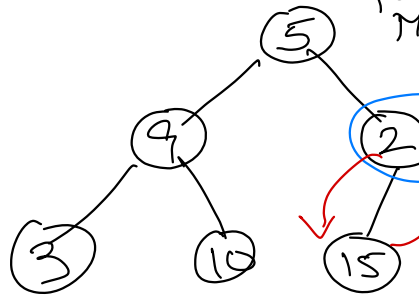
\leftarrow temp



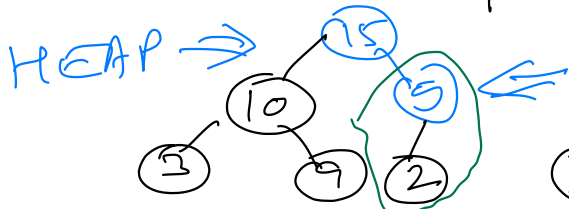


Heap Sort

Heap Property \Rightarrow Parent's data $>$ MAX(child's data)
for MAX HEAP

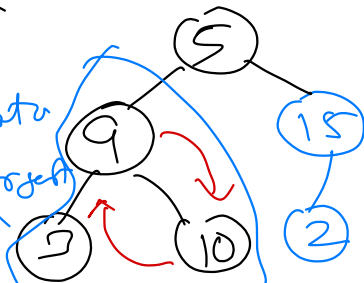


This Tree \neq Heap

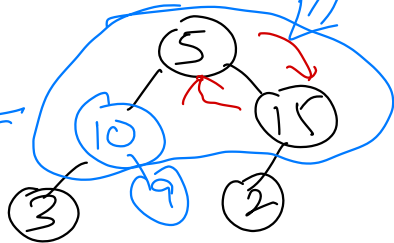


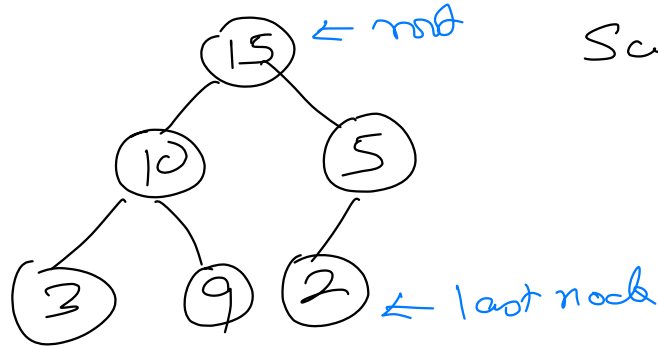
Swap data
Parent with
child

Heap Prop
NOT meet

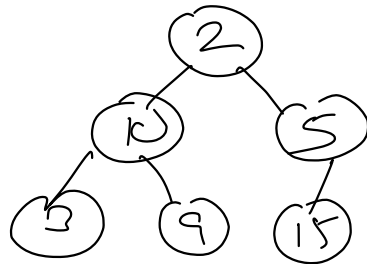


NOT A HEAP



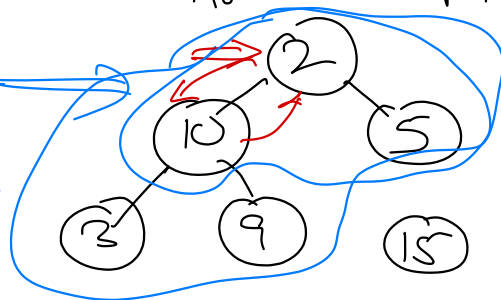
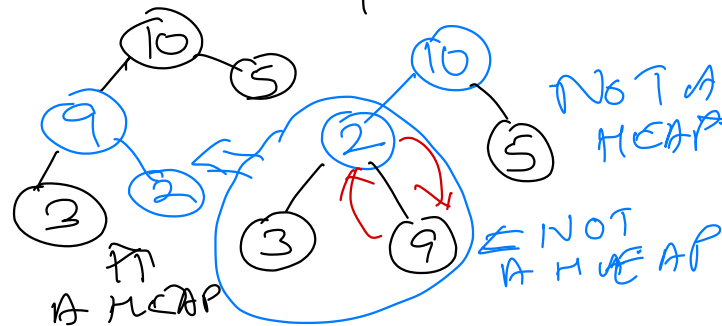


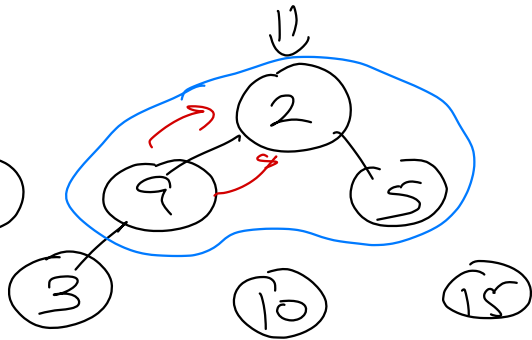
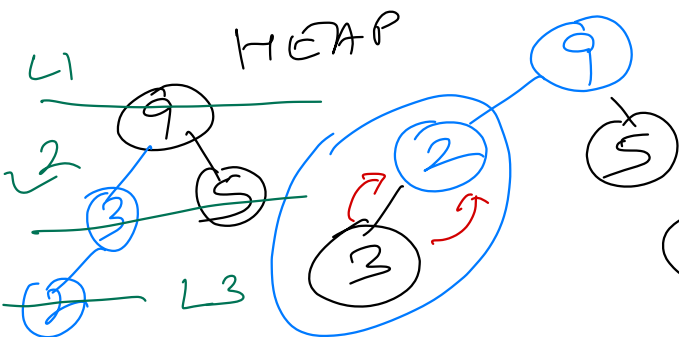
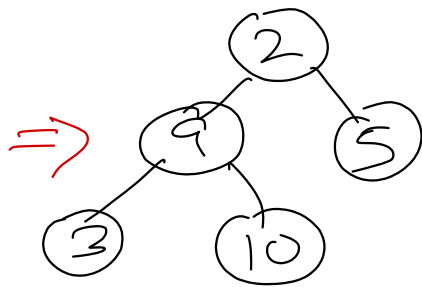
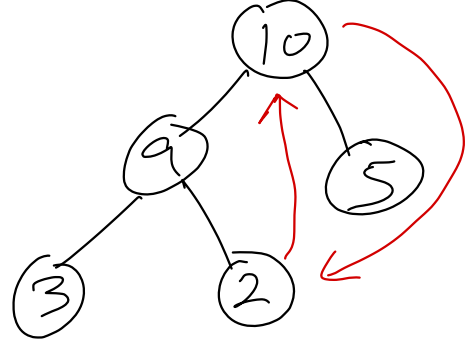
Swap not with last node.



Heap

Remove last node from heap/rec.





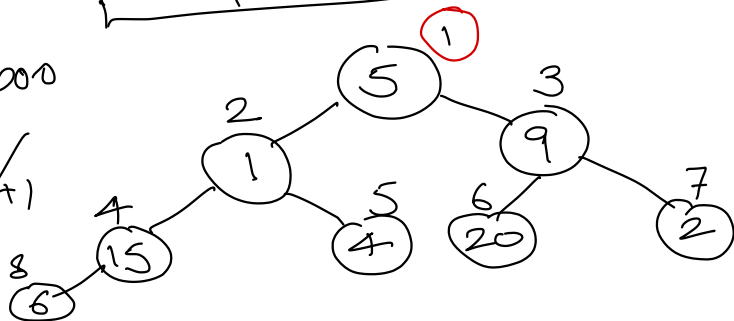
Max Heap \Rightarrow sort elements in ascending order
 Min Heap \Rightarrow sort elements in descending order

Property for MIN HEAP

\Downarrow
Parent's data < MIN (child's data)

	1	2	3	4	5	6	7	8
$i^{\text{th}} \text{ elem}$ (child / rchild) $2xi$ $2xi+1$	5	1	9	15	4	20	2	6

$n/2$ pos
 $2 \times \frac{n}{2}$
 \Downarrow
 n
~~$2 \times \frac{n}{2} + 1$
 \Downarrow
 $n+1$~~



n elements
 last parent
 node = $n/2$

Heap Sort (arr, n)

ARRAY STARTS AT 1

↓
ROOT_{st}
= 1st
elem

1. for ($i = n/2$; $i \geq 1$; $--i$)
 1.1 convertToHeap(arr, i, n)

2. for ($i = n$; $i > 1$; $--i$)
 2.1 Swap root element with
 last node/element
 // Swap (1st element, i^{th} element)
 2.2 convertToHeap(arr, 1, i)

Convert To Heap(arr, root, n)

0.

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

13.

14.

15.

16.

17.

18.

19.

20.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.

31.

32.

33.

34.

35.

36.

37.

38.

39.

40.

41.

42.

43.

44.

45.

46.

47.

48.

49.

50.

51.

52.

53.

54.

55.

56.

57.

58.

59.

60.

61.

62.

63.

64.

65.

66.

67.

68.

69.

70.

71.

72.

73.

74.

75.

76.

77.

78.

79.

80.

81.

82.

83.

84.

85.

86.

87.

88.

89.

90.

91.

92.

93.

94.

95.

96.

97.

98.

99.

100.

101.

102.

103.

104.

105.

106.

107.

108.

109.

110.

111.

112.

113.

114.

115.

116.

117.

118.

119.

120.

121.

122.

123.

124.

125.

126.

127.

128.

129.

130.

131.

132.

133.

134.

135.

136.

137.

138.

139.

140.

141.

142.

143.

144.

145.

146.

147.

148.

149.

150.

151.

152.

153.

154.

155.

156.

157.

158.

159.

160.

161.

162.

163.

164.

165.

166.

167.

168.

169.

170.

171.

172.

173.

174.

175.

176.

177.

178.

179.

180.

181.

182.

183.

184.

185.

186.

187.

188.

189.

190.

191.

192.

193.

194.

195.

196.

197.

198.

199.

200.

201.

202.

203.

204.

205.

206.

207.

208.

209.

210.

211.

212.

213.

214.

215.

216.

217.

218.

219.

220.

221.

222.

223.

224.

225.

226.

227.

228.

229.

230.

231.

232.

233.

234.

235.

236.

237.

238.

239.

240.

241.

242.

243.

244.

245.

246.

247.

248.

249.

250.

251.

252.

253.

254.

255.

256.

257.

258.

259.

260.

261.

262.

263.

264.

265.

266.

267.

268.

269.

270.

271.

272.

273.

274.

275.

276.

277.

278.

279.

280.

281.

282.

283.

284.

285.

286.

287.

288.

289.

290.

291.

292.

293.

294.

295.

296.

297.

298.

299.

300.

301.

302.

303.

304.

305.

306.

307.

308.

309.

310.

311.

312.

313.

314.

315.

316.

317.

318.

319.</

i^{th} = Parent

$2 \times i$ & $2 \times i + 1$ = child

Max elements $= n \Rightarrow i = \frac{n}{2}$

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

$$i = \frac{n-1}{2} \approx \frac{n}{2}$$

S.1 Sift (root's val, ^{max} child's val)

S.2. Convert To Heap (~~arr~~,
maxchild, n)

