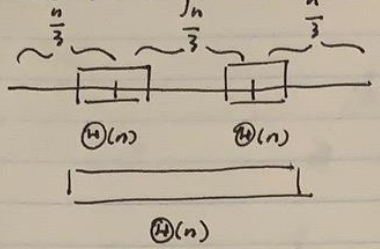
Homework 2.

Yimian Ren

1. Suppose we devide it into 3. we would have to find 2 cross-subarray plus one subarray that cross both boundary 1 and 2



So we need another function that counts the sum of second segment.

$T(n) = 3T(n/3) + \Theta(n)$. the complexity of running time is still $\Theta(n \log n)$

2. [10, 1, 6, 20, 16, 8, 33, 15]

[1 10 6 | 20 16 8 33 15]

[1 6 10 20 16 | 8 33 15]

[1 6 10 16 20 8 | 33 15]

[1 6 8 10 16 20 33 15]

[1 6 8 10 15 16 20 33]

insertion sort.

only one insert per iteration but multiple comparisons

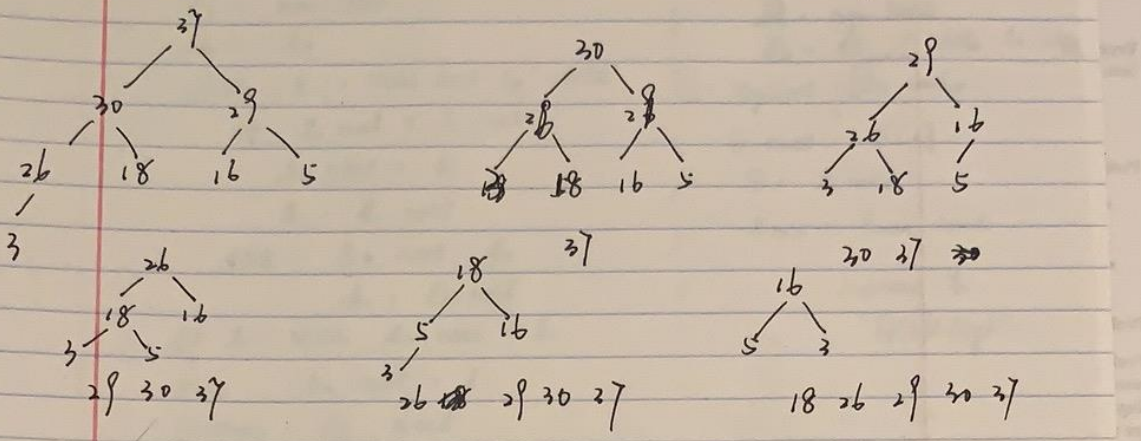
[10, 1, 6, 20, 16, 8, 33, 15] merge sort [1 6 8 10 15 16 20 33]

[10, 1, 6, 20] [16, 8, 33, 15] [1 6 10 20] [8 15 16 33]

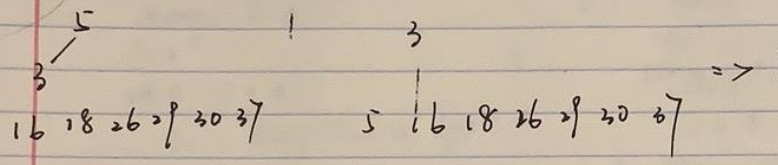
[10, 1] [6, 20] [16, 8] [33, 15] [1 6] [16, 20] [8, 16] [15 33]

[10] [1] [6] [20] [16] [8] [33] [15] [1] [6] [16] [20] [8] [16] [15] [33]

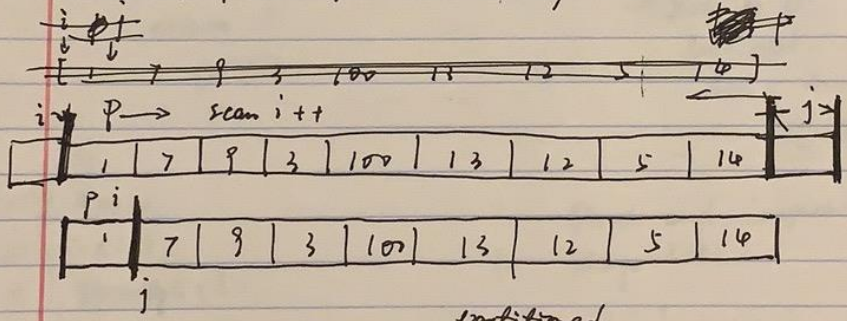
3. [30, 18, 26, 3, 37, 5, 16, 29] Heap sort



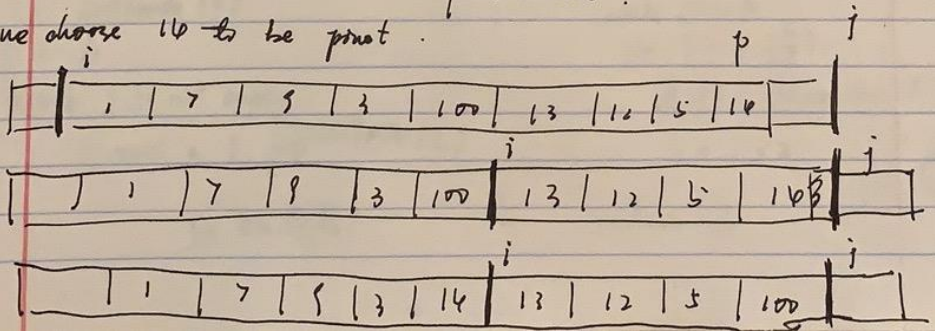
=> 3 5 16 18 26 29 30 37



6. Hoare's Partition: [1, 7, 9, 3, 100, 13, 12, 5, 14]



if we choose 16 to be pivot.



5. 5.1 MERGE-TWO(l_1, l_2):

l_3 = new Node.
l_4 = l_3.
While l_1 != NULL AND l_2 != NULL
  if l_1.val ≤ l_2.val
    l_3.next = l_1
    l_1 = l_1.next
  else: l_3.next = l_2
    l_2 = l_2.next
If l_1 = NULL l_3.next = l_2
else: l_3.next = l_1
Return l_4.next.

5.2. Merge-k(l):

j = 2
while (j ≤ k):
  l_1 = MergeTwo(l_1, l_j)
  j++
return l_1

$O(k^2 n)$

5.3. Merge-k(l):

for l_k in l:
while (l_k != NULL):
collect l_k.value in L
  l_k = l_k.next.
L = Merge-sort(L). */ Brute force
for node in L:
  l_{res}.next = node.
  l_{res} = l_{res}.next.

$O(kn \log kn)$

5.4.

priority Queue:

Merge-k(l):
Q = PriorityQ(l)
A = ListNode(0)
B = A
While Q is not empty:
  A.next = Q.EXTRACT-MAX(l)

$O(kn \log k)$

Merge-k(l):
A = new Node
B = A while l_k != NULL
S = MergeSort([l_k.value])
B.next = S[1]
B = B.next
l_{min} = l_{min}.next.

improve to $O(nk \log k)$.

Divide and Conquer:

Merge-k(l):
i = 1
while (i < k):
  for j = 1..$[\frac{k}{2}]$:
    l[j] = Merge-Two(l[j], l[j+$[\frac{k}{i}]$])
  i = i × 2

return l

$O(\sum_{i=1}^{\log k} 2 \cdot \frac{kn}{i}) = O(kn \log k)$