1. For the maximum subarray problem, if we use divide-conquer, but instead of dividing the array into two halves, we equally divide it into three segments, how should the algorithm be modified? What is the running time of the new algorithm?

   Solution: If we divide the original array A into 3 equal-sized sub-arrays S1, S2, and S3, we have 3 cases to consider in the combine phase of the divide-and-conquer algorithm:

   (a) The max subarray starts from S1, end in S2: linear scan in S1 from tail to head and scan in S2 from head to tail and summation = $\Theta\left(\frac{2n}{3}\right) + \Theta(1)$.

   (b) The max subarray starts from S2, end in S3: linear scan in S2 from tail to head and scan in S3 from head to tail and summation = $\Theta\left(\frac{2n}{3}\right) + \Theta(1)$.

   (c) The max subarray starts from S1, end in S3: linear scan in S1 from tail to head and scan in S3 from head to tail and compute the sum of S2 and summation = $\Theta(n) + \Theta(1)$.

   So, we can get the recurrence formula: $T(n) = 3T\left(\frac{n}{3}\right) + \Theta\left(\frac{2n}{3}\right) + \Theta(1) + \Theta\left(\frac{2n}{3}\right) + \Theta(1) + \Theta(n) + \Theta(1) \Rightarrow T(n) = 3T\left(\frac{n}{3}\right) + \Theta(n) = 3T\left(\frac{n}{3}\right)$. Where the result is $T(n) = \Theta(n \log n)$.

2. Demonstrate the operation of HOARE-PARTITION on the array *A* = <13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21>. Show the values of the array after each iteration of the while loop in the lines of 4 to 11 in the code of lecture notes.

   Solution:

   x = A[1] = 13.

   i = 0, j = 13

   The 1st iteration:

   j = 11 because A[11] = 6 < 13

   i = 1 because A[1] = 13 = 13

   After exchange, A = <6, 19, 9, 5, 12, 8, 7, 4, 11, 2, 13, 21>

   The 2nd iteration:

   j = 10 because A[10] = 2 < 13

   i = 2 because A[2] = 19 > 13

After exchange, A = <6, 2, 9, 5, 12, 8, 7, 4, 11, 19, 13, 21>

The 3rd iteration:

j = 9 because A[10] = 11 < 13

i = 10 because A[2] = 19 > 13

Because i > j, do not exchange. A = <6, 2, 9, 5, 12, 8, 7, 4, 11, 19, 13, 21>

Loop end.

3. For the following array:

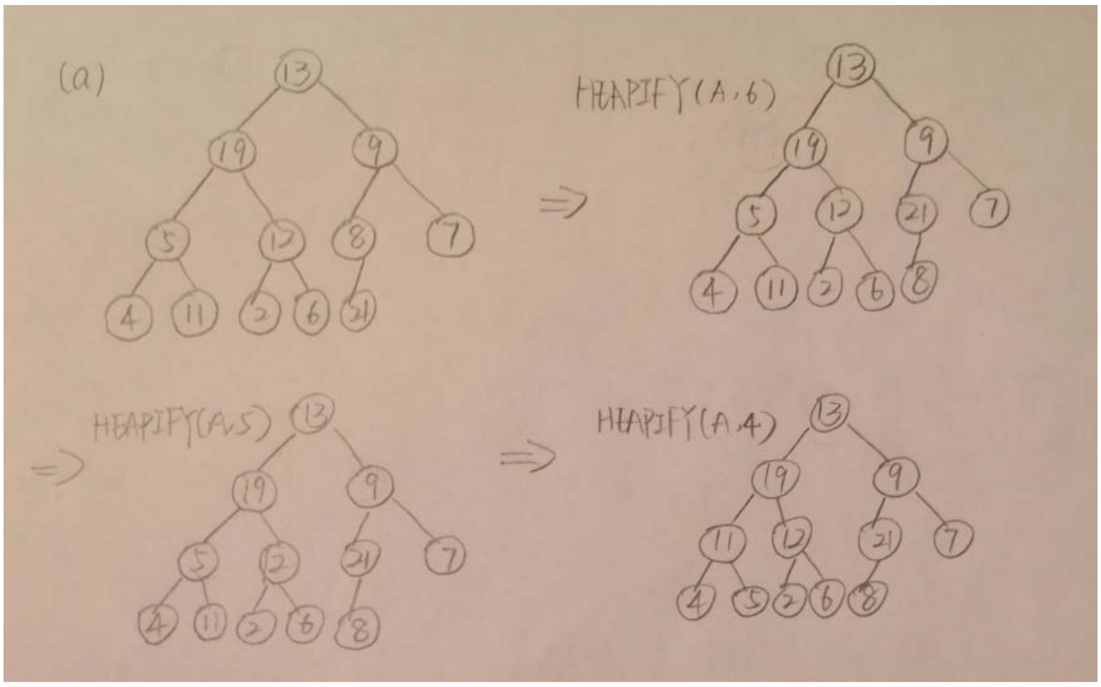
A = <13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21>

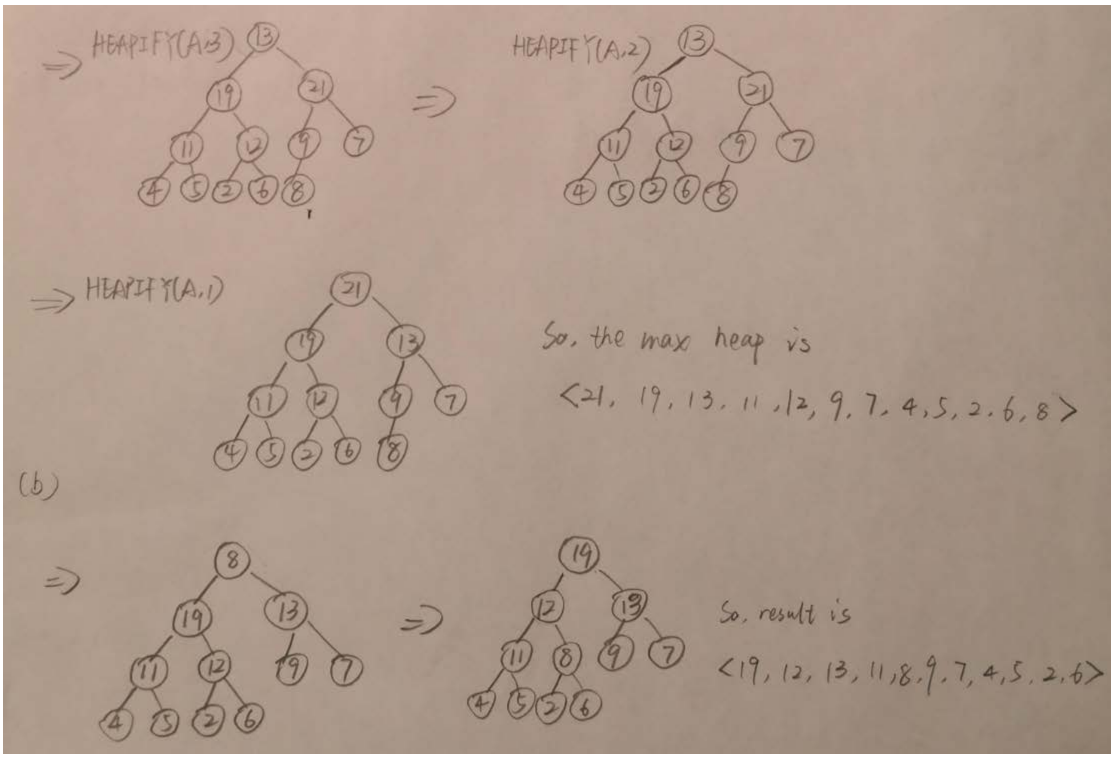
(a) Create a max heap using the algorithm BUILD-MAX-HEAP.

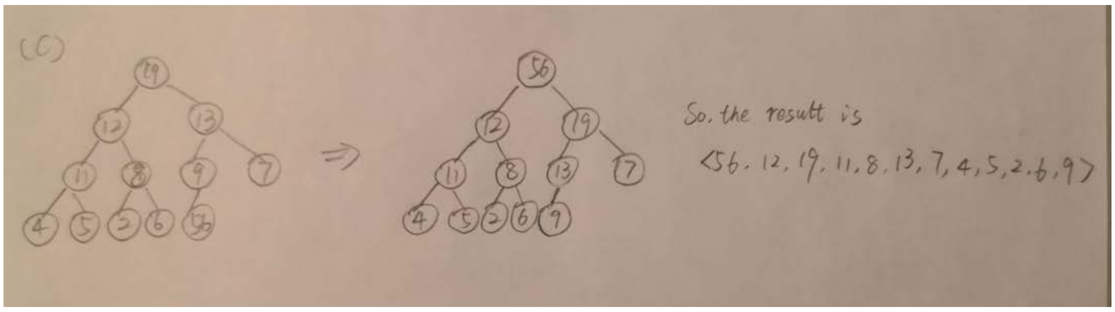
(b) Remove the largest item from the max heap you created in 3(a), using the HEAP-EXTRACT-MAX function. Show the array after you have removed the largest item.

(c) Using the algorithm HAX-HEAP-INSERT, insert 56 into the heap that resulted from question 3(b). Show the array after insertion.

Solution:



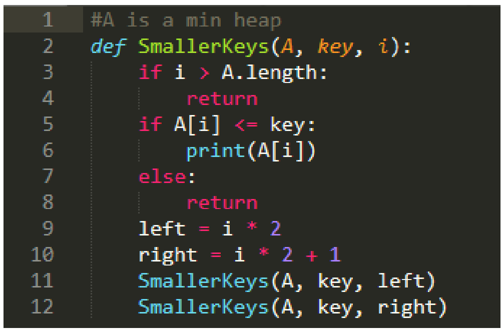




4. Let T be a min heap storing n keys. Given an efficient algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). Note that the keys do not need to be reported in sorted order. Ideally, your algorithm should run in $O(k)$ time, where k is the number of keys reported that is smaller than x.

Solution:

Because the output can be not in sorted order. We use recursive call to realize traversal, pseudocode is shown as below:



```
#A is a min heap
def SmallerKeys(A, key, i):
    if i > A.length:
        return
    if A[i] <= key:
        print(A[i])
    else:
        return
    left = i * 2
    right = i * 2 + 1
    SmallerKeys(A, key, left)
    SmallerKeys(A, key, right)
```

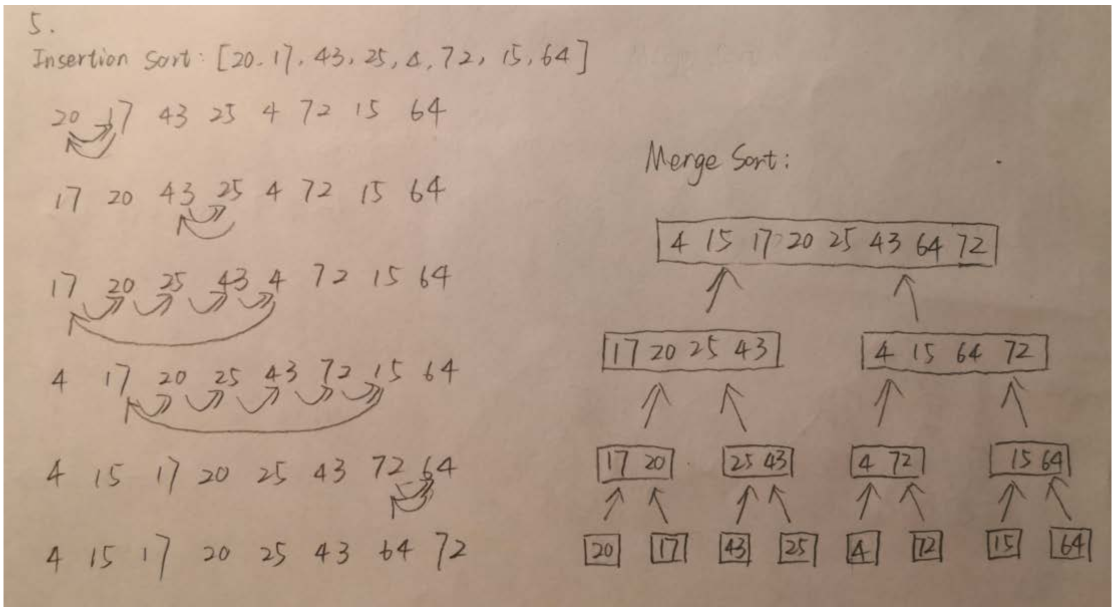
(We accept all codes make sense.)

Then we call SmallerKeys(T, key, 1), which will output all items smaller than key.

Running time: because it is a recursive call. Assume that there are k keys smaller than or equal to key. This function will call k time and return when we get to the next level of the last level. So, the total running time is $O(k)$.

5. Using Figure 2.4 on page 35 of CLRS as a model, illustrate the operation of insertion sort and merge sort on the array [20, 17, 43, 25, 4, 72, 15, 64].

   Solution:



6. Design an algorithm to merge k sorted arrays, and return it as a new array. The new array should be made by splicing together the nodes of the k arrays. Additionally, the total number of elements of all arrays is kn. (Notice that the number of elements of each array is not necessary the same). Ideally, your algorithm should run in $O(kn \log k)$ time, lower algorithm will lose some points. Please give the description of your algorithm and analyze the running time.

   For example:

   Input: A: <1, 5, 18>, B: <3, 6, 7, 11>, C: <2, 4, 9, 12, 13>

   Output: <1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 18>

   Solution:

   Basically, we can initialize a min-heap and insert 1st element in all the arrays into the heap. This will cost $O(k)$ time. And we repeat following 2 operations until all arrays has been scanned to the last item:

i. Get minimum element from heap (using EXTRAC-MIN) and store it in output array.

ii. Replace heap root with the next element from the array where the element is extracted. If the array does not have any more element (i.e. being scanned to the last item), replace root with infinite. After replacing, call HEAPIFY(A, 1).

Because totally we have kn keys, the HEAPIFY takes $O(\log k)$ time. So, the total running time is $O(k) + knO(\log k) = O(kn \log k)$.

(Description is enough, you do not need to provide any pseudocode.)