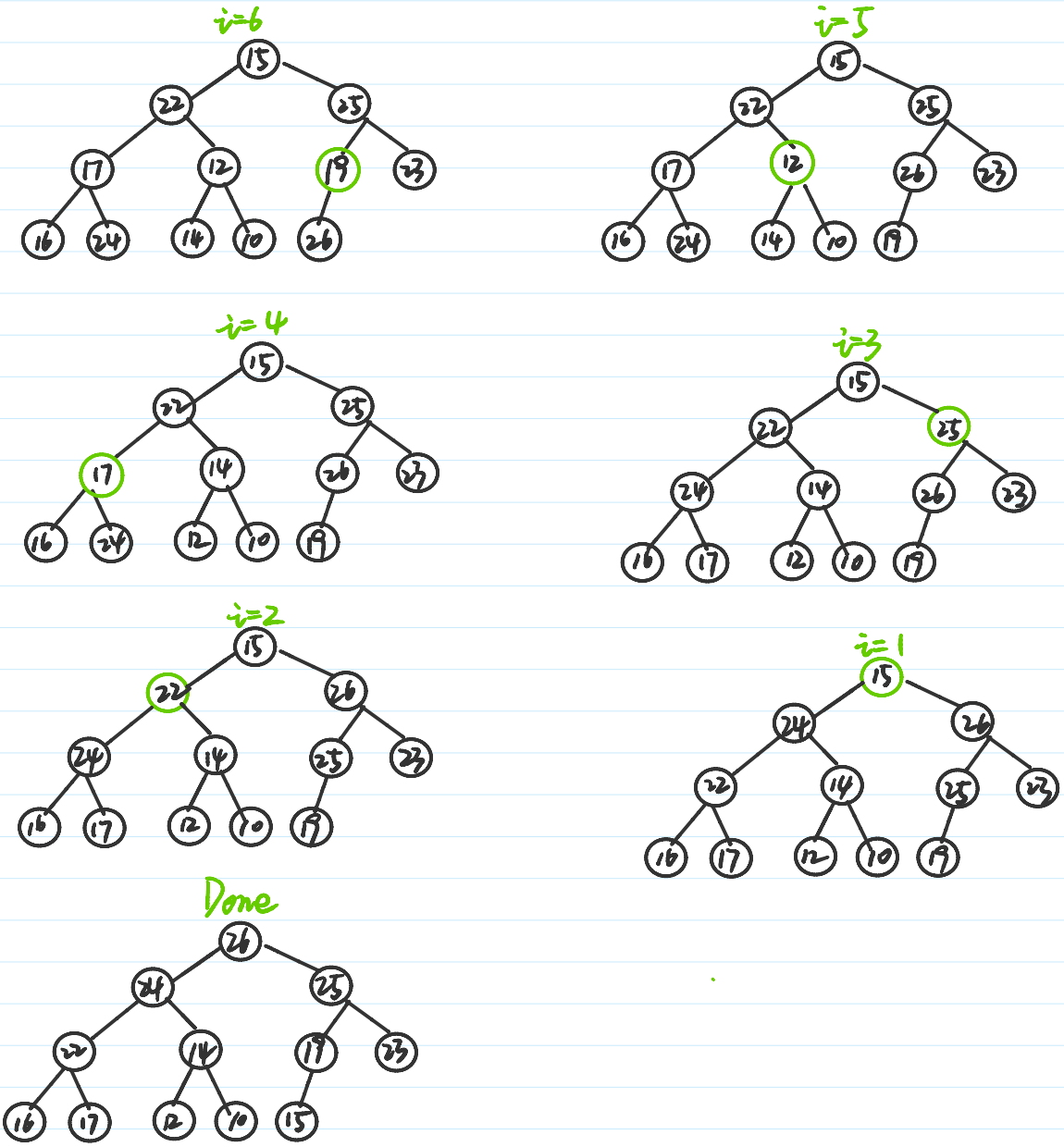
Assignment4

2021年10月10日 21:20

Q1

a). The following illustration of algorithm is drawn according to lecture slides



b) Algorithm:

step1: Put the array into heap according to their index

step2: for index = ⌊A.length/2⌋ to 1:

do min-heapify

Algorithm: Build-min-heap(A)

n = length[A]

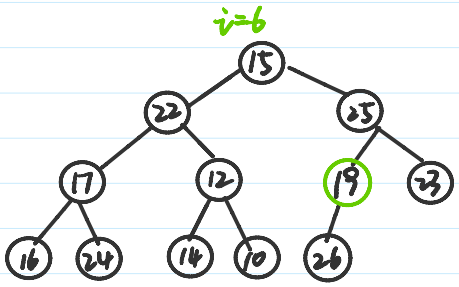
for i = ⌊n/2⌋ to 1:

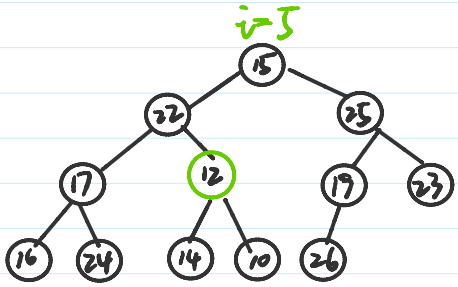
min-heapify(A, i)

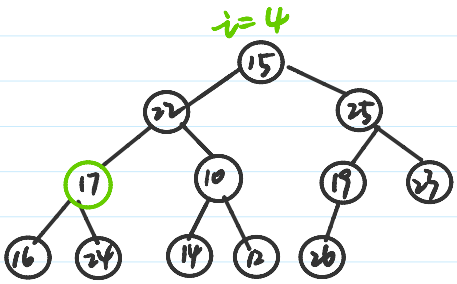
Algorithm: min-heapify (A, i)

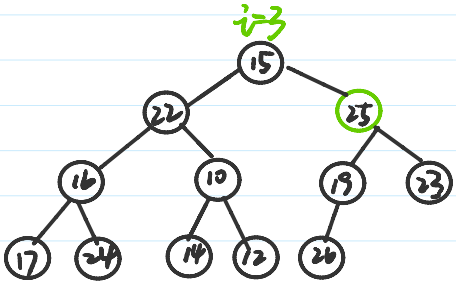
```
l=2i        ; r=2i+1
if( l <=heap_size(A) and A[l] < A[i]):
      smallest=l
else:
      smallest= i
if (r <= heap_size(A)  and  A[r] < A[smallest]):
      smallest = r
if (smallest != i)
      swap(A, i, smallest)
      min-heapify (A, smallest)
```

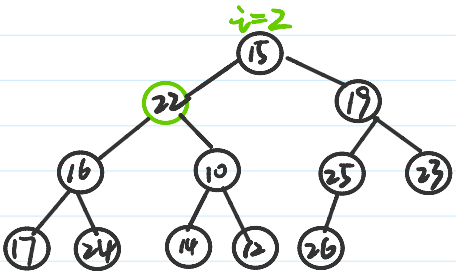
c)

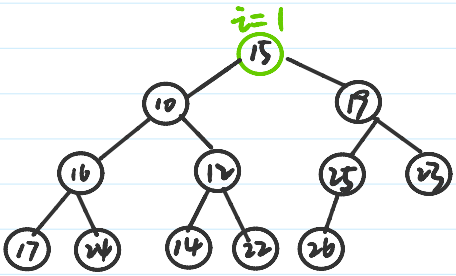


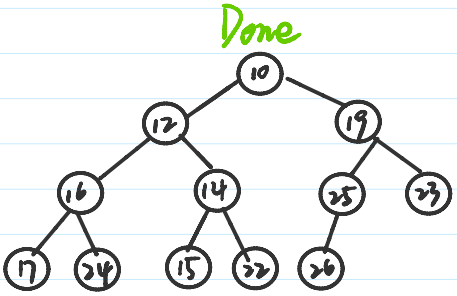




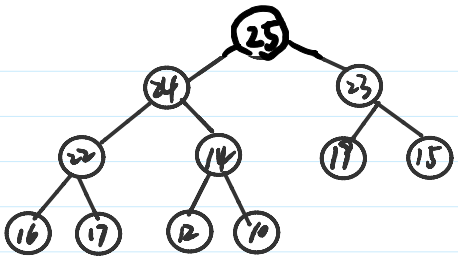




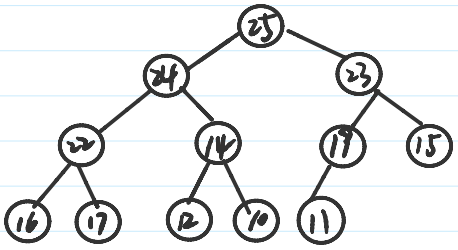




d)



e)



Q2. First algorithm:

Utilize merge method in "merge sort" to merge all the arrays.

Step1: put the remaining arrays into several groups. Each group contains two arrays

Step2: merge two arrays A,B in each group. To start with, use pointer i, j=0 point to the head of each array. if A[i]<A[j], put A[i] into merged array. otherwise, put B into it. Repeat until roll over array A and B. Get several merged arrays. Define these merged array as remaining arrays, repeat step 1.

$T(kn) = T(an) + T(kn-an) + f(n)$, $f(n)$ = merge complexity = kn

Since there are totally K arrays, algorithm above will repeat $\log k$ times.

Complexity $= O(kn \log k)$

Second Algorithm:

Step1: Remove the 1st element of each array. Put them into a min-heap, build the min-heap.

Step2: remove the root of the main heap and put it into the result array. Extract an element from the array where previous removed element belongs to. If the array is empty, do nothing. Insert the new element into the root and heapify it.

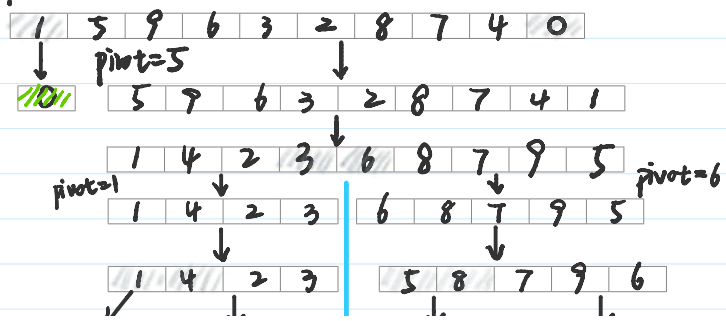
Step3: Repeat step2 until all the original arrays are empty and all the element in the result array.

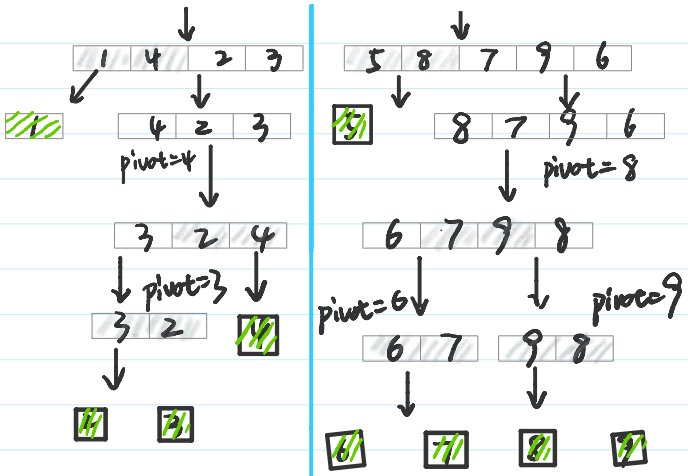
complexity $= O(kn \log k)$. Since it takes $O(\lg n)$ to heapify in a heap. and we have kn elements.

Q3.

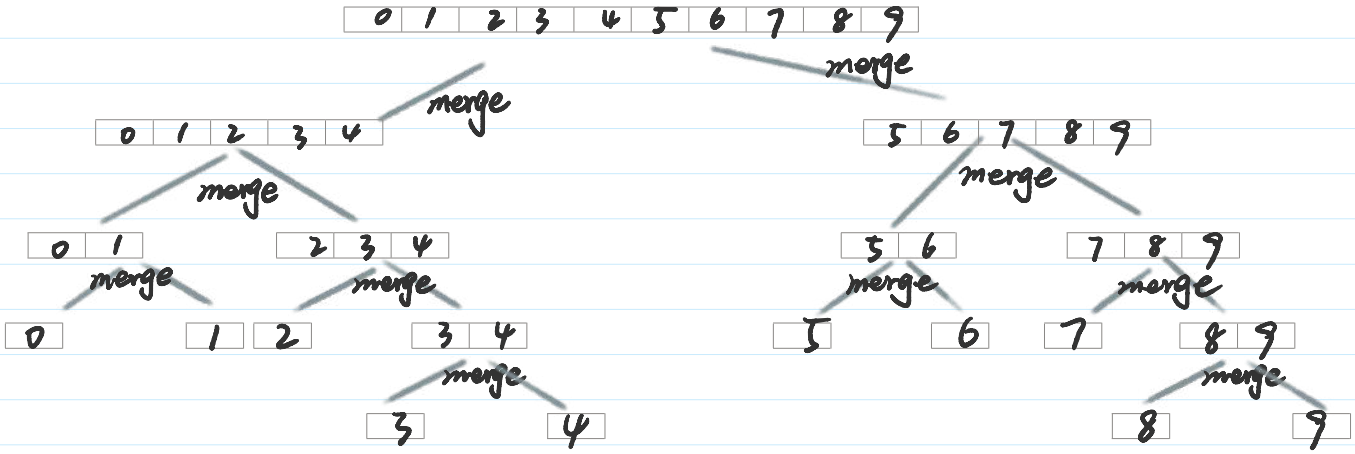
pivot=1

a)





b)



c) Quick Sort compare to merge Sort

advantages:

① Quick sort is in place. But merge sort need aditional memory.

② Quick sort is faster than merge sort in case of smaller array size.

Disadvantage:

① Division may not be equal. ② Quick sort has worst case complexity = $O(n^2)$. Merge sort has worst case complexity = $O(n\log n)$

Q4.

```
Algorithm: GetMedian(array, N):
Median = array[1]
smaller <- MaxHeap
larger <- MinHeap
for (i = 2 to N):

    if (smaller.size() < larger.size())):
        if (array[i] < median):
            smaller.add(array[i])
        else:
            larger.add(smaller.removeRoot())
            smaller.add(array[i])
        median = (smaller.getRoot() + larger.getRoot()) / 2

    if (smaller.size() == larger.size()):
        if (array[i] < median):
            smaller.add(array[i])
            median = smaller.getRoot()
        else:
            larger.add(array[i])
            median = larger.getRoot()

    if (smaller.size() > larger.size()):
        if (array[i] < median):
            larger.add(smaller.removeRoot())
            smaller.add(array[i])
        else:
            larger.add(array[i])
        median = (smaller.getRoot() + larger.getRoot()) / 2
return median
```