EL9343 Homework 4

(Due Oct 5th, 2021)

No late assignments accepted

All problem/exercise numbers are for the third edition of CLRS text book

---

1. For the following array:

$$A = \langle 15, 22, 25, 17, 12, 19, 23, 16, 24, 14, 10, 26 \rangle$$

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

```
[26, 24, 25, 22, 14, 19, 23, 16, 17, 12, 10, 15]
```

(b) Design an algorithm to create a min heap. (Pseudocode is required)

```
part1: 50%
################################################
BUILD-MIN-HEAP(A)
n = length[A]
for i ← ⌊n/2⌋ downto 1
 do MIN-HEAPIFY(A, i, n)

part2: 50%
################################################
Min-Heapify(A, i)
{
 l = Left(i); r = Right(i);
 if (l <= heap_size(A) && A[l] < A[i])
 smallest = l;
 else
 smallest = i;
 if (r <= heap_size(A) && A[r] < A[largest])
 smallest = r;
 if (smallest != i)
 Swap(A, i, smallest);
 Min_Heapify(A, smallest);
}
```

(c) Create a min heap using the algorithm you designed in 1(b)

```
[10, 12, 19, 16, 14, 25, 23, 17, 24, 15, 22, 26]
```

(d) Remove the largest item from the max heap you created in 1(a), using the HEAP-EXTRACT-MAX function. Show the array after you have removed the largest item.

```
[25, 24, 23, 22, 14, 19, 15, 16, 17, 12, 10]
```

(e) Using the algorithm MAX-HEAP-INSERT, insert 11 into the heap that resulted from question 1(d). Show the array after insertion.

```
[25, 24, 23, 22, 14, 19, 15, 16, 17, 12, 10, 11]
```

2. Design two different algorithms 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). One of your algorithms should run in $O(kn \log k)$ time. Please give the procedure of your algorithm and analyze the running time. (Description is enough, you do not need to provide any pseudocode)

For example:

Input: A: <1, 4, 7, 10>, B: <2, 5, 8, 11>, C: <3, 6, 9>

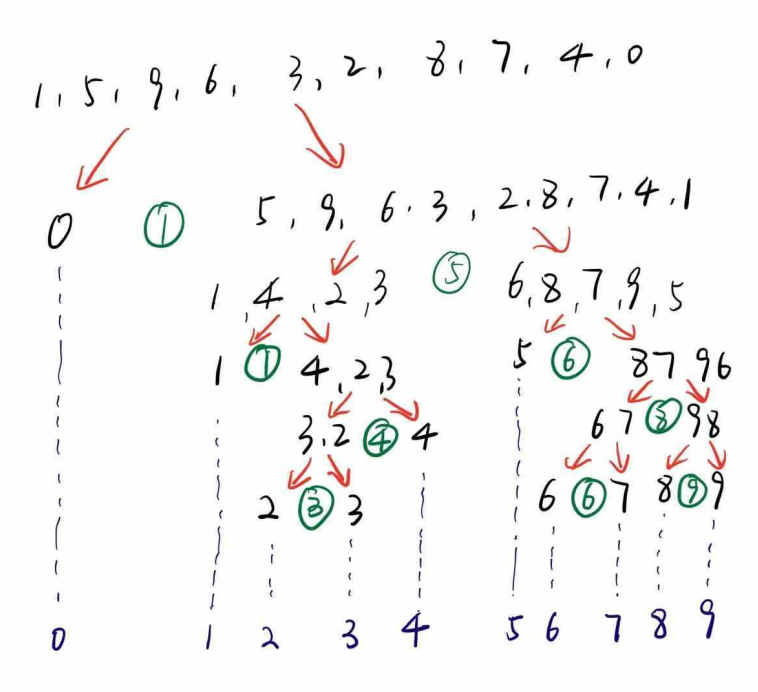
Output: <1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11>

```
knlogk:
#########################################################################################################################
Basically, we can initialize a min-heap and insert 1st element in all the arrays into the heap.
This will cost Ok 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 Ok+knOlog k =O(knlog k ).
#########################################################################################################################
The other algorithms can be implemented in these ways:
1. merge one by one
2. merge by pairs
3. put all together and sort
#########################################################################################################################
To get full mark, just implement the algorithm with the running time of O(knlogk) (80%)
and one any other algorithm.(20%)

```

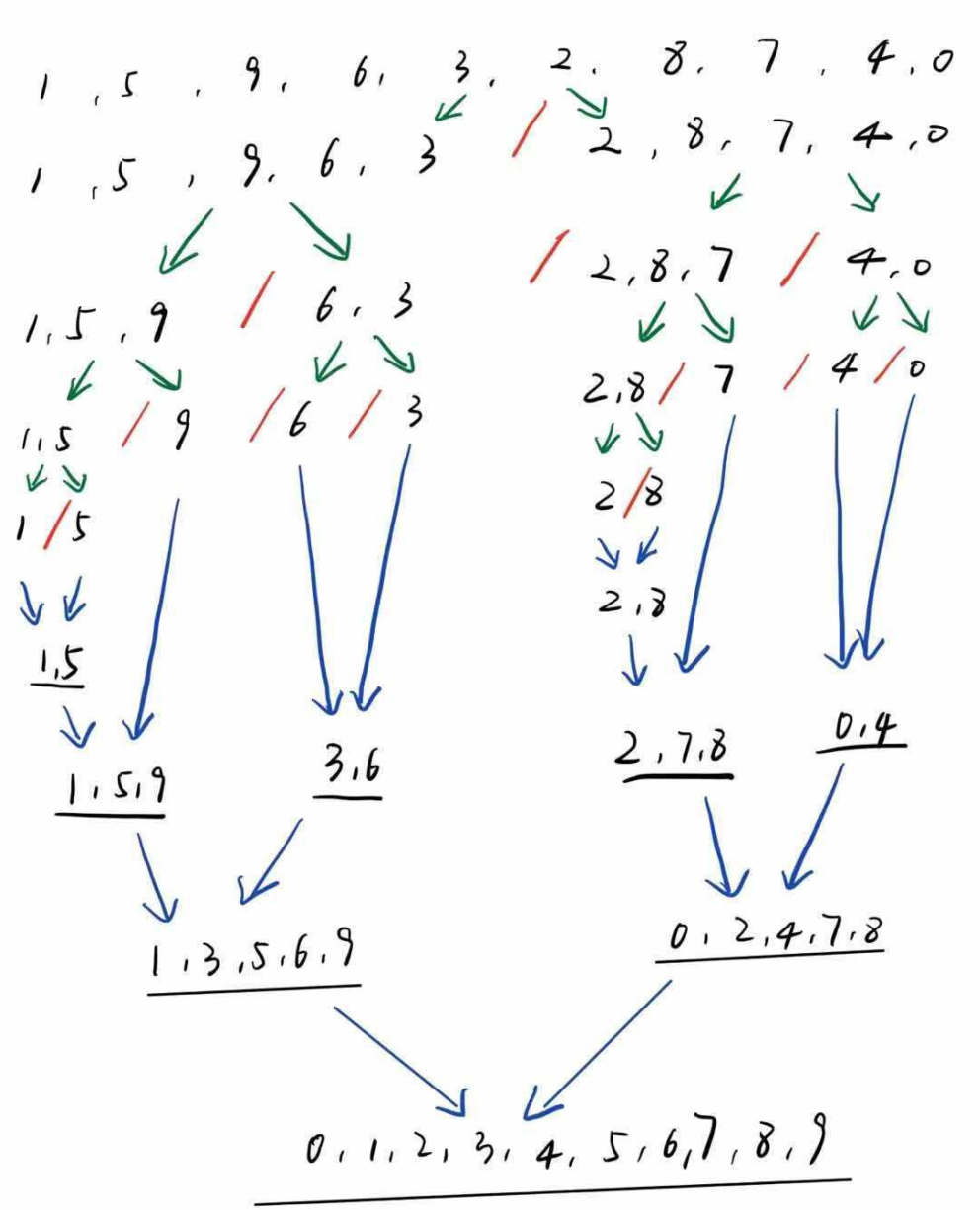
3. For the following array:

$$A = <1, 5, 9, 6, 3, 2, 8, 7, 4, 0>$$

(a) Illustrate the operation of quick sort on array A



(b) Illustrate the operation of merge sort on array A



(c) Explain the advantage and disadvantage of sorting an array by quick sort compared to using merge sort.

```
QUICK SORT:
Worst case:O(n2)
Average case: O(nlogn)
Storage: in place
Stability: Not Stable
MERGE SORT:
worst case: O(nlogn)
Average case: O(nlogn)
Storage: O(n)
Stability: Stable
```

4. For an disordered array with *n* elements, design an algorithm for finding the median of this

array. Your algorithm should traverse the array only once.

Other reasonable solutions can also get full marks

Description is enough.

```
ALG: FIND_MEDIAN(A)
HEAPSIZE = SIZE(A) / 2 + 1
HEAP = BUILD_MIN_HEAP(A,0,HEAPSIZE) // BUILD
//BUILD_MIN_HEAP stops loading data to the tree when A[HEAPSIZE] is read
//Then do similary thing as BUILD_MAX_HEAP in slides
for i <- HEAPSIZE + 1 TO SIZE(A)
    do if A[i] > HEAP[1]
        do HEAP[1] = A[i] //SWAP
            MIN_HEAPIFY(HEAP, 1) //HEAPIFY
if SIZE(A) % 2 == 1
    return HEAP[1]
else
    return (HEAP_EXTRACT_MIN + HEAP[1]) / 2
```

```
####################################
get full marked if the following parts are included:
1. BUILD HEAP, by half of the array and traverse the other half of the array(25%)
2. SWAP, when A[i] is smaller than the root value of the min heap(25%)
3. HEAPIFY, after SWAP(25%)
4. RETURN: the root value of the heap(25%)
(A similary algorithm using max heap is also accpetable)
####################################
Another way to solve this problem is build a min_heap and also a max_heap.
Get full marked if the following parts are included:
1. BUILD TWO HEAPS, one is a min_heap, the other is a max_heap, traverse the array(25%)
3. add A[i] to max_heap if A[i] <= min_heap's root value (12.5%)
4. add A[i] to min_heap if A[i] > min_heap's root value (12.5%)
5. Extract the root value of the max_heap and add it to the min_heap if len(max_heap) > len(min_heap) (12.5%)
6. Extract the root value of the min_heap and add it to the max_heap if len(min_heap) > len(max_heap) + 1 (12.5%)
7. return the root value of min_heap if len(A) is odd (12.5%)
8. return (max_heap[0], min_heap[0]) / 2 if len(A) is even(12.5%)
####################################
Algorithms implemented by quick select or sort can get 80%

```