CSCI-GA.1170-001/002 Fundamental Algorithms October 6, 2015

Solutions to Problem 4 of Homework 4 (12 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, October 6*

We wish to implement a data structure D that maintains the k smallest elements of an array A. The data structure should allow the following procedures:

- $D \leftarrow$ Initialize(A, n, k) that initializes D for a given array A of n elements.
- Traverse(D), that returns the k smallest elements of A *in sorted order*.
- Insert(D, x), that updates D when an element x is inserted in the array A.

We can implement D using one of the following data structures: (i) an unsorted array of size k; (ii) a sorted array of size k; (iii) a max-heap of size k.

(a) (4 points) For each of the choices (i)-(iii), show that the Initialize procedure can be performed in time $O(n + k \log n)$.

Solution:

Initialize

Approach

The idea is to find the k^{th} smallest element in the array A and then traverse the array to find all the elements less than the k^{th} smallest element.

Following the idea of Quicksort, the following algorithm is called Quickselect which returns the k^{th} smallest element in $O(n)$ time. We choose a random element as a pivot and now we know in which partition the k^{th} smallest element lies in. So we recursively find the desired element in that partition

```
Algorithm: QUICKSELECT(A, low, high, k)
if low is high then
    Return A[low]
end
q ← PARTITION(A, low, high)
if q is k then
    Return A[q]
else if q > k then
    Return QUICKSELECT(A, low, q − 1, k)
else
    Return QUICKSELECT(A, q + 1, high, k)
```

Algorithm 5: Algorithm that returns k^{th} smallest element of A in $O(n)$ time

Time Complexity of Quickselect

$$
\begin{aligned}
T(n) &= T(q) + O(n) \;\; (or) \;\; T(n-q) + O(n) \;\; where \;\; q \leftarrow 1 \;\; to \;\; n-1 \\
&= T(k) + O(n) \;\; where \;\; k \leftarrow 1 \;\; to \;\; n-1
\end{aligned}
$$

$$
\therefore T(n) = \begin{cases} T(1) + O(n) \\ T(2) + O(n) \\ \vdots \\ T(n-1) + O(n) \end{cases}
$$

Expected Running Time, $T(n)$ will be

$$
\begin{aligned}
T(n) &= \frac{1}{n}(T(0) + T(1) + \ldots + T(n-1) + n^2) \\
nT(n) &= T(0) + T(1) + \ldots + T(n-1) + n^2 \\
(n-1)T(n-1) &= T(0) + T(1) + \ldots + T(n-2) + (n-1)^2 \\
nT(n) - (n-1)T(n-1) &= T(n-1) + 2n - 1 \\
nT(n) &= nT(n-1) + 2n - 1 \\
T(n) &= T(n-1) + 2 - 1/n \\
&= 2n - (1/n + 1/(n-1) + \ldots + 1) \\
&\leq 2n \\
&= O(n)
\end{aligned}
$$

Therefore the average running time of Quickselect is only $O(n)$. However in the worst case, it could go to $O(n^2)$

i Using Unsorted array

```
Algorithm: INITIALIZE(A)
D ← NEWARRAY(K)
c ← 0
key ← QUICKSELECT(A, 0, n − 1, k)
for i ← 0 to n − 1 do
    if A[i] < key then
        D[c] ← A[i]
        c ← c + 1
    end
end
Return D
```

Algorithm 6: Initialize D with an Unsorted array in $O(n)$ time

It is clear that from the above pseudocode the running time of the algorithm is $O(n) = O(n + k \log n)$

ii Using Sorted Array

```
Algorithm: Initialize(A)
D ← newArray(k)
c ← 0
key ← Quickselect(A, 0, n − 1, k)
for i ← 0 to n − 1 do
    if A[i] < key then
        D[c] ← A[i]
        c ← c + 1
    end
end
Merge-Sort(D)
Return D
```

Algorithm 7: Initialize D with a Sorted array in $O(n + k \log k)$ time

It is clear that from the above pseudocode the running time of the algorithm is $O(n + k \log k) = O(n + k \log n)$

iii Using Max-heap

```
Algorithm: Initialize(A)
D ← newArray(k)
c ← 0
key ← Quickselect(A, 0, n − 1, k)
for i ← 0 to n − 1 do
    if A[i] < key then
        D[c] ← A[i]
        c ← c + 1
    end
end
Build-MaxHeap(D)
Return D
```

Algorithm 8: Initialize D with a Max-heap in $O(n + k)$ time

It is clear that from the above pseudocode the running time of the algorithm is $O(n + k) = O(n + k \log n)$

□

(b) (3 points) For each of the choices (i)-(iii), compute the best running time for the TRAVERSE procedure you can think of. (In particular, tell your procedure.)

Solution:

TRAVERSE

i Apply merge-sort on D and return D. Time taken is $O(k \log k)$

ii The array is already sorted, so just return D. Time taken is $O(1)$

iii Keep performing Extract-Max from D and return in the reverse order. Time taken is $O(k \log k)$

□

(c) (5 points) For each of the choices (i)-(iii), compute the best running time for the INSERT procedure you can think of. (In particular, tell your procedure.)

Solution:

INSERT

i Find the maximum element in D. If the inserted element x is smaller than the maximum element, swap each other. Time taken is $O(k)$

ii Perform binary search in D. Insert x at the appropriate position and shift D to the right by one index from x and swap the last element of D with the position where x was inserted in A. Time taken is $O(k + \log k) = O(k)$

iii Find-Max in D. If the x is smaller than the max-element, Extract-Max and then Insert x into the heap D and insert the extracted maximum from the heap at the position where x was inserted in A. Time taken is $O(\log k)$

□