ASSIGNMENT – 2

Introduction to Algorithms (CS 430)

1. To determine the worst-case time complexity of an alternative version of SELECT algorithm where the elements are grouped in groups of size 2m + 1 (with m >= 2 being an integer constant);

Analysis:

1. Grouping and finding medians:
   - Divide n elements into $\frac{n}{2m+1}$ groups of 2m + 1 elements each, with at most one group containing less than 2m + 1 elements.
   - We will sort each group and will find median of each group. *Time complexity: O(n)*
2. Using recursive function for medians:
   - Let M be the array of medians of all these groups. The size of M is $\frac{n}{2m+1}$.
   - Recursively find the median of M. Time complexity: T ($\frac{n}{2m+1}$)

Recurrence Relation:

- The recurrence relation for time complexity T(n) is: T(n) <= T ($\frac{n}{2m+1}$) + T ($\frac{2m * n}{2m+1}$) + O(n)

Solving the Recurrence

- T(n) <= T ($\frac{n}{2m+1}$) + T ($\frac{2m * n}{2m+1}$) + O(n)
- Where; a = 2 , b = 2m+1, f(n) = O(n)
- c = $\log_{(2m+1)} 2$
- Since, f(n) = O(n) and c = $\log_{(2m+1)} 2$ //the work done at each level of recursion tree is O(n), height of tree is O(log n)

Conclusion:

The worst-case time complexity of SELECT algorithm, grouping elements in groups of size 2m+1 where m >= 2 is: T(n) = O(n).

2. O(n lg k)-time algorithm to merge k sorted arrays into one sorted array, where n is the total number of elements in all the input arrays.

MERGE K SORTED ARRAYS(A1, A2, …, Ak)

1. Initialize a min-heap H with the first elements of each arrays A1, A2, …, Ak
2. Initialize an empty output array B
3. while H is not empty
4.   x = HEAP-EXTRACT-MIN(H)
5.   Append x to the end of B
6.   i = index of the array that x was extracted from
7.   if there are more elements in array Ai
8.   Insert the next element from Ai into H
9. Return B

The key steps are:

- Initializing a min-heap H with the first elements of each of the k sorted input arrays.
  Time taken: O(k)
- Extracting the minimum elements repeatedly from the min-heap, appending it to the output array B, and inserting the next element from that array into min-heap.
  Time taken for; Extracting minimum: O(log k), Inserting an element: O(log k).
- This is done n time, once for each element in the input arrays.
- ***The total time complexity is O(k + n log k) = O(n log k).***

3. Pseudo-code for MAX-HEAP-DELETE (A, i) that deletes the element in A[i] from a binary max-heap A.

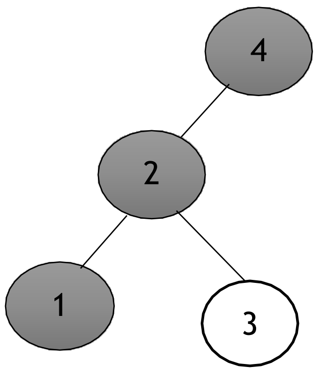
1. if i > heap_size(A) or I < 1
2.   error "index out of bounds"
3. A[i] = A[heap_size(A)]
4. Decrease heap_size(A) by 1
5. if i > 1 and A[PARENT(i)] < A[i]
6.   HEAP-INCREASE-KEY(A, i, A[i])
7. else
8.   MAX-HEAPIFY(A, i)

Time Complexity:

- HEAP-INCREASE-KEY: O(log n)
- MAX-HEAPIFY: O(log n)

❖ ***Time complexity of MAX-HEAP-DELETE is O(log n).***

4. Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k **in a binary search tree ends up in a leaf. Consider three sets:** A, the **keys to the left of the search path;** B, the **keys on the search path; and** C**, the keys to the right of the search path. Professor Bunyan claims that any three keys** $a \in A$**,** $b \in B$**, and** $c \in C$ **must satisfy** $a \leq b \leq c$**. Give the smallest possible (with fewest nodes) counterexample to the professor's conjecture.**



- We are searching for key k = 1
- The path traversed from root to leaf is the shaded nodes.
- ‘A’ indicates the keys on the left of the search path. Set A = {}; is null set/empty set.
- ‘B’ indicates the keys on the search path. ‘B’ = {4, 2, 1}
- ‘C’ indicates the keys on the right of the search path. ‘C’ = {3}
- **Since, 3 < 4, Prof. Bunyan’s claim is wrong. As, it does not satisfy the given condition.**