

3340A2

Hamza Bana, 250857725

February 27, 2020

1 Q1

Algorithm 1: Count (array, k, a, b)

Result: Output the number of integers n that fall within a range [a..b]
n = array.length;
arr = [0...k];
i = 0 **for** $j \leftarrow 0$ **to** n **do**
| arr[array[j]] ++;
end
for $j \leftarrow 1$ **to** n **do**
| arr[j] += arr[j-1];
end
//Above algorithm is from counting sort
return arr[b] - arr[a-1] //O(1) operation

2 Q2

See pdf

3 Q3

See pdf

4 Q4

No, the tree will not look the same. Inserting and deletion involves possible rotating, rebalancing and recoloring the tree nodes. See pdf for example

5 Q5

Prove that AVL tree with n nodes has height $\lg(n)$

Height of left and right subtree at every node, x, should only differ by 1 for the tree to be an AVL tree.

Let h_l be the height of the left subtree and h_r be the height of the right subtree. $|h_l - h_r| \leq 1$

Let n_h defined the minimum number of nodes in an AVL tree with height h

Base case: $n_0 = 1$ $n_1 = 1$ $n_2 = 2$

For a tree with height h , left side must be $h-1$ and right side must be $h-2$.

So the min number of nodes at height h can be expressed as the min number of nodes with height $h-1$ and height $h-2$, which are respectively the left and right subtree: $n_h = (n_{h-1} - 1) + (n_{h-2} - 2) + 1$

$h-1$ are all the nodes on the left of node x and $h-2$ are all the nodes on the right of node x. The +1 in the above formula indicates the counting of node x, or the root node.

Represents Fib sequence of $f(h) = f(h-1) + f(h-2)$

Solve the recurrence.

$n_h = 1 + n_{h-1} + n_{h-2}$

$n_h > 1 + 2n_{h-2}$

$n_h > 2n_{h-2}$

$n_h \in \theta(2^{h/2})$

$h < 2\log_2 n$

Hence at height h , $h < 2\log_2 n$ and at most could have height $\log n$ where there are n nodes.

6 Q6

Algorithm 2: ksorted(A, k)

Result: output a sorted sequence of smallest k elements

heap = heapify(A, A.length, 1);

//O(n) operation result = [k];

j = 0;

for $i \leftarrow 0$ **to** k //O(k) for loop **do**

 result[i] = heap.ExtractMinimum() //O(logn) operation;

 i ++;

end

//O(klogn)

return result;

The above algorithm is completed in $O(k\log n + n)$ time.

7 Q7

Prove that every node has at most rank $\log n$. Assume above statement is true.

If we have 1 node: $n = 1$. Rank = $\log n = 0$. Rank of node 1 = 0

Proof by induction: Assume $n+1$ total nodes.

Assume n nodes are in a set with the highest rank being $\log n$ of the parent root. Now call union operation with $(n+1)$ th node. Since the ranks are unequal of the parent node of both disjoint sets on which the union is being called, the rank of $\log n$ is preserved when the $(n+1)$ th node is called with the union operation.

Now assume there are two disjoint sets with a and b total nodes. The rank of the parent with a nodes is $\log a$ and rank with b nodes is $\log b$. a and $b \neq n$.

When calling union on $a + b$, assume that $a \neq b$ and $\log a \neq \log b$. $a + b = n+1$ and the rank will get incremented by one this time, hence the new rank will be $\log(n+1)$, where $n+1$ is the total nodes in the disjoint set.

We have proved by induction that every node has rank at most $\log n$ where n is the total number of nodes.

8 Q8

Since $\log n$ bits are needed to address n bytes. The highest rank possible is $\log n$ bytes where n is the total number of nodes.

We need to sub $\log n$ in to the first equation of addressing n bytes and we get: $\log(\log n)$

1 byte = 8 bits. Since $\log(\log n)$ bits are needed. We can use prove by example that we can use 1 byte to store rank. n is number of nodes. If $n = 10$, $\log(\log n) = 1.7$. If $n = 1,000$, $\log(\log n) = 3.3$. If $n = 100,000$, $\log(\log n) = 4$. If $n = 1,000,000,000$, $\log(\log n) = 4.9$. Hence we never use the entire 8 bits, hence 1 byte is enough to store the rank in question 10.

9 Q9

Encoding can be represented using a tree. A complete binary tree with n leaves has $2n-1$ total nodes. n characters can be address with $\log n$ bits. Since we need a tree of size $2n-1$ bits to represent the message with n characters. Each character required $\log n$ bits to be represented and there are n characters total, equaling $n(\log n)$ total memory to represent the entire prefix code.

Total memory required = memory for the tree + memory for the message

Memory for the tree = $2n-1$

Memory for the message = $n(\log n)$

Total memory required = $2n-1 + n(\log n)$

10 Q10

Correctness and complexity of question 10 code:

uandf: $O(n)$ - creating an array of size n

make_set: $O(1)$ - constant time. Allocation of data and parent pointers

union_sets:

2 find_set operations: $O(1)$

Checking if the ranks are same or not of two different set representatives = $O(1)$

Changing parent pointers = $O(1)$

find_set: $O(1)$ - check nullity of index in an array and return parent pointer

final_sets: Iterate from 1 to size of array (n). $O(n)$ operation.

Comparisons of the if-loop conditions are $O(1)$ time. Checking nullity, checking parent pointers, checking if data is equal to a certain number.

Total time complexity of final_sets() = $O(n)$