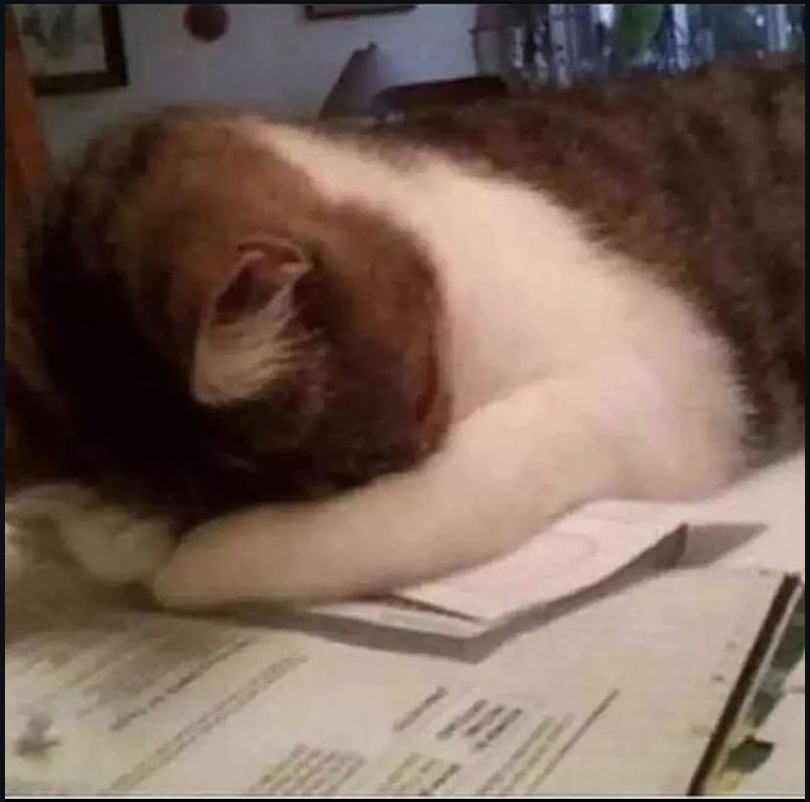
CS2040S Tutorial 6

Group T40

Week 8

Picture of the Day



:(

Problem 1: Priority Queue

Problem 1: Priority Queue

Given a data set. You'd like to know the top k largest elements. A possible solution is to store all n elements, sort it in $O(n \log n)$, then report the right-most k elements.

Give an algorithm to:

1. Find the top k largest elements better than $O(n \log n)$
2. Find the top k largest elements as the elements are streaming in, and is faster than $O(n \log n)$.

Solution

1. Use quickselect top k. Runs in average $O(n)$ time
2. Use min-heap to only keep top k. If heap is full and new number is larger than the top of heap, pop and push the new one. Runs in $O(n \lg k)$ in total

Problem 2: Union-Find Review

Problem 2a

What is the *worst case* running time of `find` operation in Union-Find with path compression, assuming without Weighted Union?

Problem 2b

```
def Find(i, j):
    return id[i] == id[j]

def Union(i, j):
    if size[i] < size[j]:
         Union(j, i)
    else:
        k1 = id[i]
        k2 = id[j]

        for every item m in list[k2]:
            id[m] = k1

        # append list[k2] on the end of list[k1] and set list[k2] to null
        size[k1] = size[k1] + size[k2]
        size[k2] = 0
```

Assumption: appending linked list is $O(1)$.

Notes

Operations\Data Structure	AVL	Binary Heap	Fibonacci Heap
insert	$O(\log n)$	$O(\log n)$	$O(1)$
find-min	$O(\log n)$ or $O(1)$ *	$O(1)$	$O(1)$
delete-min	$O(\log n)$	$O(\log n)$	$O(\log n)$
decrease-key	$O(\log n)$	N/A	$O(1)$
merge	$O(n \log n)$**	$O(n)$	$O(1)$

Notes

*: Note that for `find-min`, AVL tree can run in $O(1)$ if we also store the `successors` in a hash table. When we delete, update the min to the successor. When we insert, check whether it's smaller or not.
**: Not sure whether there's a more optimal way of merging two AVL trees besides inserting one by one.