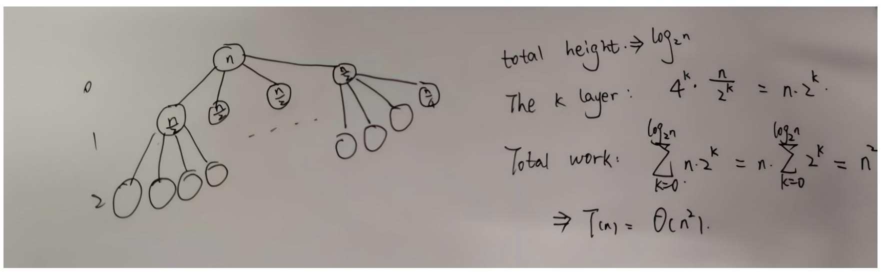
Name: LessTanker

Collaborators: Name1, Name2

Problem 2-1.

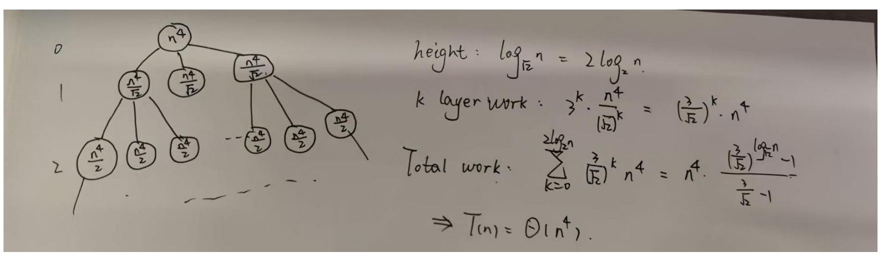
(a)

$$T(n) = \Theta(n^2)$$



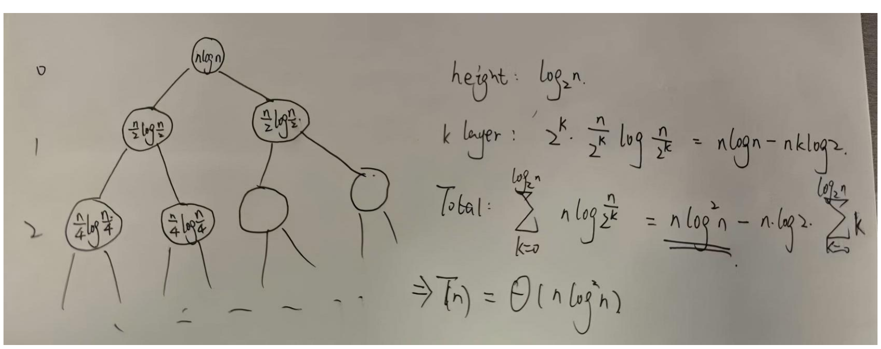
(b)

$$T(n) = \Theta(n^4)$$



(c)

$$T(n) = \Theta(n \log^2 n)$$



(d)

$$T(n) = \Theta(n^2)$$

Problem 2-2.

(a) Selection Sort

For each single item, to get it in the proper place, select sort needs a swap operation,which is 2 time set_at operation.

But for insert sort, each single item needs n times set_at operation, which is worse. Also same for merge sort.

(b) Merge Sort

For select sort and insert sort, both need $O(n^2)$ times to compare, take $O(n^2 \log n)$. But for Merge Sort, only takes $O(n \log^2 n)$.

(c) Insertion Sort

As most items in A is already sorted($\log \log n$ is much smaller than n).

Problem 2-3.

- Binary Search is quite easy to think of, I'm not sure whether I should consider that the end of the island is submerging??

Problem 2-4.

Store a viewer as a struct,includes the viewer's ID,as well as an array store pointers that point to all messages the viewer had said. Store all viewers in a sorted order with merge sort which is $O(n \log n)$,with a comparison of each viewer's ID. Then store all messages as a linked list, a single node represents a viewer's message.

For those operations:

- $build(V)$: Sort all viewers by their ID using merge sort cost $O(n \log n)$.
- $send(v, m)$: Find viewer v with binary search cost $O(\log n)$,then sends a message, change the state of the struct v and the linked list both cost $O(1)$.
- $recent(k)$: Here we can assume that each time we insert a new message, we insert it in the front rather than from the end. Then we can return the k most recent messages in $O(k)$ time.
- $ban(v)$: Find viewer v cost $O(\log n)$ and delete all the message from the linked list which is stored in struct v cost $O(n_v)$. So totally cost $O(n_v + \log n)$.

Problem 2-5.

(a) Firstly, try to find out how many time scope we need to take.We can do this by checking every tuple in B_1 and B_2, apparently this can be done in $O(n)$ time.

Then, check every time scope to see how many rooms B_1 and B_2 have reserved and then B should take the amount of $B_1 + B_2$.

(b) Take (a), seems like merge sort

(c) Submit your implementation to alg.mit.edu.