Problem Set 2

Name: Your Name

Collaborators: Name1, Name2

Problem 2-1.

(a) An example of embedding images, in case you want to include a drawing of a tree.

$a = 2$
$b = 2$
$f(n) = O(n) = n^k(\log n)^p$
$k = 1$
$p = 0$
$\log a_b = 1 = k \longrightarrow$ **<u>case 2</u>**

solution:

$$O(n \log n)$$

(b)

$a = 3$
$b = \sqrt{2}$
$f(n) = O(n^4) = n^k(\log n)^p$
$k = 4$
$p = 0$
$\log a_b < k \longrightarrow$ **<u>case 3</u>**

solution:

$$O(n^4)$$

(c)

$a = 2$
$b = 2$
$f(n) = 5n \log n = n^k (\log n)^p$
$k = 1$
$p = 1$
$\log a_b = 1 = k \longrightarrow$ **<u>case 2</u>**

solution:

$$n(\log n)^2$$

(d) substitute $T(n) = n^2$

$n^2 = (n-2)^2 + \theta(n)$
$4n - 4 = \theta(n)$ $\square$

Problem 2-2.

(a) **Selection sort** will be the best,since it is **in-place** and we can do `get_at(i)` in $\theta(1)$ and `set_at(i, x)` in $\theta(n \log n)$ and **Selection sort** does `get_at(i)` $\theta(n^2)$ times and `set_at(i, x)` $\theta(n)$ time. The overall complexity is $\theta(n^2 \log n)$.
Insertion sort is **in-place** algorithm but it does `get_at(i)` $\theta(n^2)$ times and `set_at(i, x)` $\theta(n^2)$ time. The overall complexity is $\theta(n^3 \log n)$.
merge sort is not an **in-place** algorithm.

(b) **Merge sort** will be the best, since it does the minimum number of comparasions between **Selection sort** and **Insertion sort** as follows:

Merge sort $\rightarrow (n-1)$ comparasions $\rightarrow \theta(n \log^2 n)$
Selection sort $\rightarrow n!$ comparasions $\rightarrow \theta(n^2)$
Insertion sort $\rightarrow \dfrac{n^2 - 3n}{2}$ comparasions $\rightarrow \theta(n^2)$

(c) **Insertion sort** will be the best, Selection sort and Merge sort time complexity is independent on the input ($\theta(n^2)$, $\theta(n \log n)$) but Insertion sort is not so if we do $\log \log n$ swaps this means that Insertion sort will perform only $\log \log n$ swaps to undo all inversions which is $O(n)$, then the overall running time is $O(n)$.

Problem 2-3. He can apply the binary search concept as follows:

1. Go to the midle of the island $(n/2)$th kilometer.
2. Use the tracking device to determine whether he is north or south of his currunt place.
3. Use the telepotation and go to the middle of the distance between currunt place and water in this direction.
4. Repeat step 2 and 3 till you find Datum.

since Datum is definitely exists in the island this algorithm will find him and terminate in $O(\log k)$ locations got checked.

Problem 2-4. we need

1. a sorted array A(id, m_v, b) uses the ID as a key. has m_v as a pointer to a linked list containing all viewer v messages and a boolean b tells if he is banned or not.
2. a linked list L containing all sent messages to return last k.

To support `build(V)` initialize an empty array with size $n = |V|$, set all linked lists m_v to Null, all p to flase, and sort the array in $O(n \log n)$ time.

To support `send(v, m)` do a binary search in A to check if viewer v is banned using b in $O(\log n)$, Append the message in both m_v and L in O(1), so this takes $\log n$.

To support `recent(k)` if L contains $c < k$ element return it in o(c), else we should keep a pointer at the begining of the last k elements and start returning from this pointer till the end of L which takes $O \log n$.

to support `ban(v)` do a binary search in A (takes $O \log n$) and set b to true then start deleting his messages linked list which takes $o(n_v)$, so this algorithm takes $O(n_v + \log n)$.

Problem 2-5.

(a)

(b)

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