Name: yamchips

Collaborators: None

Problem 2-1.

(a) According to *Master Theorem*, suppose we have a recurrence of the form
$T(n) = aT(n/b) + f(n)$
where n is the size of input, a is the number of subproblems in the recursion, n/b is the size of each subproblem, $f(n)$ is the cost of the work done outside the recursion call, which includes the cost of dividing the problem and cost of merging the solutions.
$T(n)$ has the following asymptotic bounds:

1. if $f(n) = O(n^{log_b a - \epsilon})$, then $T(n) = \Theta(n^{log_b a})$.
2. if $f(n) = \Theta(n^{log_b a})$, then $T(n) = \Theta(n^{log_b a} logn)$.
3. if $f(n) = \Omega(n^{log_b a + \epsilon})$, then $T(n) = \Theta(f(n))$.

where $\epsilon > 0$ is a constant
In this case, $a = 4, b = 2$, that means $n^{log_b a} = n^2$, while $f(n) = O(n^{2-1})$, that makes $T(n) = \Theta(n^2)$ by case 1 of the Master Theorem. This is true no matter the choice of $f(n) \in O(n)$.

(b) According to *Master Theorem*, we have $a = 3, b = \sqrt{2}$ in this case, that means $n^{log_b a} = n^{2log_2 3}$. If $f(n) = n^4$, then $T(n) = O(n^4)$ by case 3 of the Master Theorem, since $f(n) = n^4 = n^{2log_2 3 + \epsilon}$. If $f(n) = 0$, then $T(n) = \Omega(n^{2log3})$ by case 1 of the Master Theorem, since $0 \in O(n^{2log3 - \epsilon})$ for any positive $\epsilon < 2log3$.

(c) According to the assumption, we have $a = 2, b = 2$, that means $n^{log_b a} = n$. $f(n) = 5nlogn$, so $T(n) = \Theta(nlog^2 n)$ by case 2 of the Master Theorem.

(d) Assuming $T(n) = cn^2$, then we have:

$$T(n) - T(n-2) = cn^2 - c(n-2)^2 = 4cn - 4c = \Theta(n)$$

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

Problem 2-2.

(a) Merge sort is not in-place, so we can not choose it.
Insertion sort: $\Omega(n^2)$ comparisons and $\Omega(n^2)$ swaps, so the total time cost of getting items is $\Omega(n^2)$, setting $\Omega(n^3 logn)$.
Selection sort: $\Omega(n^2)$ comparisons and $O(n)$ swaps, so the total time cost of getting items is $\Omega(n^2)$, setting $O(n^2 logn)$.
So, we choose selection sort.

(b) Merge sort: $logn$ recursive calls, in each call it takes $O(nlogn)$ time, so the total cost is $O(nlog^2 n)$.
Selection sort: time cost in comparison is $\Omega(n^2 logn)$, swap $O(n^2)$, so the total cost is $\Omega(n^2 logn)$.
Insertion sort: time cost in comparison is $\Omega(n^2 logn)$, swap $O(n^3)$, so the total cost is $O(n^3)$.
So, we choose merge sort.

(c) Merge sort: the total time cost is $\Theta(nlogn)$.
Selection sort: the total time cost is $\Theta(n^2)$.
Insertion sort: Because there are $loglogn$ unsorted adjacent items in A, so the total time cost is $O(n + loglogn) = O(n)$.
So, we choose insertion sort.

Problem 2-3. Use binary search, starting from either end. First search 1 km, then 2 km, etc. Finally we will reach 2^i km where we passed Datum. That means $2^{i-1} < k < 2^i$, then $i - 1 < logk < i$. Until now, the time cost is $i = O(logk)$. Then for the remaining distance between 2^{i-1} and 2^i, we use binary search to check which integer kilometer Datum is at. The amount of integer point is $2i$, so the search takes $log2i$ time. Since $i < 1 + logk$, that makes the second search $O(logk)$ time. So, the total amount of searching time is $O(logk)$.

Problem 2-4. The database we build contains two data structures:

1. a doubly-linked list L containing the sequence of all undeleted messages in the chat in chronological order
2. a sorted array S of pairs (v, p_v) keyed on v, where v is the viewer ID and p_v is a pointer to a viewer-specific singly-linked list L_v storing pointers to all the nodes in L that containing a message sent by the viewer. When L_v points to None, that means viewer v has been banned.

To support $build(V)$, initialize L to be an empty linked list in $O(1)$ time, initialize S of size $n = |V|$ containing (v, p_v) for each viewer $v \in V$ in $O(n)$ time, initialize the empty linked list L_v for each viewer in $O(1)$ time, and use merge sort to sort S in $O(nlogn)$ time. This operation takes worst-case $O(nlogn)$ time in total, and maintains the invariants of the database.
To support $send(v, m)$, use binary search to find v in S in $O(logn)$ time, then insert m at the front of L in $O(1)$ time and insert the pointer to the node in L into L_v in $O(1)$ time. This operation takes worst-case $O(logn)$ time in total, and maintains the invariants of the database.
To support $recent(k)$, return the first k nodes in L. As long as the invariants on the database are correct, this operation directly returns the requested messages in worst-case $O(k)$ time.
To support $ban(v)$, find the viewer v in S using binary search, which takes $O(logn)$ time, then the n_v pointers direct to n_v nodes in L, remove the nodes by relinking pointers in L in $O(1)$ time. Finally, set p_v to None. This operation takes $O(n_v + logn)$ time in worst-case and is correct because it maintains the invariants of the database.

Problem 2-5.

(a)

(b)

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