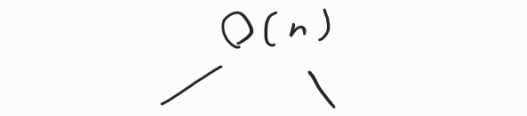
PS 2

2-1

(a) $n^{\log_{(n)}(a)} = n^{\log_{(2)}(4)} = n^2 > O(n)$,

So $T(n) = \Theta(n^2)$ by master theorem.



$4^{\log(n)} \cdot c = 4^{(\log(n))-1} \cdot 2c$

$\sum_{i=0}^{\log(n)} 4^i cn/2^i = cn\left(2^{\log(n)+1} - 1\right) \le ecn^2$

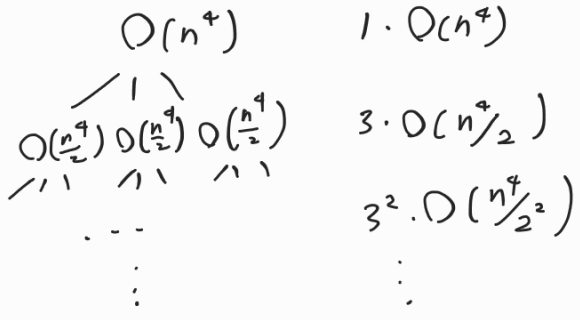
$= O(n^2)$

(b)

by Master Theorem: $n^{\log_{(3)}(a)} = n^{\log_{(\sqrt{2})}(3)} = n^{\log_{(2)}(9)} < O(n^4) = f(n)$.

So $T(n) = \Theta(f(n)) = O(n^4)$

by recursion tree:



$1 \cdot O(n^4)$

$3 \cdot O(n^4/2)$

$3^2 \cdot O(n^4/2^2)$

at the level of leaves,

$3^{\log(n^4)}$

$O(1) = C \cdot 3^{\log(n^4)}$

$\sum_{i=0}^{\log(n^4)} \left(3^i \cdot \frac{cn^4}{2^i}\right) = cn^4 \sum_{i=0}^{\log(n^4)} \left(\frac{3}{2}\right)^i$

$= cn^4 \cdot \frac{\left(\frac{3}{2}\right)^{\log(n^4)} - 1}{3/2 - 1}$

$= 2cn^4\left(3^{\log(n^4)}/n^4 - 1\right)$

2-2

(a) The choice can be justified by comparing sorting algorithms each other in this context after analyzing each algorihtm.

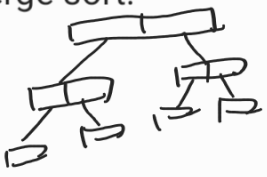
selection sort:

Traversing data structure and finding biggest one, O(n), swapping the biggest one and the last of the sequence, $2\left(\Theta(1) + \Theta(n\log(n)\right)$, and this recursively, $\Theta(n)$.

So worst-case time equlas to:

$$\left(O(n) + 2\left(\Theta(1) + \Theta(n\log(n))\right)\right)\Theta(n) = \Theta(n^2\log(n))$$

insertion sort:
Insertion sort follows the procedure: start at the first of the sequence, moving to the left(start) until metting smaller one or arriving at the tip, and repeat this recursively by moving right(end) byt 1.
Each of the steps can be evaluated:
moving to the left: By repeated swapping, it can be done. Swap take on $2(\Theta(n\log(n)) + \Theta(1))$ time, by executing get_at and set_at methods respectively twice. Thus, worst-case time is $\Theta(n^2\log(n))$
repeatedly moving right and executing the previous step:
$\Theta(nf(n))$, where $f(n) = \Theta(n^2\log(n))$
Consequently, worst-case time is $\Theta(n^3\log(n))$
merge sort:



The height of the tree is log(n).
At each level, the number of nodes is 2^h, where h = 0, 1, ..., log(n) - 1.

In each level, h, (n/(2^h)) times of comparisons occur at a node, and at each comparison, move of a data occurs by set_at and get_at. So summing the tasks,

$$n\Theta(n\log(n)) + 2\cdot\frac{n}{2}\cdot\Theta(n\log(n)) + 2^2\cdot\frac{n^2}{2^2}\cdot\Theta(n\log(n)) + \cdots + \frac{n}{2^{\log(n)-1}}\cdot\Theta(n\log(n)) = \Theta(n^2\log^2(n)).$$

Comparing the three sorting algorithms, selection sort is the best choice.

(b)
Merge sort is the best choice.
Merge sort is a sort of divide-and-conquer algorithm, the precedure can be represented by a binary tree. The height of the tree is log(n), where n is the size of input. At each level, there occur n comparisons in worst-case, so for the data structure provided it takes $\Theta(n\log^2(n))$ time for sorting by merge sort. This is pretty better than sorting algorithms in $\Theta(n^2)$ time.

(c)
Let U denote a set containing elements have to be sorted in A. |U| ~ 2log(log(n)). When apply insertion sort or merge sort, it takes $\Theta(n^2)$ and $\Theta(n\log(n))$ time respectively regardless of |A|. But when use selection sort, it takes $\Theta(n\log(\log(n)))$ time in worst-case. So insertion sort is the best choice for this situation.
This is justfication. Selection sort is executed until A is sorted such that it takes |A| times of selection. After |A| times of selections, A is sorted and no more selection is needed and finally it's done. So the process time of insertion sort in this case is,
(the number of selection) * (time to select the biggest one of $T \subseteq A$),
and $\Theta(n\log(\log(n)))$

2-3.
Let d denote the distance by kilometer from north of the island. Picard starts at d = 1. And he moves to d', where d' = 2d. Repeating this task. When he encounters a signal saying that Datum is on north of Picard, then Picard does a binary search in [d/2, d].

Thm. If Datum is at d = k, where $2^t \le k \le 2^{t+1}$, this algorithm makes Picard visit O(log(k)) locations.
Pf. Picard visits d = 2^0, 2^1, ... , 2^t, 2^(t + 1), so the number of locations Picard has visited is O(log(2^t)) = O(log(k)). Picard gets a signal such that he stops moving at 2^(t+1), and then he does binary search within [2^t, 2^(t + 1)] to find Datum. The binary search takes O(log(2^(t + 1) - 2^t)) = O(log(2^t)) = O(log(k)).
Total time needed is O(log(k)). □

2-4.
build(U): Sort V and store all elements in V in a set data structure. Each element stores ID and points to a data structure storing messages the user of the ID sent. And initialize a new chat room devised by dynamic array. And build a ban database to store positions of deleted messages of chat database.
send(v, m): Send message m to the chat and store the position of the chat in the user data base. Each user storage in user database points to a data structure and stores the message in it. The user database has been sorted when it was build, so finding the user in the database takes O(log(n)) time by binary search. And storing message to the chat database takes O(1) time since it was implemented as a dynamic array.
ban(v): Find the user v by binary search, which takes O(log(n)) time. Subsequently, using the position data, find every message v had written such that delete messages, and store the position of deleted message to the ban database. Chat database and ban database had been devised by dynamic array so that given the position, getting the elements from chat database and inserting the deleted messages to last of chat database take O(1) time respectively. So total of this operation is

$$O(1) \cdot n_v + O(\log(n)) = O(n_v + \log(n))$$

recent(k): From last of the chat database, pick one element repectively by k times. Each step includes a procedure to check the position of the last deleted message from ban database. And skip the removed messages such that the number of steps be k. So opeartion time is O(k).