Quiz 03.

Q1.

Step 1.

Let $S(j)$ be the maximum sum of all the decreasing subsequence in the given sequence. So the sum at a_{ij} is the sum of a_{i_k} ($a_{i_k} > a_{ij}$ and $k < j$) plus a_{ij}.

Thus $S(j) = S(k) + a_{ij}$ ($a_{i_k} > a_{ij}$ and $k < j$).

The sum at a_{ij} leads to the maximum sum from a_1 to a_{ij}.

Step 2:

In the optimal solution, we have:

$$S(j) = \begin{cases} a_1 & j = 1 \\ \max_{1 \le k < j} \{ S(k) \mid a_{i_k} > a_{ij} \} + a_j & \text{otherwise} \end{cases}$$

Thus, $\max_{1 \le i \le n} \{S(i)\}$ is the maximum solution.

Step 3:

For each iteration in the way of finding $S(1), S(2), S(3) \dots S(i)$ in increasing order of i, we have its running time is $O(i)$. So the total running time is $O(n^2)$.

Thus, $T(n) = \begin{cases} \Theta(1) & n = 1 \\ O(n^2) & n > 1. \end{cases}$

step 4:

Because this problem is to find an optimal solution, it doesn't require to construct the optimal solution. step 4 is omitted.

Q 2.

Let W be the knapsack capacity and the weight w_i with the profit p_i for ~~all the~~ every item i with $1 \le i \le n$.

Let $i_1, i_2, \dots i_n$ be the decreasing order of $\frac{p_i}{w_i}$.

The greedy strategy is to pack items $i_1, i_2, i_3, \dots i_{j-1}$ if $w_{ij} \ge W - \sum_{i=1}^{j-1} w_{i_i}$ and for item i_j, the capacity is $W - \sum_{i=1}^{j-1} w_{i_i}$

Then the maximum profit S is $S = \sum_{i=1}^{j-1} p_{i_i} + \frac{p_{ij}}{w_{ij}} \cdot (W - \sum_{i=1}^{j-1} w_{i_i})$

Prove:

We assume that there is another item i_n and $n > j$ will be added in the solution. Then ~~it will be~~ the same amount of item i_k ($k \le j$) will be removed. Then the updated solution S' is:

$$S' = \sum_{i=1}^{j-1} p_{i_i} + \frac{p_{ij}}{w_{ij}} \cdot (W - \sum_{i=1}^{j-1} w_{i_i}) + \left(\frac{p_{in}}{w_{in}} \cdot w'_{i_k} - \frac{p_{i_k}}{w_{i_k}} \cdot w'_{i_k} \right)$$

$$= S + w'_{i_k} \left(\frac{p_{in}}{w_{in}} - \frac{p_{i_k}}{w_{i_k}} \right) \le S \quad \text{as } \frac{p_{i_k}}{w_{i_k}} \ge \frac{p_{in}}{w_{in}}.$$

Thus, the solution S is optimal. And if the items are sorted, the running time is $O(n)$, otherwise, it takes $O(n \log n)$ time.

Q3.

Assume that there're 2 characters x and y with frequencies f_x and f_y, $f_x < f_y$. let C_x and C_y be the length of codeword of x and y.

Let A be the set of characters. Assume there exists a method for the character coding C_i for the characters in A such that the length of the document after coding is $\Sigma_{i \in A} f_i \cdot C_i$ is minimized. Thus.

$$C_x > C_y \quad \text{if } f_x < f_y.$$

Then we assume there is another coding C'_x, C'_y for the character x and y. For example, we swap the coding between x and y. For any $i \in z$, we have $C'_i = C_i$, $C'_x = C_x$, $C'_y = C_y$ and we have $C'_x < C'_y$ after swaping because $C_x > C_y$.

Thus,

$$\Sigma_{i \in A} f_i \cdot C'_i = \Sigma_{i \in A} f_i \cdot C_i - (f_x \cdot C_x + f_y \cdot C_y) + (f_x \cdot C_y + f_y \cdot C_x)$$

$$= \Sigma_{i \in A} f_i \cdot C_i + f_y (C_x - C_y) - f_x (C_x - C_y)$$

$$> \Sigma_{i \in A} f_i \cdot C_i \quad \text{where } f_x < f_y \text{ and } C_x > C_y$$

So this solution implies that there exists a longer length of the document which contradicts that the length is the smallest one. Hence, if we need to minimize the encoding length of the document, for every coding C, we must have

$$C_x > C_y \text{ if } f_x < f_y.$$

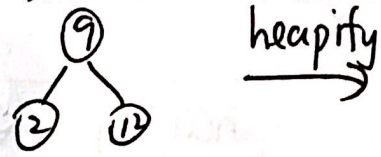
Q4.

1. Insertion:

Insert 9:

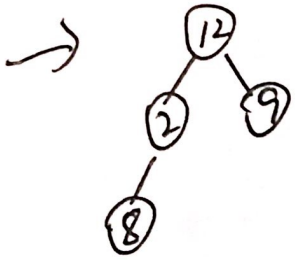
Insert 2

Insert 12



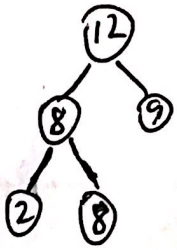
heapify

Insert 8

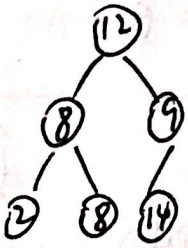


heapify

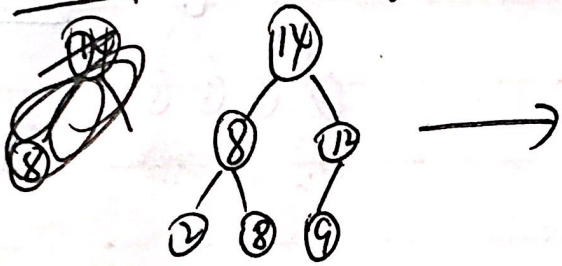
Insert 8



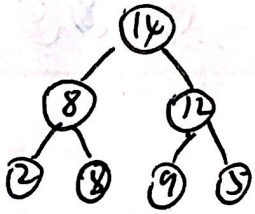
Insert 14



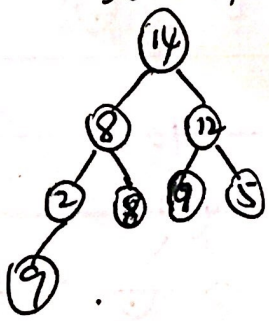
heapify ~~Insert 5~~ ~~Insert 8~~



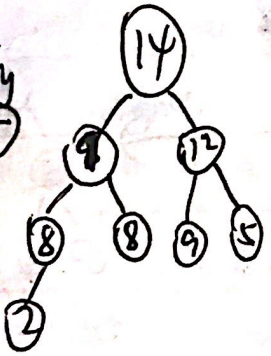
~~Insert~~ Insert 5



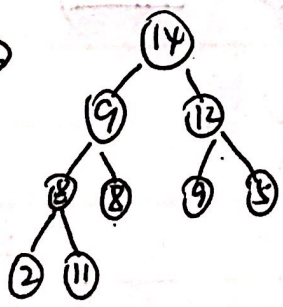
Insert 9



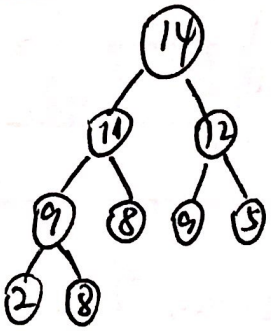
heapify



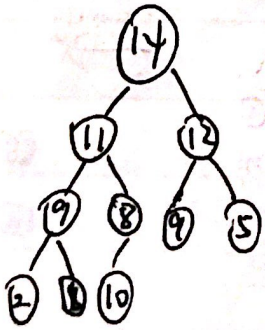
Insert 11



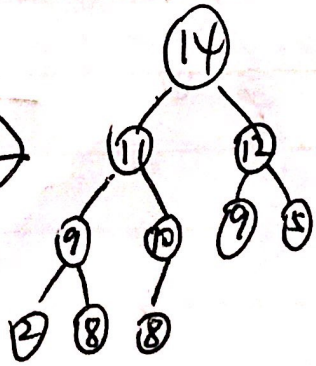
heapify



Insert 10



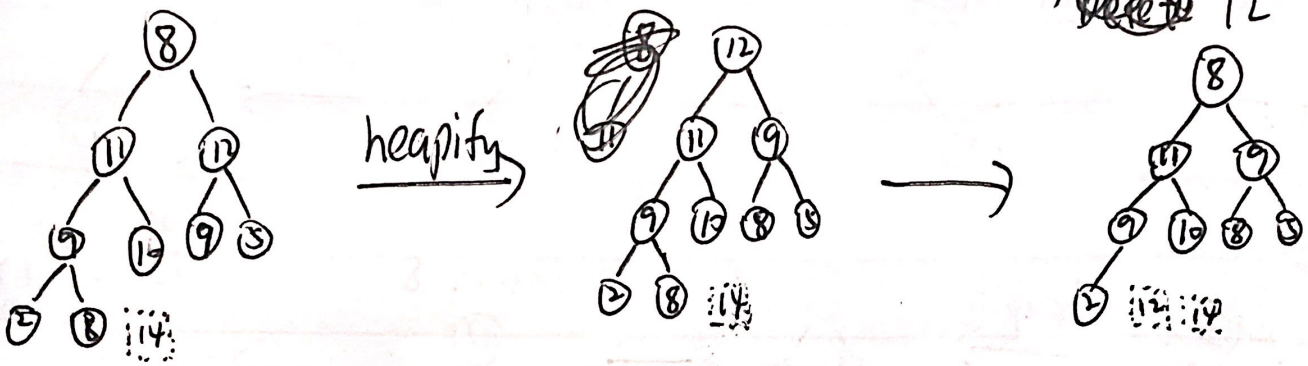
heapify

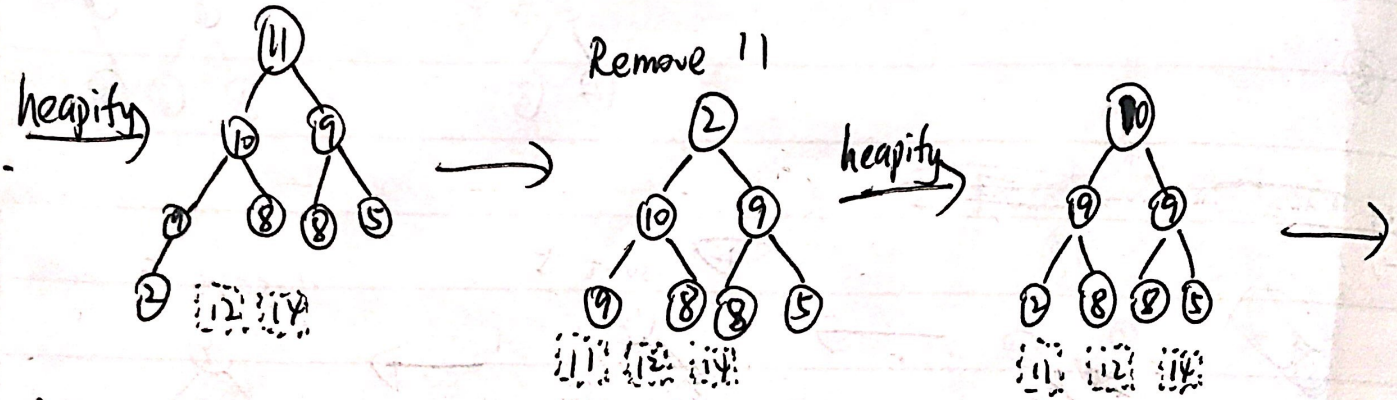


Done

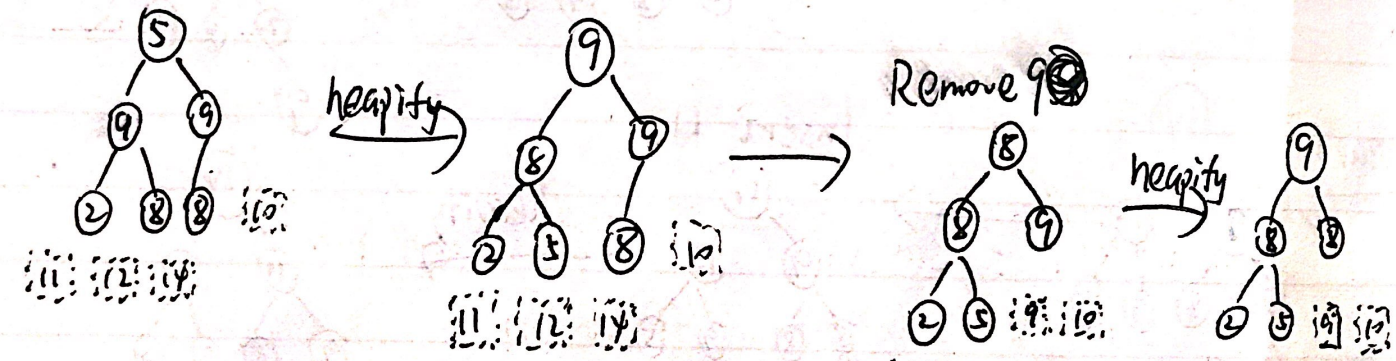
Deletion

Remove 14

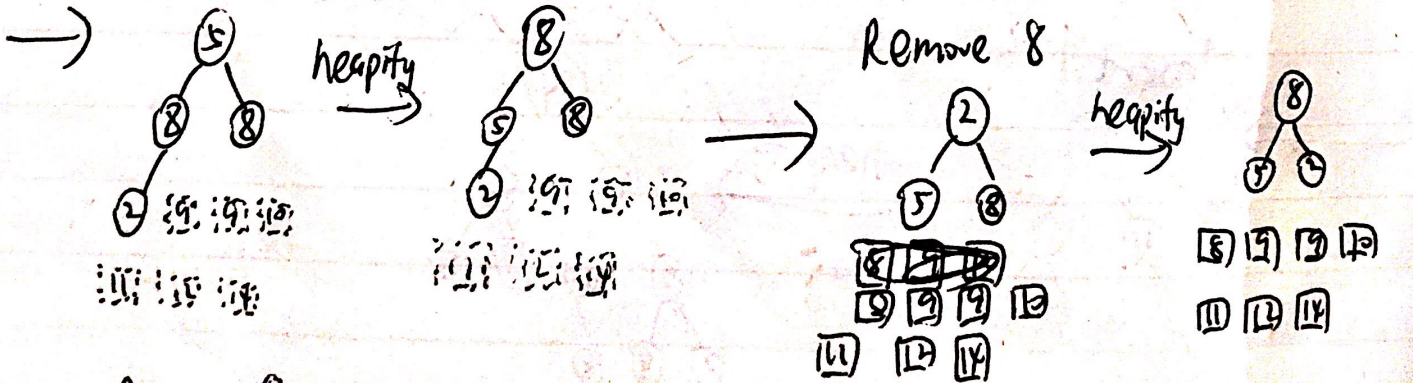




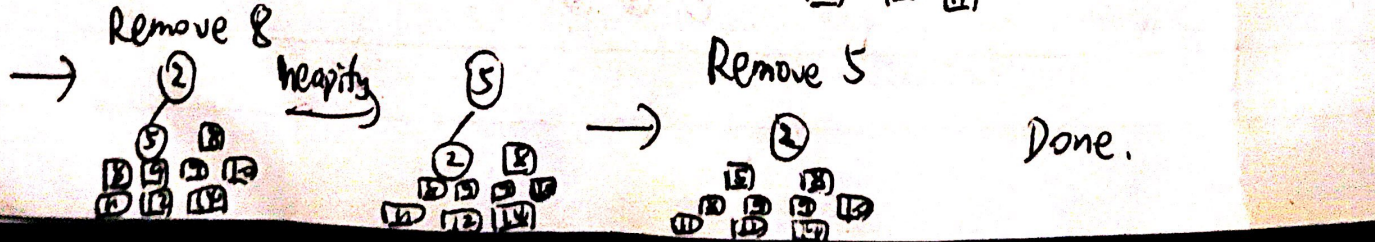
Remove 10



Remove 9



Remove 8



Done.

2.
If there're n elements, the time complexity is $O(n \log n)$

{ The initialization of a max-heap takes $O(1)$.
Heapify takes $O(\log n)$.
Remove takes $O(1)$
Replace the root and heapify takes $O(\log n)$

So the total is $O(n \log n)$.

6. (1)
First, we assume there're n slots in the hash table.

linear probing: there're $O(n)$ probing sequences.
Since we have $h(k,i) = (h'(k)+i) \bmod n$.
$0 \le h'(k) \le n-1$ and $0 \le i \le n$. as $h'(k)$ leads to a different probing sequence.

Quadratic probing: there're $O(n)$ probing sequences.
Since we have $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod n$
$0 \le h'(k) \le n-1$ and $0 \le i \le n$ with c_1, c_2 two constants, as different $h'(k)$ leads to a different probing sequence.

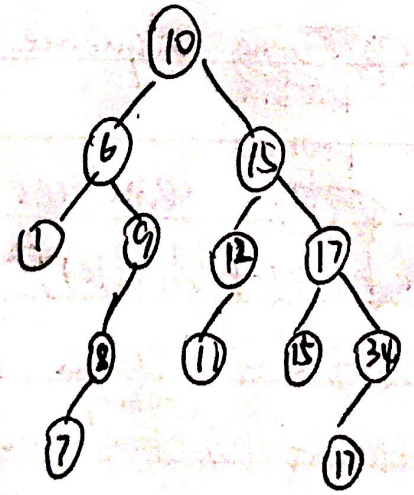
Double hashing: there're $O(n^2)$ probing sequences.
Since we have $h(k,i) = (h_1(k) + i h_2(k)) \bmod n$.
$0 \le h_1(k), h_2(k) \le n-1$ and $0 \le i \le n$ as a different pair of $(h_1(k), h_2(k))$ leads to a different probing sequence and there are $O(n^2)$ different pairs.

(2)

	Advantages	Disadvantages
linear probing	Simple takes less time	Can't avoid primary clustering problem
Quadratic probing	avoids primary clustering problem with easy computation	Can't avoid second clustering proble
Double hashing	avoids both primary and second clustering problem	takes longer running time

Q5

(1) Final binary search tree:



(2) Insert new element 12: