

1. My Answer: (selection sort)

THISQUESTION
 → EHISQUTSTION
 → EHISQUTSTION
 → EHI IQUTSTSON
 → EHI INUTSTSOQ
 → EHI INOTSTSUQ
 → EHI INOQSTSTUT
 → EHI INOQSTSTUT
 → EHI INOQSSSTUT
 → EHI INOQSSSTUT
 → EHI INOQSSSTTU

2. My Answer: (insertion sort)

THISQUESTION
 → HTISQUESTION
 → HIT SQUESTION
 → HI STQUESTION
 → HI QST VESTION
 → HI QST UESTION
 → EHI QSTUSTION
 → EHI QSS TUTION
 → EHI QSS TTUION
 → EHI IQSSSTTUON
 → EHI IOQSSSTTUN
 → EHI INOQSSSTTU

3. My Answer: (shell sort- 5,2,1)

MUCHLONGERQUESTION
 $h=5 \rightarrow$ I UCHLMNGERQUESTON
 → INCHLM OGERQUESTUN
 → INCHLM OEEROU GSTQUN
 → INCELM OEHROUGSTQUN
 → INCELM OEHROUGSTQUN
 $h=2 \rightarrow$ CNGEHMIELROUSTQUN
 → CE G E H M I N L N O Q O R T S U U
 $h=1 \rightarrow$ C E E G H I L M N N O O Q R S T U U

4. My Answer: Yes, if all keys are identical, insertion sort definitely work better than selection sort.
 My reason:

Insertion sort has to sweep the sorted list each time to determine where to insert each element, and have best case of $O(N)$, but selection sort works by taking the smallest element in an unsorted array and bring it to the front, it's only have $O(N^2)$ best case, apparently insertion sort work better than selection sort.

5. My Answer: No, if all keys are in reverse order, insertion sort does not work better than selection sort.

My reason:

Because both sorting algorithms compare the same amount of pairs, but insertion sort exchange more than selection sort. In conclusion, selection sort work better than insertion sort.

6. My Answer: In python, we can use `set()` constructor, pass the list as argument to the set constructor, and it returns a set of unique items. If we can't use `set()` constructor, we can also use for loop to iterate over the elements of the list, and check if the element has occurred only once, than choose a sorting algorithm to sort the results.

7. My Answer: Figure1 is my Python code

```

1 from collections import defaultdict as DF
2 import bisect # If you want to use insertion sort, you can use bisect.insert()
3
4 # Creat Jumble hash-map to put sorted words
5 Jumble = DF(list)
6
7 def GroupJumbles(Words):
8     for i in Words:
9         # Take letters from list words and sorting
10        Letter = str(sorted(i))
11        Jumble[Letter].append(i)
12
13    for Letter in Jumble:
14        print("Jumbles:")
15        print(" ".join(Jumble[Letter]))
16
17 Words = ["racing", "secura", "saucer", "caring", "random"]
18 GroupJumbles(Words)

```

Ln: 18, Col: 20

Run Share Command Line Arguments

```

Jumbles:
racing caring
Jumbles:
secura saucer
Jumbles:
random

```

** Process exited - Return Code: 0 **
Press Enter to exit terminal

Figure 1.

8. My Answer: Because in the h-sequence 1, 2, 4, 8, ..., 2^k , $2k$ is redundant in the sorting, we can just easily sort k , and k -length sequence will cover all $2k$ -length position, the example below will demonstrate the worst case.

ex: sorts the letters H G F E D C B A with the h-sequence 4, 2, 1.

	H	G	F	E	D	C	B	A											
$h=4 \rightarrow$	D	G	F	E	H	C	B	A	$h=2 \rightarrow$	B	C	D	A	F	G	H	E		
	\rightarrow	D	C	F	E	H	G	B	A		\rightarrow	B	A	D	C	F	E	H	G
	\rightarrow	D	C	B	E	H	G	F	A		$h=1 \rightarrow$	A	B	C	D	E	F	G	H
	\rightarrow	D	C	B	A	H	G	F	E										

9. My Answer: Because we won't get better performance to use selection sort for h-sequence in the shell sort. For example, if we use k -sequence for the shell sort and a list of N length, then the complexity will becomes $O(kN^2)$, which is worse then the best case of selection sort $O(N^2)$. In conclusion, the reason above explained that why not to use selection sort for h-sequence in the shell sort.