1)
Selection Sort.
Assuming sorting in ascending order:
T H I S Q U E S T I O N (start)
E H I S Q U T S T I O N (swap T with E)
E H I S Q U T S T I O N (H stays)
E H I S Q U T S T I O N (I stays)
E H I I Q U T S T S O N (swap S with I)
E H I I N U T S T S O Q (swap Q with N)
E H I I N O T S T S U Q (swap U with O)
E H I I N O Q S T S U T (swap T with Q)
E H I I N O Q S T S U T (S stays)
E H I I N O Q S S T U T (swap T with S)
E H I I N O Q S S T U T (T stays)
E H I I N O Q S S T T U (swap T with U)
E H I I N O Q S S T T U (sorted)

2)
Insertion Sort:
Assuming sorting in ascending order:
T H I S Q U E S T I O N (start)
H T I S Q U E S T I O N (put T after H)
H I T S Q U E S T I O N (put I after H)
H I S T Q U E S T I O N (put S after I)
H I Q S T U E S T I O N (put Q after I )
H I Q S T U E S T I O N (U same place)
E H I Q S T U S T I O N (put E in front)
E H I Q S S T U T I O N (put S after first S)
E H I Q S S T T U I O N (put T after first T)
E H I I Q S S T T U O N (put I after first I)
E H I I O Q S S T T U N (put O after I)
E H I I N O Q S S T T U (put N after I)
E H I I N O Q S S T T U (sorted)

3)
Shell sort h sequence 5, 2, 1
M U C H L O N G E R Q U E S T I O N (start h=5 sequences first)

$\underline{M}$ U C H L $\underline{O}$ N G E R $\underline{Q}$ U E S T $\underline{I}$ O N (insertion sort underlined: MOQI)
I U C H L M N G E R O U E S T Q O N

I $\underline{U}$ C H L M $\underline{N}$ G E R O $\underline{U}$ E S T Q $\underline{O}$ N (insertion sort underlined: UNUO)
I N C H L M O G E R O U E S T Q U N

I N $\underline{C}$ H L M O $\underline{G}$ E R O U $\underline{E}$ S T Q U $\underline{N}$ (insertion sort underlined: CGEN)

I N C H L M O E E R O U G S T Q U N

I N C H L M O E E R O U G S T Q U N (insertion sort underlined: HES)
I N C E L M O E H R O U G S T Q U N

I N C E L M O E H R O U G S T Q U N (insertion sort underlined: LRT)
I N C E L M O E H R O U G S T Q U N

(h=2 now)
I N C E L M O E H R O U G S T Q U N (insertion sort underlined: ICLOHOGTU)
C N G E H M I E L R O U O S T Q U N

C N G E H M I E L R O U O S T Q U N (insertion sort underlined: NEMERUSQN)
C E G E H M I N L N O Q O R T S U U

(h=1 now)
C E G E H M I N L N O Q O R T S U U (just do insertion sort all whole list for this part)
C E E G H I L M N N O O Q R S T U U (after the h=1, this is the result and we are done)

4)
Yes insertion sort does work better than selection sort if all keys are identical. Selection sort will still traverse the array N+(N-1)+(N-2)+...+1 which is $O(N^2)$. Insertion sort compares the adjacent key and if they are all the same, the inner loop breaks and so insertion will take only $O(N)$.

5)
Selection sort will work better for all keys in reverse order (well it depends on what you define work). Computational complexity wise, both selection and insertion will be $O(N^2)$ for reverse order. However, insert sort requires $O(N^2)$ search and $O(N^2)$ swaps, but selection sort will only require $O(N^2)$ search but $O(N)$ swap.

6)
Let's say list A is a list with duplicate items (e.g. [1, 5, 3 , 2, 2, 2, 4, 4, 5]). We first sort the list A using a sorting algorithm ([1, 2, 2, 2, 3, 4, 4, 5, 5]). Then we go through elements in list checking A[i] and A[i+1]. We start a new empty list and add the element to it if A[i] != A[i+1]. This will produce a list of unique items.

7)
We first created new empty list and put the results of the each word sorted into that. Then we use a nested loop to print out the duplicates next to each other. Make sure to use a flag so that multiple items are not printed multiple times. The code is below on the next page.

```python
from functools import reduce

li = ['racing', 'secura', 'saucer', 'caring', 'random']
def jumble(ali):
    result = []
    for item in ali:
        result.append(reduce(lambda a, b : a + b, sorted(item)))
    flag = []
    for i in range(len(ali)):
        if(i in flag):
            continue
        else:
            print(ali[i], end=" ")
            for j in range(i+1, len(ali)):
                if(result[i] == result[j]):
                    flag.append(j)
                    print(ali[j], end=" ")
            print("\n")

print(li)
jumble(li)
```

```
['racing', 'secura', 'saucer', 'caring', 'random']
racing caring

secura saucer

random
```

8)

The h-sequence of powers of 2 is bad for shellsort because it doesn't compare odd numbers until the end at h=1. So if things are very out of order in odd places, it will bad. Using sequences of powers of 2, the complexity is $O(N^2)$ with no improvement over insertion sort.

An example is: [200, 50, 190, 40, 180, 30, 170, 20, 160, 10, 150]

When input is in reverse order, it is the worst case for each step of the insertion sort of powers of 2 and also the largest numbers are in the odds and in reverse.

9)

A reason selection sort is not used for h-sorting in shellsort is that selection sort is slower than insertion sort when the list is close to being all sorted. After each h-step, the shellsort becomes more partially sorted, so there's an advantage to using insertion sort.