1. How many comparisons and interchanges (in terms of file size n) are performed by Simple insertion sort

Key take away is that items that are smaller are more work to insert, items that are biggest are less work to insert since the just go on the end of the sorted part.

i) A sorted file

The cost is just O(n-1) because there are **n-1 comparisons and 0 interchanges**. You pass through the unsorted portion pulling in the first item from the unsorted portion into the sorted portion. You're basically verifying the file is sorted, which it is already.

ii) A file that is sorted in reverse order (that is, from largest to smallest)

Let's see:
say we have 5,4,3,2,1
Remember the 5 is the first item in the sorted part already.

then: 10 interchanges

Comparison.	Interchanges.	New input	Original
1	1	4,5,3,2,1	5,4,3,2,1
2	2	3,4,5,2,1	4,5,3,2,1
3	3	2,3,4,5,1	3,4,5,2,1
4	4	1,2,3,4,5	2,3,4,5,1

There would be **2n comparisons and 2n swaps.**

iii) A file in which x[0], x[2], x[4]... are the smallest elements in sorted order, and in which x[1], x[3], x[5]... and the largest elements in sorted order

Say we have 1,4,2,5,3,6
1 is in the sorted position already.

Comparison.	Interchanges.	New input	Original
1	0	1,4,2,5,3,6	1,**4**,2,5,3,6
2	1	1,2,4,5,3,6	1,4,**2**,5,3,6
1	0	1,2,4,5,3,6	1,2,4,**5**,3,6
3	2	1,2,3,4,5,6	1,2,4,5,**3**,6
1	0	1,2,3,4,5,6	1,2,3,4,5,**6**

comparisons: 1.5n about and interchanges: n/2

Say we have 1,5,2,6,3,7,4,8
1 is in the sorted position already.

Comparison.	Interchanges.	New input	Original
1	0	1,5,2,6,3,7,4,8	1,**5**,2,6,3,7,4,8
2	1	1,2,5,6,3,7,4,8	1,5,**2**,6,3,7,4,8
1	0	1,2,5,6,3,7,4,8	1,2,5,**6**,3,7,4,8

3	2	1,2,3,5,6,7,4,8	1,2,5,6,**3**,7,4,8
1	0	1,2,3,5,6,7,4,8	1,2,3,5,6,**7**,4,8
4	3	1,2,3,4,5,6,7,8	1,2,3,5,6,7,**4**,8
1	0	1,2,3,4,5,6,7,8	1,2,3,4,5,6,7,**8**

In this case, there were 13 comparisons and 6 exchanges.

So maybe the better answer is that there are somewhere between 1.25 and 1.5 comparisons and n/2 and 3/4n exchanges?

2. How many comparisons and interchanges (in terms of file size n) are performed by Shell Sort using increments 2 and 1 for the following files:
i) A sorted file
ii) A file that is sorted in reversed order (that is from largest to smallest)
iii) A file in which x[0], x[2], x[4]... are the smallest elements in sorted order, and in which x[1], x[3], x[5]... and the largest elements in sorted order

i) Say file is 1,2,3,4,5,6
If k = 2,
One subfile is: 1,3,5
Another subfile is 2,4,6
We then insertion sort on each subfile but since they are already in order then only 2 comparisons for each are made and no interchanges.

When we aggregate it back up, it's 1,2,3,4,5,6. Then we do another insertion sort and it's n-1 comparisons and no interchanges.

So comparisons for this are 2+2+5=9=**3/2*n comparisons and 0 interchanges.**

ii) Say file is 6,5,4,3,2,1
One subfile is 6,4,2
Another is 5,3,1
to sort subfile each is 3 comparisons, 3 interchanges.

Then reaggregated it's 2,1,4,3,6,5 and then you do one more insertion sort.

Original	New	Comparisons	Interchanges
2,**1**,4,3,6,5	1,2,4,3,6,5	1	1
1,2,**4**,3,6,5	1,2,4,3,6,5	1	0
1,2,4,**3**,6,5	1,2,3,4,6,5	2	1
1,2,3,4,**6**,5	1,2,3,4,6,5	1	0
1,2,3,4,6,**5**	1,2,3,4,5,6	2	1

Comparisons: 3 + 7 = 10 = n + (2/3)*n
Interchanges = 6 = n

iii) Say file is 1,4,2,5,3,6
Oh this is interesting!
Then one subfile is 1,2,3
Another subfile is 4,5,6

2 comparisons are made for each and no exchanges
recombined we have: 1,4,2,5,3,6
using insertion sort again, we have:

comparisons	exchanges	original	new
1	0	1,4,2,5,3,6	1,**4**,2,5,3,6
2	1	1,2,4,5,3,6	1,4,**2**,5,3,6
1	0	1,2,4,5,3,6	1,2,4,**5**,3,6
3	2	1,2,3,4,5,6	1,2,4,5,**3**,6
1	0	1,2,3,4,5,6	1,2,3,4,5,**6**

Comparisons: 2+2+8 = 12 = 2n
Interchanges: 0 + 3 = 3 = n/2

3. Determine the number of comparisons (as a function of n and m) that are performed in merging two ordered files a and b of sizes n and m, respectively, by the merge method presented in the lecture, on each of the following sets of ordered files:
a. m=n and a[i] < b[i] < a[i+1] eg: a = [6,9,12,15,29,37] and b = [8,10,14,25,33,45]

a[i] refers the value in position i of file a, etc.

Comparisons	sorted array
6 and 8	6
8 and 9	6,8
9 and 10	6,8,9
10 and 12	6,8,9,10
12 and 14	6,8,9,10,12
14 and 15	6,8,9,10,12,14
15 and 25	6,8,9,10,12,14,15
25 and 29	6,8,9,10,12,14,15,25
29 and 33	6,8,9,10,12,14,15,25,29
33 and 37	6,8,9,10,12,14,15,25,29,33
37 and 45	6,8,9,10,12,14,15,25,29,33,37

comparisons: 11 and so **11 = 2n -1 comparisons**

b. m=n and a[n] < b[1]. eg: a = [2,5,9] b = [12,14,16]

Comparisons	sorted array
2 and 12	2
5 and 12	2,5
9 and 12	2,5,9

once an array is done, the rest in the remaining array goes in so only 3 comparisons are needed for the 6 items in the example. so 3 = **n comparisons were needed**

4. Determine the number of comparisons (as a function of n and m) that are performed in merging two ordered files a and b of sizes n and m, respectively, by the merge method presented in the lecture, on each of the following sets of ordered files:
a. m=n and a[n/2] < b[1] < b[m] < a[(n/2)+1]

So say:
a = [2,5,7,55,61,72]
b = [9,15,17,21,29,46]

Comparisons	sorted array
2 and 9	2
5 and 9	2,5
7 and 9	2,5,7
9 and 55	2,5,7,9
15 and 55	2,5,7,9,15
55 and 17	2,5,7,9,15,17
21 and 55	2,5,7,9,15,17,21
29 and 55	2,5,7,9,15,17,21,29
46 and 55	2,5,7,9,15,17,21,29,46

and just add 55,61,72 to sorted list.

comparisons: (3/2)*n since n = 6 and the comparisons = 9

b. m=1 and b[1] < a[1]

a = [2,3,4,5]
b = [1]

so all of b (just one value) is smaller than a.

comparisons = 1 = m. Once we compare 1 and 2, we can just add the rest of a to the sorted list so only 1 comparison needs to be done.
c. m=1 and a[n] < b[1]

eg: a = [3,4,5,6]
b = 10
comparisons = n because we still need to compare every value to the one large value in b before adding all of a to the sorted array.

For questions 5 – 6, compare the efficiency of searching an ordered sequential table of size n and searching an unordered table of the same size for the key target:

5.
a) If no record with key target is present

If no record with the key is present in an ordered table, we can terminate once the record in the file is larger than the target key. An avg of n/2 comparisons are necessary

If it's unordered, you still have to search the entire file and cannot stop until the end.

cost of ordered < cost of unordered

b) If one record with key target is present and only one is sought.

Sometimes we get lucky and the target is the first record, sometimes it's the last though so… Cost: **O(n/2)**, regardless of order. Once we find it, we stop and don't search the rest of the file

cost of ordered = cost of unordered

6.
a) If more than one record with key target is present and it is desired to find only the first one. I think the cost is O(n/2). The records could be bunched up at the beginning or bunched up at the end but the more Xs there are, the smaller the cost since the % that one of the Xs

I think regardless of order, the cost is the same.

b) If more than one record with key target is present and it is desired to find them all.
cost: **O(N) if the file is unordered.** Even if you find the target… you still have to search to the end of the file for the rest of them since the X could potentially be the last item and you don't know that it is not at the end.

However, order changes this. If hte file is ordered, you may be able to stop searching the rest of the file once X is found.. Say target = X

The file could look like
X,3,4,5,6
X,X,X,3,4,5
3,4,5,6,X,X
3,4,5,6,X,9

Once X is found and the next item is > than X, you can stop searching because you know that X won't be at teh end of the file. **So the cost is certainly less than O(n).**

I think the cost depends on how many Xs there are and how big n is, but I think the avg cost is O(n/2)

Cost of ordered < cost of unordered. Ordered records results in a more efficient search.