1. For this question we consider last activity to start instead of first activity to finish. It is still greedy choice algorithm because we are looking for the best choice at the moment. That is, it is making a local optimal choice which in turn becomes a global optimal choice. This is exactly similar to Theorem 16.1 but the only difference is we start from end rather than starting from the beginning. The proof for the same will be as follows.

Consider any nonempty subproblem S_k and let a_m be the activity in S_k with the last time time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.

Then let's arrange the activities in order of monotonically decreasing order of start time. Each activity $a_k = [s_k, f_k]$ where s_k = start time and $f_k = finish\ time\ of\ activity\ k.$

Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the last start time.
If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k.
If $a_j \neq a_m$, let the set $A_k^{'} = A_k - \{a_j\} \cup \{a_m\}$ be A_k but substituting a_m for a_j. Here we considered a new set where we have a_j belonging to A_k with $s_i > s_m$. Since $|A_k^{'}| = |A_k|$
We can say that new set is mutually compatible with the original set. $A_k^{'}$ is a maximum-size subset of mutually compatible activities of S_k.
Optimal Substructure says if it is an optimal solution or not. The resulting solution has optimal substructure because the new set is always mutual compatible with the original set, The optimal solution for the original set maps directly to the optimal solution in the new set and hence lead to global optimal structure.

2.
Coniser there are a set of activities S. There are m no of Halls. Using the Greedy-Activitiy-Selection algorithm, I will get a maximum set of compactable activities A1. I will assign the activity set A to one of the class hall. In the next iteration I will get a maximum set of compactible activities A2 from the Set of activities S - A1, This iteration continues until activity set S is empty. The algorithm for the same would look like.

```
Hall_Allocation :
 1.   Let S[1…..n] , H[1…..m], i=0
 2.   While S ≠ ∅ :
 3.       Sort(S)
 4.       A= Greedy-Acitvity-Selection(S)
 5.       H[i] = A
 6.       i++
 7.       S = S - A
 8.   return H
```

3.
Greedy choice property states that an optimal solution to a problem can be obtained by making local best choices at each step of the algorithm.

Proof by Contradiction.

Let there be a total of n items.
The set of items be denoted by items = $\{item_1, \ldots\ldots\ldots\ldots, item_n\}$Weight of the items is denoted by weights = $\{weight_1, \ldots\ldots\ldots\ldots, weight_n\}$
Values of the items are denoted by values = $\{value_1, \ldots\ldots\ldots\ldots, value_n\}$
Let the optimal solution be called O which includes the following items $\{O_1, \ldots\ldots\ldots\ldots, O_n\}$.
Let the item with the highest value/weight ratio be i and lets assume that this is not the part of the optimal solution O.

Let there be an item called x which is a part of the optimal solution, and we have

$$\frac{v_x}{w_x} < \frac{v_i}{w_i}$$

Now let remove a fraction f of item x and add a fraction of item i to the optimal solution.
Then we have new optimal solution O'. from our assumption the new $O' < O$
But we have $O' - O = f\frac{v_i}{w_i} - f\frac{v_x}{w_x} = f(\frac{v_i}{w_i} - \frac{v_x}{w_x}) > 0$ and hence we can say $O' > O$ and this contradicts our assumption that O is optimal without having the item i, so our assumption is wrong and we can say that fractional knapsack follows greedy choice property.

4. The Algorithm to calculate the pay off would be as follows.

```
Pay_Off(a,b):
  1.    Sort(a)
  2.    Sort(b)
  3.    for i = 1……length(a):
  4.        Pay = a[i]**b[i]
  5.    return Pay
```

This algorithm is greedy in nature we are sorting the a and b such that largest number has largest exponent value.
Suppose that we have two terms for the product a1<a2 and b1> b2. In this case the (a1 ^ b2) * (a2 ^ b1) is greater than (a1 ^ b1) * (a2 ^ b2) , where we can improve the payoff by swapping the terms a1 and a2. hence original payoff is not optimal but after sorting we get the optimal solution. Hence my algorithm is correct because it sorts the a and b and hence provide optimal solution.

5.

Let H = {H1,……..,Hn} be the houses sorted from east to west order. P = {P1,……,Pn} be the base station distance from east end point. The algorithm would be as follows.

```
Base_station_locations:
1.      H[1….n], P[1….n] cur = 0, i =0
2.      While H ≠ ∅ :
3.          cur = H[1] + 4
4.          P[i] = cur
5.          i++
6.          H =  H - A              // where A is all the houses covered by P[i]  from H
7.      return P
```

In this algorithm we are finding the left most house close to east end point in the set H. We will place our base station 4 miles to the right of left most house. We will remove all the houses that are reachable from the base station. We will insert a new base station based on the new set, find the left most house in the new set, place the new base station 4 miles to the right of left most house. This process is done until all the houses are covered. The complexity of the algorithm will be O(nlogn).