Vakhid Betrakhmadov
141044086

1)
Didn’t have time to watch the movie, but it’s probably very good!

2)
First, we construct sequence of jobs, where size(jobs) == size(RA). If we have more RA then jobs, then missing jobs in the sequence are completed with -1. Then we permutate this sequence of jobs, and end up having all permutations in hand. This operations is accomplished by recursive “findAllJobsDistributions” function. Recurrence relation for this function is as follows:

$$T(0) = 1$$
$$T(n) = n\,T(n-1)$$
$$\begin{aligned} T(n) &= n\,T(n-1) = n(n-1)T(n-2) = n(n-1)(n-2)T(n-3) \\ &= n(n-1)(n-2)(n-3)T(n-4) = n * \ldots * 3 * 2 * 1 * T(0) \\ &= n * \ldots * 3 * 2 * 1 * 1 = !n \end{aligned}$$

In the next step we just iterate through all permutations trying each one by one, which is also !n time.
So “findOptimalAssistanship” function yields 2*!n time complexity, and that is valid for all best, worst and average cases.
So A(n) = W(n) = B(n) = Θ(!n)

3)
First, if the cost of building a lab is less then repairing a road, then we simply build labs in all departments. That is result will be: $numberOfDepartments * labCost$
Otherwise, we have to build at least by one lab in all the parts of the university which can’t be connected together. If we assume these non-connectable parts of the university as connected graphs of one big unconnected graph, we can apply DFS to each connected part, and find all depth-first search trees (forest). Then in each depth-first search trees we build lab in the root of the tree, and repair all the roads in this tree. Notice that depth-first search tree doesn’t contain any back edges, so we will not have any unnecessary roads repaired.
This way result is: $numberOfRoots(in\ the\ forest) * labCost + numberOfEdges(in\ the\ forest) * roadCost$

In the worst case, we have to find all DFS trees. We know that DFS algorithm time complexity is Θ(|E|), where E is the number of edges in the graph. Our DFS algorithm returns a map of parents, where keys in this map imply vertices and values imply their parents in the DFS tree. If vertex value is -1, then it’s a root in its tree, otherwise it’s a child, and we know that we will have number of edges equals number of children.
So size (map of parents) = |V|, where V is number of vertices in the graph.
In our second step, after applying DFS, we iterate through our map of parents in order to find the final result. Time complexity of this operation is thus |V|.
So W(n) = Θ(|E| + |V|)

4)
Insertion sort:
1. 12, **34**, 54, 2, 3
2. 12, 34, 54, 2, 3

Shell sort:
Gap = 5/2 = 2
1. 12, 34, **54**, 2, 3

3. 12, 34, **54**, 2, 3
4. 12, 34, 54, 2, 3
5. 12, 34, 54, **2**, 3
6. 12, 34, 54, **54**, 3
7. 12, 34, **34**, 54, 3
8. 12, **12**, 34, 54, 3
9. **2**, 12, 34, 54, 3
10. 2, 12, 34, 54, **3**
11. 2, 12, 34, 54, **54**
12. 2, 12, 34, **34**, 54
13. 2, 12, **12**, 34, 54
14. 2, **3**, 12, 34, 54
15. 2, 3, 12, 34, 54

2. 12, 34, 54, 2, 3
3. 12, 34, 54, **2**, 3
4. 12, 34, 54, **34**, 3
5. 12, **2**, 54, 34, 3
6. 12, 2, 54, 34, **3**
7. 12, 2, 54, 34, **54**
8. 12, 2, **3**, 34, 54

Gap = 2/2 = 1

9. 12, **2,** 3, 34, 54
10. 12, **12**, 3, 34, 54
11. **2**, 12, 3, 34, 54
12. 2, 12, **3,** 34, 54
13. 2, 12, **12**, 34, 54
14. 2, **3**, 12, 34, 54
15. 2, 3, 12, **34**, 54
16. 2, 3, 12, 34, **54**
17. 2, 3, 12, 34, 54

The advantage of Shell sort is that, when a swap occurs in Shell sort, it moves the elements further up and much closer to their final positions, thus by the time the standard insertions sort is applied, the array becomes almost sorted and we know that insertion sort is very fast on nearly sorted arrays.
Insertion sort, unlike Shell sort on the other hand, moves items in the array only 1 position at a time, which results in worse performance on unsorted arrays in comparison to Shell sort.