CS5800: Algorithms Spring 2018
Assignment 4.1

Saptaparna Das

February 23, 2018

(a) Algorithm:

```
Let benifit is a 2D array holding benifit of each project for
all possible number of programmers

1. Set assignedProgrmrs, maxProgrmrsAssignedToAProg and projVal to 0
2. Create empty array numProgsAssigned of size = number of projects
3. Till assignedProgrmrs < number of programmers, repeat following steps:
        a. Set maxPossibleBenefitFrmAnyProject to integer min value
        b. Set maxPossibleProjIndex to -1
        c. Set i=0. For each project, do the following steps:
                i. set benifit to 0
                ii. If project doesn't have any programmer assigned
                        Set benifit to benifit produced by assigning single programmer i.e.
                        benefitArr[i][ numProgsAssigned[i] ]
                    Else
                        Set benifit to benfit produced by new programmer i.e.
                        benefitArr[i][ numProgsAssigned[i] ] - benefitArr[i][lastProgrmrIndx]

                iii. If benifit is more than maxPossibleBenefitFrmAnyProject
                        set benifit to maxPossibleBenefitFrmAnyProject
                        set i to maxPossibleProjIndex

        d. Increament numProgsAssigned[maxPossibleProjIndex] by 1
        e. Add maxPossibleBenefitFrmAnyProject to  projVal

4. Return projVal to print max benifit of all projects
   or numProgsAssigned to print actual project assignments
```

(b) Correctness: To prove the greedy algorithm, we have to prove following properties:
1. Optimal substructure: Let A_ij be the optimal solution or max benifit for projects i to j. Say, it includes a solution a_k which is benift from k^{th} project. So, A_{ij} can be written as A_{ik}+A_{kj}+a_k. Now A_{ij} should contain optimal solution of its subproblems. If its not true, then there exists a solution $A'_{ik} > A_{ik}$. So, if we add a_k to the solution (benifit of kth project, which is not there in A_{ik}) we get a new solution A'_{ik} +a_k+ $A_{kj} > A_{ik}$+A_{kj}+a_k. So, $A'_{ij} > A_{ij}$, which is contradiction to original solution.

So, the problem exhibits optimal substructure property i.e. its optimal solution contains optimal solution of subproblems.

2. Greedy choice: To have a greedy choice for this solution means to have as much benifit possible from aproject. If we make a greedy choice, we are left with one subproblem i.e. getting benifits from other prjects. For the first choice we don't need to consider other projects because we are always choosing the project with maximum benifit.

Now to prove greedy choice is optimal, let us assume a_m is the project with max benifit. Let A_k be the optimal solution where a_j is the project where first programmer is assigned. Now, if a_j != a_m, then we can have a new solution $A'_k = A_k - a_j + a_m$. Since a_m gives max benifit, we can write $a_m > a_j$. So, $A'_k > A_k$, which is a contradiction to original solution. Hence, $a_j = a_m$ i.e. the greedy choice is always part of the optimal solution.

Complexity analysis:
Here we are calculating benifit of each project for every programmer. So, for m projects and n programmers, it should

take O(mn) time. Time complexity = O(mn). The amount of space taken by the algorithm changes depending number of projects. numProgsAssigned is the array that holds project assignments and is of size number of projects. So for m projects, space complexity should O(m).