In [17]:
```
#Q3
```

In [30]:
```
# Expectation Calculation
# It is assuming that only the first non solved task will be attempted again. aft
def expectedReturnFirstNonSolvedTask(policy,attempts,first_task_nonsolved):
    if len(policy) > 0 and attempts <= 8 :  # Max number of attempts is 8
        # we take the next task and increase attempts
        task = policy[0]
        if  first_task_nonsolved == False:
            second_attempt_pass = task[1] + expectedReturnFirstNonSolvedTask(poli
            second_attempt_fail = 0 + expectedReturnFirstNonSolvedTask(policy[1:
            first_attempt_pass  =      task[2]     * ( task[1] +  expectedReturnF
            first_attempt_fail  =  (1-task[2])     * (    0    +  (task[2]/2)*se
            return first_attempt_pass + first_attempt_fail
        if first_task_nonsolved == True : #This means that once task has already
            first_attempt_pass  =      task[2]     * ( task[1] +  expectedReturnF
            first_attempt_fail  =  (1-task[2])     * (    0    +  expectedReturnF
            return first_attempt_pass + first_attempt_fail
    else:
        return 0
```

In [31]:
```
# Find expectations of Policy A and B
tasks = [(1, 12, 0.25), (2, 4, 0.4), (3, 10, 0.35), (4, 5, 0.6), (5, 7, 0.45), (
```

In [32]:
```
policyA = tasks #Policy A, sequential order
policyB = sorted(tasks, key=lambda task: task[2], reverse=True) # Policy B, order
print("Policy A: ",policyA)
print("Expected return of policy A: ",expectedReturnFirstNonSolvedTask(policyA,0

print("Policy B: ",policyB)
print("Expected return of policy B: ",expectedReturnFirstNonSolvedTask(policyB,0
```

```
Policy A:  [(1, 12, 0.25), (2, 4, 0.4), (3, 10, 0.35), (4, 5, 0.6), (5, 7, 0.4
5), (6, 3, 0.5), (7, 50, 0.15)]
Expected return of policy A:  24.6665459375
Policy B:  [(4, 5, 0.6), (6, 3, 0.5), (5, 7, 0.45), (2, 4, 0.4), (3, 10, 0.35),
(1, 12, 0.25), (7, 50, 0.15)]
Expected return of policy B:  24.497423437500004
```

In [33]:
```
#Q4
```

In [34]:
```
# Randomly shuffle the order of tasks to generate all policies. Check the expecte
# Simulation of the iteration algorithms . Generating a shuffle of all policies c
import itertools
maxer = 0
for policy in list(itertools.permutations(tasks)) :
    #print("Current Policy :",policy)
    er = expectedReturnFirstNonSolvedTask(policy,0,False)
    if er > maxer :
        maxer = er
        policyC = policy
```

In [35]:
```
print("Improved policy C: ",policyC)
print("Expected Return of policy C: ",maxer)
```

```
Improved policy C:  ((7, 50, 0.15), (3, 10, 0.35), (5, 7, 0.45), (1, 12, 0.25),
(4, 5, 0.6), (2, 4, 0.4), (6, 3, 0.5))
Expected Return of policy C:  26.6859575625
```

In [ ]:
```

```