$\frac{9}{9}$ $\frac{25}{50}$

Ex 8-4. Read ✓✓

Ex 8-5.

If we take the complement ("¬") of this Tautology then it evaluates that this compliment is this;

$\neg(\forall x;\ \phi = 1) \Rightarrow \quad \neg\phi \neq 0 \Rightarrow \neg\phi = 1 \Rightarrow \phi = 0$
for one combinat

Then we can evaluate $\phi = 0$ in $O(n)$ since we check one combination. This is in poly. time so ¬Tautology is NP thus Tautology is co-NP.

$\frac{5}{8}$

Ex 8-6.

In proof, function f maps every string to circuit and we go iteratively with this function. And the algoritm A that is used by algoritm F that is also polynomial since it feeds the f algoritm, these steps are done in polynomial each with input strings with some length thus memory used is poly. size and it can increase in polynomial factor.

We can also see that since A runs in poly., each state is mapped to another state in polynomial time for inputs and writes outputs one-by-one then the memory usage is contigous.

$\frac{3}{8}$

3/5

P 8-2.

a) We can think this way, say we have a two set problem which generates two answers and evaluates them, this would take $O(n)$ which is polynomial, so it is NP problem.

Let ϕ be $\rightarrow$ "$(X_1 \wedge X_2)$ - - - - - -" and if we add "$\vee X_n$" to this and if there are two sets such as "1001 - - 0" and "1001 - - 1" we have two solutions for x_n this problem, with same logic if we divide ϕ to two "$\vee$" parts and do this to smallest subset and multiply these numbers, we can find answers in poly. time.

b)
```
algo(y, U)
for each u in U
    if u<k or (isNeighbour(u,u-1)) or (isNeighbour(u,u+1))
        return false
    else u++
isNeighbour(x, y):
    for each neighbour of x:
        if (neighbour == y)
            return true
    return false.
```

4/10

We can also say that if there is poly. time algorithm that output our desired list U, then another algorithm that checks whether previously found solution is optimal and makes sure all $p(u) \geq k$ for vertices should also run in polynomial time in worst case. Then if such algorithm is existing, we can follow that this algorithm is P and since this problem Donut is NP-hard and we solved in polynomial time we see P=NP for this problem.

c) For our algorithm to be decision algo, we need to determine if two things hold true for problem.
1- for each process a_i:
  $sum(t_0, t_1, t_2 \cdots t_{i-1}) \leq d_i$ //for all executed before deadline
and
for each group that satisfy property (1) is our solution the max$(p_1, p_2, p_3 \cdots p_i)$ for combinations in this problem. If property 1 is not satisfied or our solution is not most profitable combination that is feasible then it is not correct.

Thus we changed this problem to a decision problem as requested.

p:	4	6	2	8
d:	3,	5,	7,	6
t:	2	4	5	7

For algoritm,

~~$P_{i+1} = max(P_i)$~~

```
algo(){
sorted= sort(p0,p1,p2 ... pi)
  bool[n]
  for(i=0 to n){
      for(j=min(t-1, sorted(i).d -1) to j=0){
          if(bool(j)== 0){
              bool(j)=1
              add to jobs(i) ; t -= t + tj
          }
      }
  }
```

$\frac{4}{10}$

This sorts the prices, if there is time left and if deadline has not come yet, it adds this job to scheduled jobs, deducts it's length from the time and moves for next available time. Hopefully produces optimal solution. Maybe we could implement this with linear programming by maximize $\sum x_p$ s.t. each $\sum_0 t_i \leq d_i$.