Basic Compiler Optimizations:

Question 1:

The optimizations are listed below:

1. Common subexpression elimination
2. Copy propagation
3. Invariant code motion
4. Strength reduction
5. Test Elision and Induction Variable Elimination and
6. Constant Propagation and Dead Code Elimination

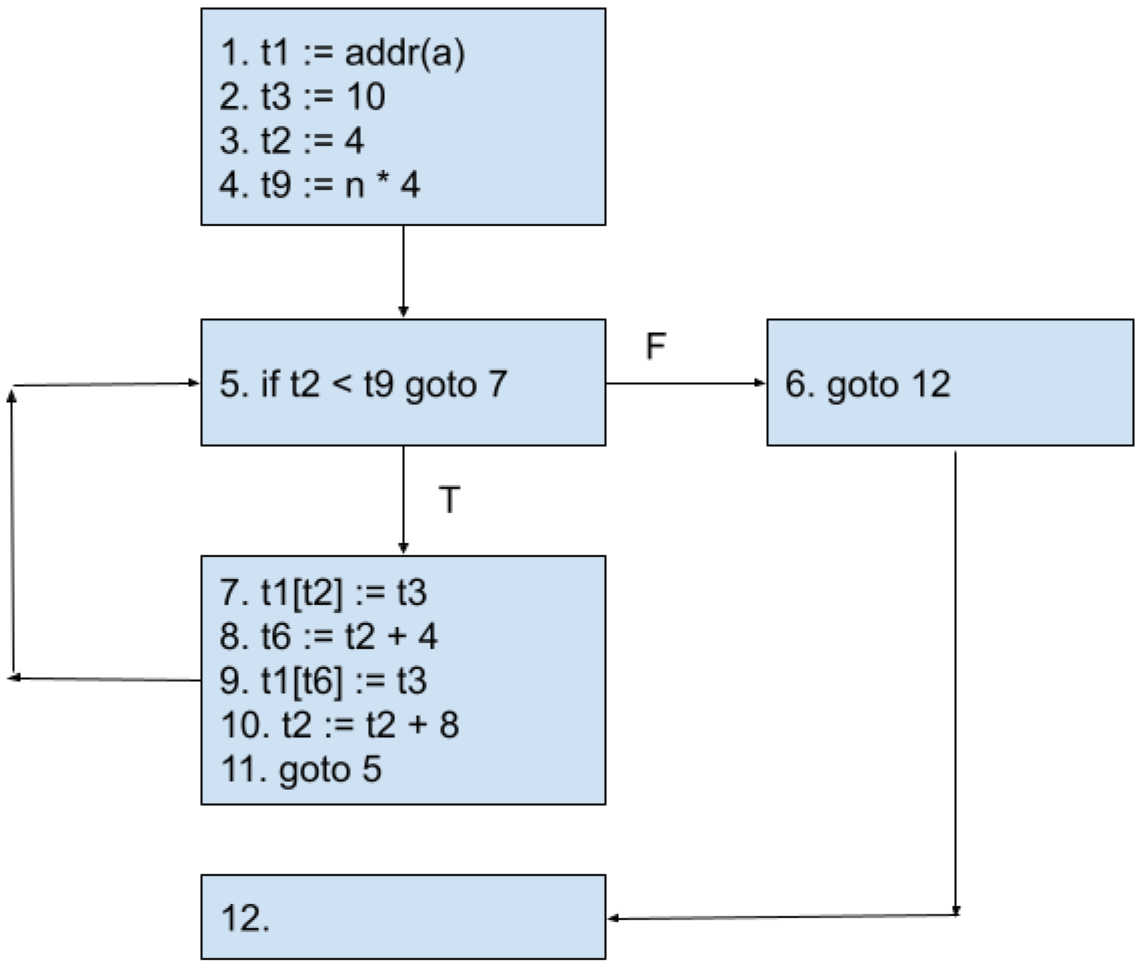
Question 2:

The optimized code after applying the optimizations listed above.

```
1. t1 := addr(a)
2. t3 := 10
3. t2 := 4
4. t9 := n * 4
5. if t2 < t9 goto 7
6. goto 12
7. t1[t2] := t3
8. t6 := t2 + 4
9. t1[t6] := t3
10. t2 := t2 + 8
11. goto 5
12.
```

Question 3:

The CFG of the optimized code is attached below.



Dataflow Frameworks:

Question 1:

a) The property set is the lattice of all subsets of the variables in the problem. As it is a may problem, ≤ is the subset operation. 0 is the empty {} set and 1 is the universal set.
Join is the set union operator.

b) Transfer functions: in(j) = gen(j) ∪ (out(j) - kill(j))
   out(j) = {∪ in(i), i is successor of j}

   gen(j) = {all the variables in the expression that means the variables in the both left and right side}
   kill(j) = {} because used or defined actually does not kill any variables.
   For example, for the expression, j: x=y+z, gen(j)={x, y, z}, kill(j)={}

c) The initial value of the extremal node is {}. The initial values of the other nodes are {} also.

Question 2: A. True

Question 3: B. False

Question 4: A. MFP = MOP

Question 5:

No, the answer would not be safe to use for program transformation because some information is being lost. For example, in Reach problem, some definition (x,k) does not reach a node when in fact it reaches the node.

RTA, XTA, PTA and Context Sensitivity:

Question 1:

For l.evaluate():
{ConstExp.evaluate(), VarExp.evaluate(), OrExp.evaluate(), AndExp.evaluate()}

For r.evaluate():
{ConstExp.evaluate(), VarExp.evaluate(), OrExp.evaluate(), AndExp.evaluate()}

Question 2:

For l.evaluate():
{ConstExp.evaluate(), VarExp.evaluate(), OrExp.evaluate(), AndExp.evaluate()}

For r.evaluate():
{ConstExp.evaluate(), VarExp.evaluate(), OrExp.evaluate(), AndExp.evaluate()}

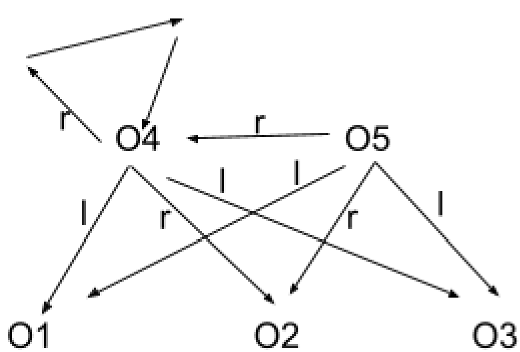
Question 3:

For l.evaluate():
{ConstExp.evaluate()}

For r.evaluate():
{VarExp.evaluate(), OrExp.evaluate()}

Question 4:

The points-to graph is given below.



Question 5: o3, o1

Question 6: o3

Question 7: o3

Abstract Interpretation:

Question 1:

The constants abstraction defined in the class is:

1) $\alpha(c) = \perp$ if $c = \{\}$
2) $\alpha(c) = \underline{n}$ if $c = \{n\}$
3) $\alpha(c) = T$ otherwise
4) $\gamma(T) = Z$
5) $\gamma(\underline{n}) = \{n\}$
6) $\gamma(\perp) = \{\}$

1. We know that, α and γ form a Galois connection, if
   for every $a \in A$ and every $c \in C$,
   $c \subseteq \gamma(a)$ iff $\alpha(c) \leq a$

   Now, for these constants abstraction we can write,
   If $c = \{\}$, then for every a, $\{\} \subseteq \gamma(a)$ and $\alpha(\{\}) \leq a$.
   If $c = \{n\}$, then $c \subseteq \gamma(a)$ or $\alpha(c) \leq a$ for $a = \underline{n}$ or $a = T$.
   If c is any other set, then $c \subseteq \gamma(a)$ or $\alpha(c) \leq a$ for $a = T$

   So, we can say α and γ form a Galois connection.

2. Now we can show,
   $\alpha(\gamma(T)) = \alpha(Z) = T$
   $\alpha(\gamma(\underline{n})) = \alpha(\{n\}) = \underline{n}$
   $\alpha(\gamma(\perp)) = \alpha(\{\}) = \perp$

   That proves that $\alpha(\gamma(a)) = a$ for all a.

So, we can say that the constants abstraction is a Galois insertion.

Question 2: A. True

Question 3: B. False

Types:

Question 1:

Define factorial = fix ƛf. ƛx. if (iszero x) 1 (times x (f (pred x)))
Where,
times = fix ƛf.ƛx.ƛy. if (iszero x) 0 (plus y (f (pred x) y))
plus = fix ƛf. ƛx.ƛy. if (iszero x) y (f (pred x) (succ y))

Question 2:

let factorial n = if n == 0 then 1 else n* factorial (n-1)

Question 3:

a) [Int / t_0, Int -> Int / t_1]
b) [t_0 / t_1, t_0 / t_2, t_3 -> t_4 / t_0]
c) [Int / t_0, Int / t_1]
d) No principal unifier exists.

Question 4:

(t1 -> t2 -> t3) -> (t1 -> t2) -> t1 -> t3

Question 5:

A. The type is, int -> int

Question 6:

B. Because z comes from outer scope and may not be polymorphic.

Question 7:

a) YES.

b) ((((t_0 -> t_0, t_0 -> t_0), (t_0 -> t_0, t_0 -> t_0)), ((t_0 -> t_0, t_0 -> t_0), (t_0 -> t_0, t_0 -> t_0))), (((t_0 -> t_0, t_0 -> t_0), (t_0 -> t_0, t_0 -> t_0)), ((t_0 -> t_0, t_0 -> t_0), (t_0 -> t_0, t_0 -> t_0))))

c) It will increase the complexity of the program exponentially. I ran the haskell representation of the code in haskell and it got kind of stuck. It should return 2^{10} pairs.