Question 1
Q1.a) F = !(A*B), ou seja, é uma NAND2
Q1.b) F = !(A*B*C), ou seja, é uma NAND3
Q1.c) F = !(A+B), ou seja, é uma NOR2
Q1.d) F = !((A+B)*C*D)
Q1.e) F = !((A+B)*C + D)
As tabelas-verdade podem ser conferidas com a ferramenta Karma, disponível no laboratório LogiCS da UFRGS (procurem no Google por esses termos, baixem a versão DEsktop, e nela há um arquivo .jar para ser executado).

Question 2
Para essa questão, implementaremos sempre a equação dada, com as adaptações estritamente necessárias (para negações), mas sem nenhuma outra simplificação, e usaremos inversores para negar todos os sinais, assumindo que eles estão disponíveis como entrada sempre apenas não invertidos.

Q2.a)
F = !((C*!D)+(!A*B*D)+(!A*!B*!C)+(A*!B*!D))

Função tem uma soma lógica negada, e pode ser implementada por por uma porta complexa com caminhos alternativos levando a GND.
Esquemático da rede pull-down tem quatro caminhos alternativos, cada um com os transistores NMOS com gates ligados aos literais dos termos produto em série. Rede pull-up é complementar. Cada rede tem então 11 transistores. Total são 22 transistores mais quatros inversores, resultando em 30 transistores.

Com portas não complexas, implementa-se como uma soma de produtos feita somente com NANDs, com as negações necessárias nas entradas e na saída, que é negada. Cada NAND do termo produto tem 2 vezes o número de entradas em transistores, e a NAND correspondente à soma tem 4 entradas. Total fica 1 porta de 4 xtores, 3 portas de 6 xtores, uma porta de 8 xtores e cinco inversores, resultando em 40 transistores.

Q2.b)
F= !C*!D + !A*C + !A*!B*!D + A*B*!C
Para ser implementada como uma porta complexa, a rede pull-down deve levará a saída a GND, e então primeiro usaremos o teorema de DeMorgan para gerar uma função equivalente com uma negação na saída (topo), que fica:
!!F= !((C+D)*(A+!C)*(A+B+D)*(!A+!B+C))
Assim, vemos que essa função é um produtório, ou seja, tem uma sequência em série para levar a saída até GND, e cada estágio desse caminho tem 2 ou três alternativas, dos termos-soma.
A rede de pull-up é sempre complementar, e note que poderíamos

ter analisado ela primeiro, com a equação original, já considerando que os transistores PMOS ativam com 0, portanto implementam os literais !C, !D, etc diretamente. O circuitos resultante é o mesmo, e tem 2 vezes o número de literais da expressão como transistores, ou seja, 20 transistores, mais os 4 inversores necessários, totalizando 28 transistores.
Para implementar com portas CMOS básicas, não complexas, é uma soma de produtos normal, com dois termos-produto de duas variáveis e dois de três variáveis, totalizando 2 portas de 2 entradas, 2 portas de 3 entradas, uma porta de 4 entradas para a soma, e mais 4 inversores, resultando em 36 transistores.

Q2.c)
F = !(E + (ABC) + D)
Essa função pode ser implementada facilmente como uma porta complexa pois tem apenas uma negação na saída, e a sua rede pull-down reflete a estrutura paralelo/série para as operações de or/and lógicos, respectivamente, e terá 10 transistores no total, pois não necessita nenhum inversor.
Já para a implementação com portas simples, precisamos gerar mais uma dupla inversão interna, usando um par NAND-INV para implementar a subfunção AND. No total, teremos duas portas de 3 entradas e mais um inversor, totalizando 14 transistores.
F = !(E + !!(ABC) + D)

Q2.d)
F = (A+D)*C*B
Essa função não tem nenhuma negação, e precisamos usar o teorema de DeMorgan para processar a função equivalente not not F, que irá gerar a negação necessária na saída para implementar usando os transistores NMOS na rede de pull-down, o que terá o efeito colateral de requerer todas as entradas negadas.
!!F = !((!A*!D)+!C+!B)
No total, precisaremos de 8 transistores na porta e mais 4 inversores, resultando em 16 transistores.
Já na implementação com portas simples, precisamos de apenas dois inversores para gerar as funções AND e OR a partir de NAND e NOR, e teremos uma implementação com apenas 14 transistores.

Question 3)
Y1 = !A + !B
Y2 = A*B + C*D + E*F
Y3 = !A*B + A*!B

Question 4)
Q4.a) F = !(A*B) já é uma NAND2
Q4.b) F = !(A*B*C) já é uma NAND3

Q4.c) F = !(A+B) é uma NOR2, F= !!!(A+B) = !(!(!A*!B)), ou seja, uma NAND com entradas e saída invertidas.
Q4.d) F = !((A+B)*C*D) = (!A*!B)+(!C)+(!D) em soma de produtos, e sabe-se que uma soma de produtos pode ser implementada por dois níveis de NAND, ou seja, uma NAND para cada produto (invertido) e uma NAND para a função soma na saída. Os produtos de uma variável devem ser invertidos, e nessa expressão, (!C) e (!D) já estão invertidos, então, além de duas NANDs é preciso apenas dois inversores para gerar !A e !B.
Q4.e) F = !((A+B)*C + D) é equivalente a (!A*!B*!C)+(!C*!D), minimizando por mapa de Karnaugh. Assim, pode ser implementada por 2 NANDs, sendo uma de 3 entradas, e mais 5 inversores.

Question 5)
Implementar uma XOR apenas com NOR e INV. Para implementar uma função com NOR, ela pode ser representada como um produto de somas, considerando os zeros da função. A função XOR é (!A*B)+(A*!B), em produto de somas fica igual a (A+B)*(!A+!B), e essa estrutura pode ser implementada com 3 NORs e dois inversores assim: !(!(A+B)+!(!A+!B))

Question 6)
A XOR com NANDS pode ser implementada a partir da expressão de soma de produtos (A*!B)+(!A*B), implementada com 3 NANDs para os dois produtos e a soma final, e mais duas NAND2 com as entradas unidas para fazer as inversões e gerar !A e !B.

Question 7)
Há duas formas de representar um circuito qualquer apenas com portas NAND. A forma mais direta e ineficiente é substituir cada porta por uma espressão (ou subcircuito) equivalente construído com NANDs. Assim, usaríamos INV(NAND) para fazer AND, NAND(INV,INV,...) para fazer OR, e NANDs com entradas iguais para fazer os invs. A outra forma é tomar a expressão, que nesse exercício é F=((!A*B)+(B*C))*(C+!D)*E, minimizar para soma de produtos com uma ferramenta que use um algoritmo como QuineMcCluskey (por exemplo, Karma, que nesse caso minimiza para F+(B*C*E)+(!A*B*!D*E)), e implementar essa soma de produtos com dois níveis de NANDs, mais NANDs para inversões, e tomando o cuidado de inverter as entradas que vão direto ao segundo nível, que são 'produtos' de apenas um literal.

Questão 8)
A simplificação pode ser feita apenas com os teoremas básicos:
F= (A+B)*(A+!B) = A*(B+!B) = A*1 = A
S= (A*B)+(!B) = (A+!B)*(B+!B) = (A+!B)*(1) = A+!B
H= (!A*B*C)+(A*B*C) = (!A*A)+(B*C) = 0+(B*C) = B*C

Questão 9) semelhante a 7, mas usando a equivalência com NORS: OR = INV(NOR) e AND = NOR(INV,...) ou minimizando para produto de somas.

Questão 10)
F1 = !A*B*C + !A*!B*C + !A*B*!C = (!A*B)+(!A*C), pode ser implementada com 3 NAND2 e 1 INV, = 14 transistores.
F2 = A*B*C*!D + A*!B*C*!D + A*B*C*D + A*B*C*!D = (A*B*C)+(A*C*!D), usando 2 NAND3, 1 NAND2, 1 INV, 18 transistores
F3 = !A*B + A*!B – implementada com 3 NAND2 e dois INVs fica com 16 xtores, e a XOR2 usa apenas 12.