Exercice 1 :

1/

$c_1 = \frac{1}{5}$ $n_0=10$ $\frac{1}{2}n^2-3n\geq\frac{1}{2}-3\frac{n^2}{10}=\frac{1}{5}n^2$ 3n donne : $\frac{3n^2}{10}$ car $n=\frac{n^2}{n_0}$ et $n_0=10$

$c_2 = \frac{1}{2}$ $\forall n\in\mathbb{N}\frac{1}{2}n^2-3n\leq\frac{1}{2n^2}\leq n^2$

2/

Soit $n_0\in\mathbb{N}, c_{1,} c_2>0$: on cherche $>n_0$ tq : $5n^3<c_{1n}^4$ => $5<c_{1n}$ => $\frac{5}{c_1}<n$ donc $n>\frac{5}{c_1}$ convient

3/

$P(n)\in\Theta(Q(n))$ <=> $deg(P)=deg(Q)$

$|P(n)|\in\Theta(n^{deg(P)})$ $k=deg(P)$

$P(n)=\sum_{i=0}^{k} a_i n^i$

Par récurrence : si $k=0, P(n)=a_0\in\Theta(1)$

Sinon on a $P(n)=a_k n^k+Q(n) avec Q(n)=\sum_{i=0}^{k-1} a_i n^i$

Par hypothèse de récurrence : $P(n)=a_k n_k-1+\Theta(n^{deg(Q)})$

$P(n)\leq a_k n^k+c_2 n^{k-1}$ $P(n)\geq a_k n^k-c_2 n$

Si $n>2\frac{c_2}{|a_k|}$ $P(n)\geq a_k n^k+\frac{c_{2n}^k}{2\frac{c_2}{|a_k|}}$ $P(n)\leq a_k n^k-\frac{c_{2n}^k}{2\frac{c_2}{|a_k|}}$

$\frac{|k|}{2}n^k\leq|P(n)|\leq3\frac{|a_k|}{2}n^k$

4/

Oui car c'est deux fois le premier

Non car on augmente la puissance ça serait comme n et n^2

5/

n! est plus grand car 2^n consiste à multiplier n 2 alors que n! multiplie les n premiers entiers.

n^n est plus grand car il multiplie n fois n et n! que les n premiers entiers

7/

8/

La même chose que max avec moins au lieu de plus

Exercice 2 :

1/ f(n) + g(n)

2/ f(n) puissance g(n)

3/

Exercice 3 :

1/ n^3

2/ Θ n^2 $\frac{n(n+1)}{2}$

3/ Θ n^2 $4\frac{\frac{n}{2}n(n+1)}{2}$

Exercice 4 :

La complexité de compte est de n car on doit parcourir toutes les cases de T.

```
def compte(i, T) :
        compteur = 0
        for(j in T) :
                if(i == j) :
                        compteur += 1
        return compteur

def algo2(T, m) :
        S = [0] * (m+1)
        for i in T
                S[i] += 1
        return S
```

Complexité : taille du tableau

Exercice 5 :

1/

```
def max(T):
    max = T[0]
    for i in T:
        if T[i] > max : max = T[i]
    return max
```

complexité temps : linéaire
complexité espace : 1
2/

```
def inf(T, x):
    n = 0
    for i in T:
        if x > T[i]:
            n +=1
    return n
```

complexité temps : linéaire
complexité espace : 1

3/

```
def trie(T):
    for i in range(1, len(T)):
        if T[i] < T[i-1] :
            return False
    return True
```

complexité : linéaire
complexité espace : 0
4/

```
def circulaire2(T):
    cesure = 0
    for i in range(len(T)):
        if T[i] > T[(i+1)%len(T)]:
            cesure +=1
        return (cesure <= 1)
```

complexité temps : linéaire
complexité espace : 1