ANÀLISI I DISSENY D'ALGORISMES (Codi: 102783)
Primer Parcial: 31/10/2018
Nom i Cognoms: .....NIU: .....

IMPORTANT: Responeu en l'espai disponible a sota de cada pregunta

1. (5 punts) Són certes o falses, les següents afirmacions? Cada resposta correcta val 0.25 punts i cada resposta incorrecta resta 0.25.

    1) Un algorisme és robust quan compleix amb la seva especificació.
    2) Les precondicions d'una funció s'utilitzen per detectar quan el codi que la crida no compleix amb les condicions de l'especificació de la funció.
    3) La invariant d'una classe la compleixen els objectes de la classe mentre s'executi codi dels mètodes de la classe.
    4) Les condicions especificades en un assert s'executen en la versió de depuració del programa en C++.
    5) Una funció amb recursivitat final només pot tenir una crida recursiva.
    6) El mètode general per passar una funció recursiva a iterativa genera una funció que utilitza una pila, un while i un switch.
    7) L'optimització de recursivitat en cua converteix el codi d'una funció recursiva en un bucle.
    8) Una funció f(n) amb recursivitat lineal fa n crides recursives per cada crida a f excepte en el cas trivial de sortida de la recursivitat.
    9) Els algorismes de backtracking fan un cerca en profunditat prioritària per l'arbre de cerca o espai de solucions.
    10) Els algorismes de backtracking són els més apropiats per resoldre problemes d'optimització que compleixen que la unió de les solucions òptimes dels seus subproblemes forma una solució òptima del problema global.
    11) Per obtenir un bon rendiment utilitzant backtracking, s'han de comprovar les condicions de la solució quan abans millor.
    12) La complexitat dels algorismes de backtracking és logarítmica.
    13) *Greedy* pren decisions sense tenir en compte les decisions prèvies.
    14) El punt crític perquè *Greedy* doni la millor solució possible, dins de les seves limitacions, és el criteri de selecció.
    15) Quan apliquem Greedy expandim tot l'arbre de decisió.
    16) Sempre que tinguem un problema d'optimització Greedy ens donarà una solució òptima.
    17) És millor conèixer θ*(f(n))* que *O(f(n))*.
    18) Si un algorisme té millor complexitat que un altre, això serà cert per a qualsevol volum de dades d'entrada.
    19) El volum de dades d'entrada és un dels paràmetres d'entrada de la funció.
    20) Si un algorisme és θ(f(n)) llavors és *O*(f(n))

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
V		x		x	x	x	x		x		x			x			x		x	x
F	x		x					x		x		x	x		x	x		x		

2. (1 punt) Afegeix al codi de la classe MyArray les assercions que assegurin el correcte funcionament dels constructors, destructor i mètodes de la classe.

```
class CElement {...};
    class MyArray {
        int m_Size;
        CElement *m_pArray;
        MyArray() {
            m_Size=0;
            m_pArray=NULL;
        }
        MyArray(int sz) {
            assert(sz>0);
            m_Size=sz;
            m_pArray=new CElement[m_Size];
        }
        ~MyArray() {
            assert(m_pArray);
            delete [] m_pArray;
        }
        void Set(int index,CElement e) {
            assert(m_pArray);
            assert(index>=0 && index<m_Size);
            m_pArray[index]=e;
        }
        CElement Get(int index) {
            assert(m_pArray);
            assert(index>=0 && index<m_Size);
            return m_pArray[index];
        }
        ...
    };
```

3. (1 punt) Ordena de més a menys eficients les següents complexitats:
$O(n)$, $O(2^n \log_2 n)$,$O(n^n)$, $O(3^n)$, $O(n^{10000})$, $O(n!)$

$$O(n) \subset O(n^{10000}) \subset O(2^n \log_2 n) \subset O(3^n) \subset O(n!) \subset O(n^n)$$

4. (1.5 punts) Passa la funció f escrita en C a una funció iterativa escrita en C que faci el mateix.

```
int f(int x, int y)
{
    if (x==0) return y;
    else if (y==0) return x;
    else if (x>y) return x*f(x/2,y);
    else return y*f(x,y/2);
}
```

SOLUCIÓ:

```
Fun f(x,y)=>
{
  If (x==0) y;
  Else if (y==0) x;
  Else if (x>y) {x*f(x div 2,y); }
  else { y*f(x,y div 2); }
}
```

```
Fun f2(x,y)=>
{
  var r=1;
  while (x!=0 && y!=0) {
    if (x>y) { r=r*x; x=x div 2; }
    else { r=r*y; y=y div 2; }
  }
  r
}
```

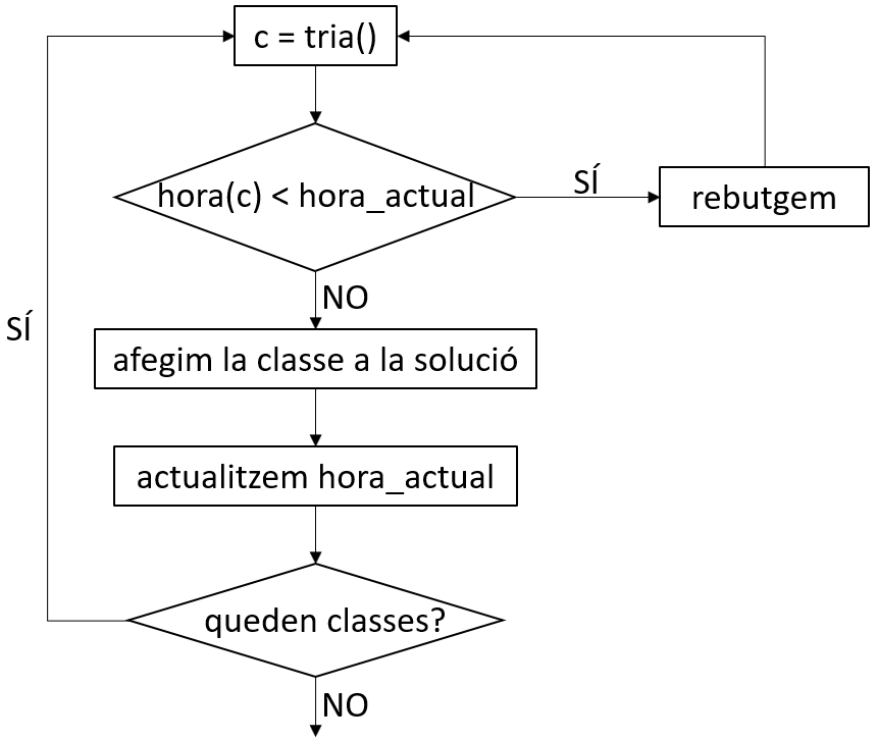
5. (1.5 punts) A l'escola s'han fet un embolic amb els horaris i algunes assignatures es solapen. Per poder ajudar als alumnes que volen seguir assistint a la majoria de les classes ens han demanat que fem un programa per ajudar-los a triar a quines assignatures han d'assistir tenint en compte que no hi ha una assignatura més important que una altra, no poden entrar ni sortir de classe quan la classe està en marxa i saben a quina hora comencen, però no saben si s'allargarà més de l'hora prevista.
    a. Dissenya un algorisme Greedy per tal que doni la solució òptima deixant clars tots els passos que faries.
    b. Si féssim backtracking milloraríem la solució? Justifica la resposta.

SOLUCIÓ:

a. Considerem que podem conèixer l'hora actual:

   Candidats: classes a assistir
   Funció de selecció: la que comença a l'hora més propera a l'hora actual
   Completable? Si la classe que trio comença a l'hora actual o més tard
   Funció objectiu: anar al màxim de classes possible
   Quan serà solució? Quan no ens quedin classes per triar

   Diagrama: c = tria() -> tria la classe que comença més d'hora



b. Podríem millorar si sapiguéssim l'hora de finalització. En el cas que hem aplicat greedy no tenim per què obtenir la solució òptima, ja que anem fent segons ens trobem i en aquest cas podria haver-hi una solució alternativa que permetés anar a més classes. Però per altra banda, si no sabem l'hora de finalització no podem aplicar backtracking.