NIU: 1

EXAMEN DE TEORIA DEL PRIMER PARCIAL TDA 9-11-2022

Nom:

Cognoms:

Respostes al Test											
1	a	b	**c**	d	e	**9**	a	b	c	**d**	e
2	a	**b**	c	d	e	**10**	**a**	b	c	d	e
3	a	b	**c**	d	e	**11**	a	b	**c**	d	e
4	**a**	b	c	d	e	**12**	a	b	c	**d**	e
5	**a**	b	c	d	e	**13**	a	b	c	d	**e**
6	a	b	**c**	d	e	**14**	a	**b**	c	d	e
7	a	b	**c**	d	e	**15**	a	**b**	c	d	e
8	a	b	**c**	d	e						

Nota: Las respostes equivocades resten ¼ del valor de la pregunta.

ESPECIFICACIÓ FORMAL

1.- Quin avantatge dona afegir precondicions en forma d'assercions en un codi?

a) Ens assegura que els resultats de las funciones son correctes.
b) Ens assegura que els bucles funcionen correctament.
c) Ens assegura que els paràmetres de les crides a funcions son correctes.
d) Ens assegura que els objectes de les classes sempre estaran en estats correctes.
e) Cap de les anteriors.

2.- Com podem detectar el mal us d'un programa per part de l'usuari?

a) Amb assercions.
b) Amb tractament estructurat d'excepcions.
c) Amb assercions a la versió final de programa (release).
d) b) i c).
e) Cap de les anteriors.

COMPLEXITAT ALGORÍSMICA

3.-Si tinc dos algorismes que tenen complexitat O(f(n)) i Ω(f(n))...

a) Els dos algorismes poden ser igual de rapits.
b) Un pot ser 1000 vegades més rapit que l'altre.
c) a) i b).
d) Un pot ser n^2 vegades més rapit que l'altre.
e) Cap de les anteriors.

4.- Quina complexitat té la funció f?

```
int f(int n) {
  if (n<1) return 1; else return f(n/2)*n;
}
```

a) O(log n)
b) O(n)
c) O(n*log n)
d) O(n^2)
e) Cap de les anteriors.

ALGORISMES GREEDY

5.- Un algorisme greedy es pot aplicar sempre que:

a) el conjunt de solucions òptimes locals portin a la solució òptima global
b) la prioritat és trobar la millor solució
c) el problema es pugui subdividir en problemes més petits
d) hi hagi una única solució
e) cap de les anteriors

6.- Què caracteritza a un algorisme greedy?

a) El conjunt de candidats, la funció objectiu, i la funció de selecció
b) La funció de solució, la funció de selecció, la funció de factibilitat, i la funció d'optimització
c) El conjunt de candidats, la funció de solució, la funció de selecció, la funció de factibilitat, i la funció objectiu
d) La funció de solució, el conjunt de candidats, la funció de factibilitat, la funció de selecció, la funció objectiu, i la funció d'optimtització
e) cap de les anteriors

7.- En l'algorisme de Kruskal, com es pot representar també la complexitat O(e*v + e*log(e))

a) O(e*v + v*log(e))
b) O(e^2 + e*log(e))
c) O(v^2 + e*log(e))
d) O(e*v + e*log(v))
e) cap de les anteriors

RECURSIVITAT

8.- La complexitat d'un algorisme recursiu

a) és proporcional al quadrat de trucades a la funció
b) és proporcional a la profunditat de trucades a la funció
c) es pot determinar utilitzant polinomis
d) és proporcional al valor d'entrada de la funció
e) cap de les anteriors

9.- En quin cas es pot convertir una funció recursiva en una funció iterativa amb un sol bucle?

a) Quan la funció té recursivitat indirecta
b) Quan la funció té recursivitat directa
c) Quan la funció té recursivitat múltiple
d) Quan la funció té recursivitat final
e) cap de les anteriors

10.- Quin tipus de recursivitat té la funció

```
int f(int x)
{
      if(x<=0) return 1;
      else
      {
            n = 5;
            for(int i = 0; i<10; i+)
            {
                  n += f(x-i);
            }
            return n;
      }
}
```

a) Recursivitat directa, múltiple
b) Recursivitat indirecta, múltiple

c) Recursivitat directa, final
d) Recursivitat lineal, indirecta
e) cap de les anteriors

DIVIDEIX I VENCERÀS

11.- Un algorisme de Divideix i Venceràs:
a) divideix el problema en dos problemes més petits recursivament, soluciona cadascuna de les parts per separat, i agrega els resultats
b) divideix el problema en varios subproblemes, soluciona un d'ells, i retorna el resultat del subproblema
c) divideix a cada pas el problema en dos o més problemes més petits, soluciona cadascun dels subproblemes per separat, i agrega els resultats
d) divideix el problema en varios subproblemes fins a trobar un cas trivial, que retorna com a solució
e) cap de les anteriors

12.- La complexitat d'un algorisme de Divideix i Venceràs, segons el Teorema Mestre, depèn de:
a) n i k
b) b i l
c) log b, n, i k
d) b, l, i k
e) cap de les anteriors

BACKTRACKING

13.- Un algorisme de backtracking:
a) Retorna la millor solució
b) Retorna la primera solució que troba
c) Retorna si existeix una solució o no
d) Retorna totes les solucions possibles
e) cap de les anteriors

14.- Un algorisme de backtracking és recomanable quan:
a) el conjunt de solucions òptimes locals portin a la solució òptima global
b) la prioritat és trobar la millor solució
c) el problema es pugui subdividir en problemes més petits
d) hi hagi una única solució
e) cap de les anteriors

15.- La principal manera de reduir la complexitat d'algorismes de backtracking és:
a) Reduir el nombre d'operacions per node visitat
b) Forward Checking
c) Reduir l'espai de solucions
d) Backward Checking
e) cap de les anteriors

PROBLEMES:

En una empresa de màquines expenedores de menjar volen implementar un sistema de retorn de canvi de monedes quan es fa una compra, prioritzant retornar el menor número de monedes possibles. Us encarreguen la implementació del sistema, que té en consideració les monedes que li queden a la màquina, i que en cas de no poder tornar el canvi exacte amb cap combinació de monedes, cancel·li la compra i retorni l'import de la compra.
El sistema ha de rebre com a inputs el preu del producte, l'import que l'usuari ha introduït a la màquina, i la llista de monedes disponibles a la màquina; i ha de retornar un indicador de si la compra s'ha de cancel·lar, i el conjunt de monedes a retornar.
La llista de monedes disponibles a la màquina està ordenada de manera decreixent (Per exemple: [2, 0.5, 0.5, 0.2, 0.1, 0.1, 0.5]).

a) Implementa l'algorisme explicat en C++

```
pair<bool, vector<float>> tornarCanvi(float preu, float import, vector<float> monedesDisponibles){
	float canvi = import-canvi;
	vector<float> conjuntCanvi;
	if(canvi==0){
		return pair<true, conjuntCanvi>;
	}
	return canviBacktracking(canvi, monedesDisponibles, conjuntCanvi, 0)
}
pair<bool, vector<float>> canviBacktracking(float canvi, vector<float> monedesDisponibles, vector<float> conjuntCanvi, float canviAcumulat){
	if(canviAcumulat==canvi){
		return pair<true, conjuntCanvi>;
	}
	else if(canviAcumulat > canvi){
		return pair<false, conjuntCanvi>;
	}
	while(monedesDisponibles.size()>0){
		float moneda = monedesDisponibles[0];
		monedesDisponibles.erase(0);
		conjuntCanvi.push_back(moneda);
		canviAcumulat += moneda;
		pair<bool, vector<float>> result = canviBacktracking(canvi, monedesDisponibles, conjuntCanvi, canviAcumulat)
		if(result.first){
			return result;
		}
		canviAcumulat -= moneda;
		conjuntCanvi.pop_back()
	}
	return pair<false, conjuntCanvi>;
}
```

b) Quina és la complexitat de l'algorisme implementat, en el pitjor dels casos, en funció del número de monedes disponibles a la màquina?

O(n!)