# Teoria

1. Dato il seguente array bidimensionale:

double M[3][4];

Scrivi un'espressione che, usando l'aritmetica dei puntatori, acceda all'elemento M[1][2].

2. Data la seguente funzione ricorsiva:

int g(int n) {

if (n <= 2) return 1;

return g(n-1) + g(n-3);

}

• Qual è il parametro su cui viene fatta la ricorsione?

• Qual è la misura di complessità del problema?

• Spiegare perché questa misura decresce ad ogni chiamata ricorsiva.

• Determinare il fattore minimo di decrescita della misura di complessità ad ogni chiamata ricorsiva.

3. Considerata la seguente struttura:

struct coordinate {

unsigned int x: 10;

unsigned int y: 10;

unsigned int z: 12;

};

Quanto vale 2\*sizeof(struct coordinate)/3?

4. Assumendo che un int occupi 4 byte, cosa stampa il seguente codice?

int mat[2][3] = {{1,2,3}, {4,5,6}};

printf("%d", \*(\*(mat+1)+1));

5. Data la funzione:

int S(int n) {

if (n <= 1) return 1;

return S(n-1) + n;

}

• Calcolare S(5)

• Qual è la complessità temporale di questa funzione in termini di notazione O-grande?

6) Cosa stamperanno le seguenti istruzioni? Spiegare perché.

printf("%d\n", \*\*\*r);

printf("%p\n", \*\*r);

printf("%p\n", \*q);

Se dopo le dichiarazioni iniziali eseguiamo le seguenti istruzioni:

int y = 20;

\*p = y;

Come cambierà la rappresentazione in memoria? Cosa stamperà ora printf("%d\n", x);?

7. Analizzare il seguente codice:

int\* createDanglingPointer() {

int local = 42;

return &local;

}

int main() {

int \*ptr = createDanglingPointer();

printf("%d\n", \*ptr);

return 0;

}

a) Spiegare perché questo codice crea un dangling pointer.

b) Quali sono i potenziali problemi che possono derivare dall'uso di questo dangling pointer?

c) Come si potrebbe modificare la funzione createDanglingPointer() per evitare il dangling pointer?

8. Considerare il seguente codice:

int \*p, \*q;

p = (int\*)malloc(sizeof(int));

\*p = 10;

q = p;

free(p);

\*q = 20;

a) Dopo l'esecuzione di free(p), cosa rappresenta q?

b) Quali sono i rischi associati all'uso di q dopo la chiamata a free(p)?

c) Come si potrebbe modificare questo codice per evitare potenziali problemi di memoria?

# Esercizi

1. Implementare una funzione ricorsiva che inverta una lista concatenata. La funzione deve avere la seguente firma:

struct Node\* reverse\_list(struct Node\* head);

Dove Node è definito come:

struct Node {

int data;

struct Node\* next;

};

Esempio:

Input: 1 -> 2 -> 3 -> 4 -> NULL

Output: 4 -> 3 -> 2 -> 1 -> NULL

Implementare la funzione in modo efficiente, discutendo la strategia utilizzata e analizzando la complessità computazionale.

2. Implementare una funzione che trovi il più lungo cammino in un albero binario. La funzione deve avere la seguente firma:

int longest\_path(BTree\* root);

Dove BTree è definito come:

typedef struct btree {

int value;

struct btree\* left;

struct btree\* right;

} BTree;

La funzione deve restituire la lunghezza del cammino più lungo dalla radice a una foglia.

Esempio:

Per l'albero:

1

/ \

2 3

/ \ \

4 5 6

\

7

La funzione deve restituire 4 (cammino 1 -> 2 -> 5 -> 7)

Scrivere la funzione in modo ricorsivo, specificando PRE e POST condizioni e discutendone brevemente la correttezza.