1. Attraversamento di un campo fiorito (8 punti, integrazione ≥ 6 crediti)

Implementa una funzione ricorsiva che calcoli il numero di percorsi possibili per attraversare un campo fiorito rappresentato da una matrice. Il campo è rappresentato da una matrice di dimensioni DIM_X x DIM_Y, dove 0 indica la presenza di un fiore e 1 l'assenza. Si può muovere solo verso l'alto o verso destra. La funzione deve restituire il numero di percorsi validi dal basso verso l'alto del campo.

```
#define DIM_X 5
#define DIM_Y 5

int attraversa_campo(int campo[DIM_X][DIM_Y], int pos_x, int pos_y);
```

2. Rimozione di triple consecutive (7 punti, integrazione ≥ 6 crediti)

Scrivi una funzione che, dato un array di interi, rimuova tutti gli elementi uguali al precedente e al successivo (il primo e l'ultimo elemento non vanno mai rimossi). La funzione deve modificare l'array in-place e aggiornare la dimensione.

```
void rimuovi_triple(int *A, int *dim);
```

3. Ricerca in array con elementi ordinati a coppie (8 punti, integrazione ≥ 6 crediti)

Implementa una funzione ricorsiva che, dati due array A e B, restituisca 1 se tutti i valori di A sono presenti in B (con lo stesso segno o segno opposto) nello stesso ordine, ma non necessariamente consecutivi. Restituisca 0 altrimenti.

```
int ricerca_ordinata(int *A, int *B, int dim_A, int dim_B);
```

4. Clonazione di una lista con skip (6 punti, integrazione ≥ 3 crediti)

Scrivi una funzione che, data una lista collegata, ne crei una copia saltando un elemento ogni due. Se la lista originale è 1->2->3->4->5, la nuova lista sarà 1->3->5.

```
struct nodo {
    float value;
    struct nodo *nextPtr;
};

typedef struct nodo Lista;
```

```
void clone_list_skip(Lista *srcPtr, Lista **destPtr);
```

5. Bilanciamento di un albero binario (7 punti, integrazione ≥ 3 crediti)

Implementa una funzione che verifichi se un albero binario è bilanciato. Un albero è considerato bilanciato se per ogni nodo la differenza di altezza tra il sottoalbero sinistro e destro non supera 1.

```
struct btree {
    int valore;
    struct btree *leftPtr;
    struct btree *rightPtr;
};

typedef struct btree BTree;

int is_balanced(BTree *root);
```

Certo, ecco un esercizio alternativo che coinvolge la manipolazione di array, con un livello di difficoltà intermedio ma comunque interessante:

6. Rotazione ciclica di un array (7 punti, integrazione ≥ 6 crediti)

Implementa una funzione `rotate_array` che ruota ciclicamente un array di `n` elementi di `k` posizioni verso destra. La rotazione deve essere fatta in-place, ovvero senza utilizzare un array ausiliario. La funzione deve gestire efficacemente anche il caso in cui `k` sia maggiore di `n`.

Esempio:

- Input: A = [1, 2, 3, 4, 5], n = 5, k = 2
- Output: A = [4, 5, 1, 2, 3]

```
void rotate_array(int A[], int n, int k);
```

Mi scuso per l'equivoco. Hai ragione, è meglio concentrarsi sulle strutture e i concetti presentati nei file d'esame che hai fornito. Ecco alcuni esercizi più in linea con quelli:

7. Manipolazione di lista collegata (6 punti, integrazione ≥ 3 crediti)

Implementa una funzione ricorsiva che rimuova dalla lista tutti gli elementi il cui valore è multiplo di 3.

```c
struct nodo {
    int value;
    struct nodo *nextPtr;
};

typedef struct nodo Lista;

void remove_multiples_of_three(Lista **ptrPtr);
```

8. Operazioni su albero binario di ricerca (7 punti, integrazione ≥ 3 crediti)

Scrivi una funzione ricorsiva che calcoli la somma dei valori di tutti i nodi che hanno esattamente un figlio in un albero binario di ricerca.

```c
struct btree {
    int valore;
    struct btree *leftPtr;
    struct btree *rightPtr;
};

typedef struct btree BTree;

int sum_single_child_nodes(BTree *root);
```

9. Manipolazione di array bidimensionale (8 punti, integrazione ≥ 6 crediti)

Implementa una funzione che, data una matrice quadrata, scambi gli elementi della diagonale principale con quelli della diagonale secondaria.

```c
#define N 4
void swap_diagonals(int matrix[N][N]);
```

10. Ricorsione su array (8 punti, integrazione ≥ 6 crediti)

Scrivi una funzione ricorsiva che, dato un array di interi, restituisca 1 se l'array contiene una sequenza crescente di lunghezza almeno 3, 0 altrimenti.

```c
int has_increasing_sequence(int A[], int size);
```