1. Considerate questo programma.

```
int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
   z  = **ppz;
  *py += 2;
   y = *py;
   x += 3;
   py = &x;
   return x + y + z;
}

void main()
{
   int c, *b, **a;
   c = 4;
   b = &c;
   a = &b;
   printf("%d ", f(c, b, a))
   printf( "%d", c);
   return 0;
}
```

a) Cosa stampa il programma? **Risposta:**

b) Rappresentate graficamente lo stato della memoria all'inizio della chiamata di f e subito prima che la f termini. In entrambe le situazioni, distinguete lo spazio del main e quello della funzione f.

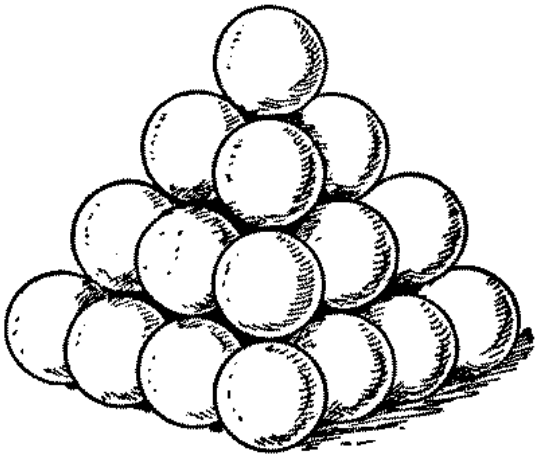
Risposta:

2. Assumendo che un float occupi 4 byte, prevedete quale sarà l'output di questo programma:

```
#include <stdio.h>

int main()
{
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;

    printf("%f ", *ptr2);
    printf("%d", ptr2 - ptr1);

   return 0;
}
```

Risposta:

3. Dei sassi sferici sono ammucchiati a formare una piramide, con un sasso in cima, posto al centro di un quadrato formato da 4 sassi (2 per lato), posti a loro volta sopra un quadrato formato da 9 sassi (3 per lato), e così via.



Scrivete una funzione ricorsiva
int sassi(int height)
che, data l'altezza `height` della piramide, restituisca il numero di sassi che la compongono.

Ad esempio:
f(1) deve restituire 1
f(2) deve restituire 5.

Risposta:

4. Sia Node il tipo che rappresenta un nodo di una lista concatenata semplice. La funzione `move_to_front` riceve l'indirizzo `head` del primo nodo di una lista e deve modificare la lista spostando l'ultimo nodo all'inizio della lista, quindi restituire l'indirizzo della lista modificata.

La funzione è incompleta.

```
Node *move_to_front( Node *head ) {
  Node *p, *q;
  q = NULL; p = head;
  while ( p -> next != NULL )  {
    q = p;
    p = p -> next;
  }

  // ......

  return head;
}
```

a) Quali sono le istruzioni che mancano per completare la funzione? Scegli tutte le opzioni corrette.

- [ ] **p -> next = head; head = p; q -> next = NULL;**
- [ ] **q -> next = NULL; p -> next = head; head = p;**
- [ ] **q = NULL; p->next = head; head = p;**
- [ ] **q->next = NULL; head = p; p->next = head;**
- [ ] **head = p; p->next = q; q->next = NULL;**

- [ ] **head = p; q -> next = NULL; p -> next = head;**

b) Se N è la lunghezza della lista, quale è la complessità in tempo della funzione?

- ○ **log N**
- ○ **2^N**
- ○ **N**
- ○ **N logN**
- ○ **N^2**

c) Scrivete in C una definizione per il tipo Node

Risposta:

d) Se si mantenesse anche un **puntatore all'ultimo elemento** della lista, si potrebbe implementare la funzione f in maniera più efficiente?

- ○ **Sì**
- ○ **No**
- ○ **Non so**

e) Giustificate la risposta. Se avete risposto sì, **spiegate** come si può ottenere una soluzione migliore, **riscrivete** la funzione (se lo ritenete opportuno potete modificare il prototipo) e **indicatene la complessità**. Se avete risposto di no, **spiegate perché** e **mostrate** un esempio in cui non si può ottenere un tempo di esecuzione migliore.

Risposta:

f) Se la lista fosse **doppiamente concatenata**, e si mantenesse anche un **puntatore all'ultimo elemento** della lista, si potrebbe implementare la funzione f in maniera più efficiente?

- ○ **Sì**
- ○ **No**
- ○ **Non so**

g) Giustificate la risposta. Se avete risposto sì, **spiegate** come si può ottenere una soluzione migliore, **riscrivete** la funzione (se lo ritenete opportuno potete modificare il prototipo) e **indicatene la complessità**. Se avete risposto di no, **spiegate perché** e **mostrate** un esempio in cui non si può ottenere un tempo di esecuzione migliore.

Risposta:

5. Dato un **albero binario di ricerca** X, chiamiamo pre(X) il vettore che contiene la sequenza delle chiavi di X che si ottiene visitando X con ordine anticipato (pre-order).

Sia T un albero tale che pre(T) = {28, 17, 10, 15, 25, 23, 36, 34, 44}.

a) Qual è la radice dell'albero T? **Risposta:**

b) Qual è il figlio sinistro della radice di T? **Risposta:**

c) Qual è il figlio destro della radice di T? **Risposta:**

d) Ricostruite l'albero T (fatene un disegno oppure datene una descrizione).

Risposta:

e) **Scrivete una funzione C** che, dato un intero n e un vettore A di n interi, tale che A = pre(X) per qualche albero binario di ricerca X, restituisca, se esiste, l'indice corrispondente al figlio destro della radice di X. Se tale figlio non esiste, la funzione deve restituire -1.

Indicate il tempo di esecuzione della funzione; più la funzione è veloce, meglio è.

Risposta:

f) Dato il vettore pre(X) di un qualsiasi albero binario di ricerca X, è sempre possibile ricostruire X a partire da pre(X)?

- ○ **Sì**
- ○ **No**
- ○ **Non so**

g) Dato un vettore pre(X) di un qualsiasi albero binario di ricerca X, progettate e descrivete un algoritmo per ricostruire X a partire da pre(X). Negli eventuali casi in cui non si può

costruire l'albero, l'algoritmo deve stampare un messaggio d'errore. **Indicate il tempo di esecuzione** dell'algoritmo; più l'algoritmo è veloce, meglio è.

Potete spiegare la vostra soluzione a parole (siate brevi), con pseudocodice, con degli schemi, oppure potete scriverla in C.

Risposta: