Esercizio 1

$$
\begin{aligned}
foo(19) &= 1 + foo(18) \\
&= \underbrace{1 + (2}_{} + foo(9)) \\
&= \underbrace{3 + (1}_{} + foo(8)) \\
&= 4 + (2 + foo(4)) \\
&= 6 + (2 + foo(2)) \\
&= 8 + (2 + foo(1)) = 10 + 5 = 15
\end{aligned}
$$

Si nota che al più dopo una coppia di passi consecutivi l'argomento della funzione foo() si dimezza, infatti

$$
T(n) = \begin{cases} T\left(\frac{n}{2}\right) + O(1) & \text{se } n \text{ pari} \quad (1) \\ T(n-1) + O(1) & \text{se } n \text{ dispari} \end{cases}
$$

ma la seconda equazione si può sviluppare per un altro passo e quindi ottenere per n dispari che:

$$
T(n) = T\left(\frac{n-1}{2}\right) + \underbrace{O(1) + O(1)}_{O(1)} \quad (2)
$$

Ne consegue da (1) e (2) che:

$$
\begin{aligned}
T(n) &\leq T\left(\frac{n}{2}\right) + O(1) \quad \forall n \\
&= O(\log n)
\end{aligned}
$$

Esercizio 2

Progettiamo un algoritmo ricorsivo di visita dell'albero binario T che riceve in input un nodo u dell'albero, di cui si vuole calcolare la media dei valori da esso discendenti, e restituisce in output 4 "valori":

- dim = #nodi discendenti da u in T, con u incluso
- val = somma dei valori contenuti nei nodi discendenti da u, con u incluso
- maxM = media massima tra i sottoalberi dei nodi discendenti da u
- max = radice del sottoalbero con media maxM

```
                            dim  val  maxM   max
                             ↓    ↓    ↓    ↓
MediaMax(u)
   if (u==NIL) return  <     0 ,  0 ,  0 , NIL>
   else
      <dimS, valS, maxMS, maxS> = MediaMax(u.left);
      <dimD, valD, maxMD, maxD> = MediaMax(u.right);
      dim = dimS + dimD + 1;
      val = valS + valD + u.key;
      if (val/dim > max(maxMS, maxMD)) { max = u;
                                         maxM = val/dim; }
      else { maxM = max(maxMS, maxMD);
             if (maxM==maxMS) max = maxS;
             else max = maxD; }

   return <dim, val, maxM, max>;
```

Esercizio 3

Esistono tre modi per risolvere il problema con complessità in tempo $m(m+n)$, $n(m+n)$, $m+n$.

Descriveremo formalmente il primo e accenneremo a parole agli altri due.

L'idea algoritmica consiste nel rimuovere a turno ogni arco (u,v) del grafo G, verificando se si sono realizzate due componenti connesse. Per effettuare quest'ultima verifica, e come visto in classe, basta fare una visita di $G - \{(u,v)\}$ controllando alla fine se tutti i nodi sono stati raggiunti. Chiaramente non importa il nodo radice della visita che può essere qualunque. Come visita utilizzeremo la BFS, e alla fine controlleremo se tutti i nodi sono stati colorati di NERO (G ancora connesso) oppure no (G si è disconnesso).

L'unico accorgimento da prestare consiste nell'evitare l'uso dell'arco (u,v), che quindi viene passato come ulteriore parametro alla "nuova" BFS che quindi risulta:

```
checkConnesso (G, u, v)
foreach x ∈ V - {s}
  x.color = bianco;
s.color = grigio;
Q = ∅;
Enqueue (Q, s);
while Q ≠ ∅ do
  x = dequeue (Q);
  foreach y ∈ Adj[x] do
    if (x ≠ u ∧ y ≠ v ∧ y.color = bianco)
      {y.color = grigio; enqueue (Q, y);}
  x.color = nero;

connesso = True;
foreach x ∈ V do
  connesso ∧= (x.color == nero)
return connesso;
```

A questo punto basta costruire una procedura che rimuove ogni arco di G e ne verifica la connettività.

```
Ponte (G)
  foreach (u,v) ∈ E do
     if (deanConnesso(G, u, v) == false) return (u,v);

  return NIL;
```

La seconda soluzione che proponiamo restringe la rimozione a un sottoinsieme degli archi di G, di dimensione n. Questi archi sono quelli di un albero BFS o DFS di G. Infatti, se il ponte esiste, allora l'albero di copertura di G deve necessariamente includerlo.

La terza soluzione, più sofisticata e ottima in tempo, si basa ~~parte~~ ~~della~~ sulla considerazione che l'arco (u,v) non è un ponte se fa parte di un ciclo. Siccome G è un grafo non orientato sappiamo dal Teorema 22.10 che in una visita DFS un arco o è dell'albero o è all'indietro. Pertanto, per ogni nodo x, calcoliamo il minimo valore di d[y] per ogni nodo y destinazione di un arco all'indietro (x,y). Confrontando poi opportunamente min[y] con d[x] è possibile capire, se min[y] ≤ d[x], che l'arco non è un ponte. Esistono anche altre varianti che però seguono lo stesso principio e richiedono solo una visita di G.

Esercizio 4

Lo schema algoritmico da applicare è quello della Generazione Binaria operante sul vettore V.

La famiglia di sottoinsiemi S^* è un exact cover sse ogni intero $j \in [1, n]$ appartiene a uno solo degli insiemi di S^*. Pertanto specializziamo la procedura Elabora affinché esegua questa verifica:

```
Elabora (V, S, n, m)
  for j = 1 To n do
      conta = 0;
      for i = 1 To m do
          if (S[i][j] == 1 ∧ V[i] == 1) conta++;
      if (trovato ≠ 1) return false;
  return true
```

La complessità in tempo di Elabora è $O(n \cdot m)$.
Il fattore moltiplicativo n potrebbe essere ridotto a $|S^*|$ restringendo il secondo ciclo for a una scansione solo sugli indici "i" corrispondenti ai sottoinsiemi di S^*, e quindi alle posizioni per cui $V[i] = 1$.