```
procedure ordinaperfusioni(var a: nelements; n: integer);
var b: nelements;
    procedure mergesort(var a,b: nelements; primo,ultimo:integer);
    var q: integer;
        procedure merge(var a,b: nelements; primo, ultimo, mezzo: integer);
        var i,j,k,h: integer;
        begin {merge}
        …
        end {merge}
    begin {mergesort}
        if primo<ultimo then
            begin
                q := (primo+ultimo) div 2; mergesort(a, b, primo, q);
                mergesort(a, b, q+1, ultimo); merge(a, b, primo, ultimo)
            end
    end {mergesort}
begin {main}
    mergesort(a,b,1,n)
end
```

Ricerca binaria

```
procedure binarysearch(var a: nelements; n,x: integer; var found: boolean);
var lower, upper, middle: integer;
begin
    lower := 1;
    upper :=n;
    while lower<upper do
      begin
        middle := (lower+upper) div 2;
        if x>a[middle] then
                lower := middle+1;
        else
                upper := middle
      end;
    found := (a[lower]=x)
end
```

Operazione dominante: confronto

Ricerca binaria

La complessità di questo algoritmo non dipende dalla distribuzione dei dati.

Il ciclo più interno è ripetuto fino a quando la distanza dei limiti inferiore e superiore del segmento in cui ricercare ***x*** è maggiore di zero.

Osserviamo che la distanza si dimezza ad ogni ciclo.

Inizialmente è n, quindi $n/2$, poi $n/4$, e così via.

In generale alla k-esima iterazione la distanza è pari a $n/2^{k-1}$.

Il ciclo si arresta al passo k successivo a quello in cui

$n/2^{k-1}=1$

Da cui $k=\lfloor \log_2 n \rfloor +1$. Poiché ad ogni passo si effettua un solo confronto, il numero massimo di confronti sarà:

$$f(n)= \lfloor \log_2 n \rfloor +1$$

Ricerca binaria (vers. ricorsiva)

```
function RICBIN(var a: nelements; x,lower,upper: integer): boolean;
var middle: integer;
begin
    if lower> upper then
          RICBIN :=false;
    else
      begin
          middle := (lower+upper) div 2;
          if x=a[middle] then
                    RICBIN := true;
          else if x < a[middle] then
                    RICBIN := RICBIN(a, x, lower, middle-1)
               else
                    RICBIN := RICBIN(a, x, middle+1, upper)
      end
end
```

Operazione dominante: confronto

Ricerca binaria (vers. ricorsiva)

Posto per semplicità $n=2^k$, analizziamo la complessità di questo algoritmo nel caso pessimo (questa volta dipende dai dati) mediante relazioni di ricorrenza.

f(n)=1 se n=1

$f(n)=f((n-1)/2) + 1 \leq f(n/2)+1$ per n>1.

Ricordiamo il seguente risultato per relazioni di ricorrenza con lavoro di combinazione indipendente da n:

c) relazioni con partizione dei dati

$f(1)=c$

$f(n)= a \cdot f(n/k) + b$ per $n=k^m$, m intero e n>1

Se a = 1 allora $f(n)$ è O(log n).

Se a > 1 allora $f(n)$ è $O(n^{\log_k n})$.

Quindi si giunge immediatamente a una complessità logaritmica.

Calcolo dell'*n*-esimo numero di Fibonacci

```
function RFIB(n: integer): integer;
begin
   if n=0 then RFIB := 0
          else if n=1 then RFIB := 1
                      else  RFIB := RFIB(n-1) + RFIB(n-2)
end
```

Operazione dominante: addizione

La complessità non dipende dalla distribuzione dei dati in ingresso.
Essa è definita mediante la seguente relazione di ricorrenza:

f(n) = 0 per n=0,1

f(n)= f(n-1) + f(n-2) + 1 per n > 1

Calcolo dell'*n*-esimo numero di Fibonacci

Ricordiamo il seguente risultato per relazioni di ricorrenza con lavoro di combinazione indipendente da n:

a) relazioni lineari di ordine *h*

$f(1)=c_1$ $f(2)=c_2$ ... $f(h)=c_h$

$f(n)= a_1 f(n-1) + a_2 f(n-2) + ... + a_h f(n-h) + b$ per $n > h$

Allora $f(n)$ è $O(a^n)$ con $a > 1$, ovvero f è esponenziale.

Quindi la complessità in tempo del precedente algoritmo è esponenziale!

Come mai? Per calcolare l'n-esimo numero di Fibonacci F_n si invoca indipendentemente RFIB sia per calcolare F_{n-1} che per calcolare F_{n-2}; benché all'interno di RFIB(n-1) venga di fatto calcolato F_{n-2}, tale calcolo non viene utilizzato dalla RFIB(n-2) ; e ciò avviene ad ogni passo della ricorrenza.

Calcolo dell'*n*-esimo numero di Fibonacci

```
function RFIBLIN(n: integer; var s:integer): integer;
var temp: integer;
begin
    if n=1 then
          begin
            s := 0
            RFIBLIN:= 1
          end
          else
          begin
             temp := RFIBLIN(n-1,s);
             RFIBLIN := temp+s;
             s := temp
          end
end
```

Operazione dominante: addizione

Calcolo dell'*n*-esimo numero di Fibonacci

La funzione RFIBLIN contiene ora una sola chiamata ricorsiva, che restituisce però i due valori della successione F_{i-1} (in RFIBLIN) e F_{i-2} (in s) necessari al calcolo del nuovo valore F_i. F_{i-1} sarà a sua volta memorizzato in s e restituito come parametro per contribuire al calcolo di F_{i+1}.

La complessità è definita mediante la seguente relazione di ricorrenza:

$f(n) = 0$ per $n=1$

$f(n)= f(n-1) + 1$ per $n > 1$

Ricordiamo il seguente risultato per relazioni di ricorrenza con lavoro di combinazione indipendente da n:

Calcolo dell'*n*-esimo numero di Fibonacci

b) relazioni lineari di ordine h, con $a_h = 1$ e $a_j = 0$ per $1 \le j \le h-1$

$f(1)=c_1$ $f(2)=c_2$ ... $f(h)=c_h$

$f(n)= f(n-h) + b$ per $n > h$

Allora $f(n)$ è $O(n)$, cioè è lineare.

dove h=1 in questo caso. Quindi RFIBLIN ha complessità lineare.

Calcolo dell'*n*-esimo numero di Fibonacci

Osserviamo che le relazioni di ricorrenza possono essere applicate anche per la valutazione della complessità in spazio. Bisogna tuttavia tenere presente che tempo e spazio differiscono per il seguente aspetto: il tempo, una volta consumato, non può essere riutilizzato; lo spazio di memoria può invece in certi casi essere ricuperato e riutilizzato.

Ad esempio, osserviamo che al ritorno della chiamata RFIB(n-1), lo spazio di memoria usato per l'esecuzione di RFIB(n-1) può essere ricuperato e riutilizzato per RFIB(n-2). In sostanza tutto procede come se i richiami ricorsivi fossere uno invece di due. La relazione di ricorrenza diventa:

$f(n) = c$ per $n=0,1$

$f(n)= f(n-1) + b$ per $n > 1$

che ha soluzione di ordine lineare.

Riepilogando, RFIB ha complessità in tempo esponenziale, ma complessità in spazio lineare.