La soluzione più logica consiste nell'adottare un approccio ricorsivo al problema, ovvero sapendo che la serie ha come valori:

$F_1 = 1 \quad F_2 = 1 \quad F_n = F_{n-1} + F_{n-2}, n > 2$

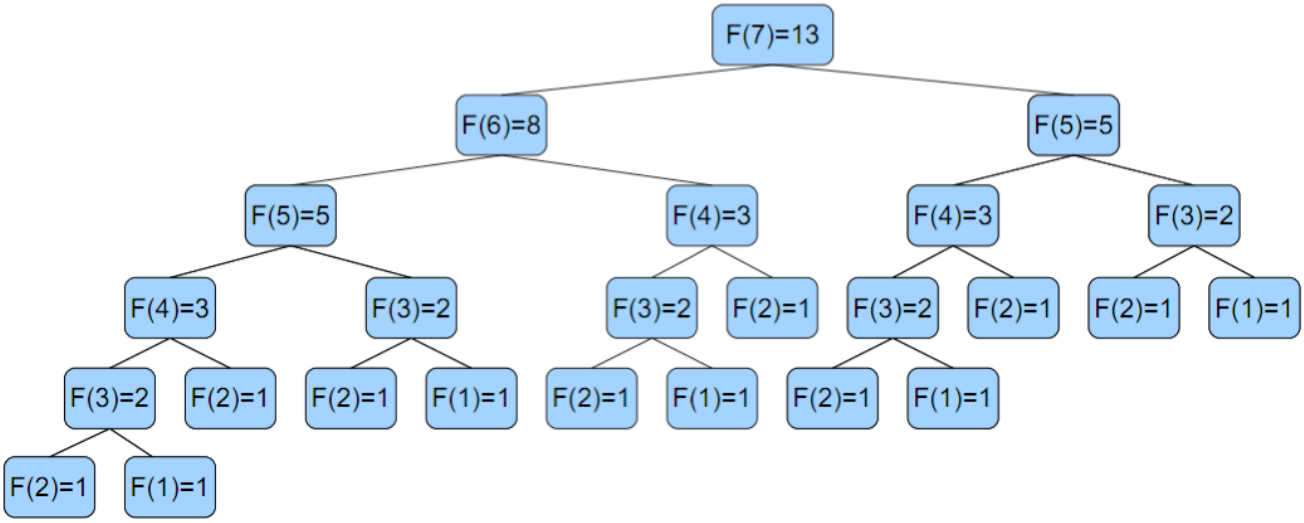
pseudo-codice:

```
algoritmo Fibonacci2 (int n) -> int //un algoritmo che dato un intero restituisce un intero
if(n==1||n==2) then //caso base conosciuto, ovvero conosciamo i soli valori della serie (F1=1 F2=1)
        return 1;
else
        return Fibonacci2(n-1)+Fibonacci2(n-2)      //iterazione ricorsiva
endif
```

Con l'approccio ricorsivo non verrà mai calcolata la funzione per n=0, anzi per meglio dire in questo blocco di pseudo codice non è considerata l'opzione che l'n passato dal chiamante della funzione possa essere proprio '0'.

Limiti della funzione:

La funzione ricorsiva richiede un tempo di esecuzione molto alto, per meglio dire: per le esecuzioni con n<50 (dove 50 è un numero determinato dalla potenza dell'hardware sul quale viene eseguito il programma) il tempo di calcolo è accettabilmente basso, salvo poi crescere esponenzialmente per valori più alti di tale soglia.
Questo perché ad ogni invocazione della funzione (quindi anche quando viene chiamata all'interno del blocco ricorsivo) vengono calcolati tutti i valori precedenti della serie che portano a quel risultato *(struttura ad albero).*



I tempi di esecuzione dipendono dal linguaggio scelto, dalla macchina e l'implementazione della formula matematica a volte risulta difficile.
Calcolando il numero di operazioni elementari eseguite nello pseudo-codice posso fare una stima del costo computazionale dell'algoritmo.

operazioni elementari=operazioni che richiedono un tempo costante per l'esecuzioni

Le istruzioni che richiedono tempo costante sono gli operatori matematici (+, -, *, /) e quelli logici (&&, ll, !). Le chiamate ricorsive o generalmente le chiamate a funzione NON richiedono tempo costante, ma richiedono il tempo dato dalla stima dei tempo delle istruzioni elementari interne ad essa (quindi la chiamata di per sé non comporta nulla, ma è il suo contenuto ad aumentare le operazioni elementari da eseguire).
Per esempio nella nostra funzione sopra descritta ci sono 5 operazioni elementari che vengono eseguite ad ogni chiamata.

Formula chiusa di Fibonacci:

Col tempo si è arrivati a determinare una formula chiusa per calcolare il valore di un numero all'interno della serie di Fibonacci, tale formula introduce un coefficiente di errore dato dal fatto che usa numeri irrazionali, diciamo che il valore calcolato in questo modo si approssima al valore ottenuto dall'esecuzione della nostra funzione ricorsiva.

$$F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right) \quad dove \quad \phi^n = \frac{1+\sqrt{5}}{2} \approx 1.618 \quad \hat{\phi}^n = \frac{1-\sqrt{5}}{2} \approx -0{,}618$$

Calcolo del numero di nodi dell'albero di ricorsione per calcolare Fn:

$$T(n)\begin{cases} 1 \quad se \quad n = 1 \,\|\, n = 2 \\ 1 + T(n-1) + T(n-2) \quad se \quad n > 2 \end{cases}$$

Esecuzione del nodo stesso — Ramo sx — Ramo dx

T(n) è il numero di operazioni che vengono eseguite per il calcolo dell'n-esimo valore.

Il legame fra il numero di operazioni e la serie di Fibonacci è:

$T(n)=2F_n-1$

Pertanto possiamo dire che il numero di operazioni necessarie cresce esponenzialmente all'aumentare di n, così come per la serie di Fibonacci.

DIMOSTRAZIONE:

$T(n) = T(n-1) + T(n-2) + 1$ sappiamo inoltre che i valori di T(n) sono monotoni e crescenti, ovvero T(n-1) sarà ≥ T(n-2)

possiamo quindi continuare la dimostrazione dicendo che:

$$T(n-1) \geq T(n-2)$$

$$allora$$

$$T(n-2) + T(n-1) + 1 \geq T(n-2) + T(n-2) + 1$$

$$T(n-1) + T(n-2) + 1 > 2T(n-2) + 2 + 1$$

$$quindi \quad anche$$

$$T(n) \geq 2T(n-2) + 1$$

$$Vale \quad anche:$$

$$T(n-2) \geq 2T(n-4) + 1$$

$$ovvero:$$

$$T(n) \geq 2\left[2T(n-4) + 1\right] + 1$$

$$T(n) \geq 4T(n-4) + 2 + 1$$

queste supposizioni sono applicabili ricorsivamente k volte

$$T(n) \geq 8T(n-8) + 2^2 + 2 + 1$$

fino ad avere:

$$T(n) \geq 2^k T(n-2k) + \sum_{i=0}^{k-1} 2^i$$

La ricorsione termina quando si ha k=$\frac{n}{2}$

(sostituisco $\frac{n}{2}$ a k)

$T(n) \geq 2^{\frac{n}{2}} T(n-2\frac{n}{2}) + \frac{2^{\frac{n}{2}}-1}{2-1}$ che sarà sicuramente maggiore di $2^{\frac{n}{2}}$ pertanto T(n) avrà sicuramente una crescita esponenziale.

Come evitare di calcolare più volte gli stessi valori?

Potrei utilizzare un array di appoggio e non affidarmi ad una soluzione ricorsiva, partendo a calcolare i miei valori non a ritroso (ovvero da F_n) ma direttamente da F_1.

Pseudocodice:

```
algoritmo Fibonacci3(int n) ⟶ int
        Sia Fib[1...n] un array di n interi
        Fib[1]:=1;
        Fib[2]:=1;
        for i:=3 to n do
                Fib[i]:= Fib[i-1]+ Fib[i-2];
        endfor
        return Fib[n];
```

Il valore di ogni i-esimo valore della successione di Fibonacci viene calcolato una ed una sola volta e memorizzato una ed una sola volta all'interno dell'array.

Quanto vado a risparmiare concretamente?

Il costo di questo algoritmo è proporzionale ad n. Quindi è sicuramente meglio della versione ricorsiva (che ricordiamo aveva una crescita di T(n) esponenziale).
A differenza dell'algoritmo precedente, questo algoritmo richiede dalla memoria, in particolar modo per la struttura dati (array) che memorizza i valori della serie, lo spazio che serve è proporzionale al numero di elementi.