Esercizi Complessità

1. Studiate la complessità in tempo del seguente algoritmo iterativo:

```
Fill( Array of int A):int
int n := length(A);
int z := 1;
int i := 0;
for i := 1; i++; n do
    A[i]:= 2^i
for i := 1; i++; n do
    j := A[i];
    while j ≥ 1 do
        z := z + 1; j := ⌊j/2⌋;
return: z
```

2. Studiate con l'albero della ricorsione la complessità in tempo del seguente algoritmo ricorsivo: Provate per induzione la stima fatta.

```
Tip( n :int):int
int z := 1;
for i := 1; i++; n do
    for j := 1; j++; n/i do
        z := z + 1
if (n > 2) then
    return: Tip(⌊n/2⌋) * Tip(⌊n/2⌋ + 2 * z
else
    return: z
```

3. Verificare o confutare applicando la definizione di ordine di grandezza:

   (a) $2^{\frac{3}{2}\log_2 n} \in \Theta(n)$

4. Calcolare il limite superiore e il limite inferiore della complessità in tempo dei seguenti algoritmi:

   - ```
     Test;
     int t := n;
     while t > 1 do
         t := t/16
     ```

   - ```
     Test;
     int t := n;
     while t > 1 do
         t := log_2 n
     ```

   - ```
     Test;
     int t := n;
     while t > √n do
         t := t − 5
     ```
     ```

- ```
  Test;
  int a := ⌊log_2 n⌋;
  int r := 1;
  while a > 2 do
      a := a − 2;
      r := 2r;
  while r > 1 do
      r := r − 1
  ```

5. Stimare il limite superiore e inferiore della complessità in tempo del seguente algoritmo:

```
Esercizio2(A : array of int, s, e: int)
if (e − s + 1) > 3 then
    int q1 := s + ⌈(e−s+1)/3⌉;
    int q2 := s + 2⌊(e−s+1)/3⌋;
    Esercizio2(A,q1,e);
    Esercizio2(A,q1,q2-1);
```

6. Provare per quali valori di a e b la seguente eq. di ricorrenza cade nel terzo caso del Master Theorem:

   - $T(n) = \begin{cases} 1 & n \leq 2 \\ aT(\frac{n}{b}) + n^3 & n \geq 3 \end{cases}$

7. Calcolare il limite superiore alla complessità in tempo dei seguenti algoritmi.

   - ```
     Test 1 (n : int);
     while n > 2 do
         n := √n;
     ```

   - ```
     Test 2(n : int):int;
     if n > 15 then
         int c := 0;
         i := n;
         while i ≥ 0 do
             c++; i−−
         Test2:= Test2(⌊n/4⌋)+2;
     else
         Test2:= 6
     ```

8. Calcolare il limite superiore alla complessità in tempo dei seguenti algoritmi.

   - ```
     Test (n : int);
     if n > 1 then
         n := n − 1;
         Test1(n);
     else
         int t := n;
         while t > 1 do
             t := t − 1;
     ```

   - ```
     Test (n : int);
     if n > 1 then
         int t := n;
         while t > 1 do
             t := log_2(t);
         n := ⌊n/2⌋;
         Test2(n);
     ```
     ```

9. Calcolare il limite superiore alla complessità in tempo dei seguenti algoritmi:

   - ```
     Test 1 (n : int):int;
     if n > 1 then
         n := n − 5;
         return 5 ∗ Test1(n);
     else
         return 0
     ```

   - ```
     Test 2(n : int):int;
     a := 0;
     while n > 2 do
         n := ⌊n/2⌋; a := a + 1;
     while a > 1 do
         b := a;
         while b ≥ 1 do
             b := b − 1;
         a := a − 1;
     ```

10. Scrivete e calcolate l'ordine di grandezza dell'equazione $T(n)$ che descrive il tempo di esecuzione del seguente algoritmo.

```
Pippo  (n : integer) ;
{int a=0;
for (i = 1; i ≤ n; i++)
 for (j = 1; j ≤ n; j++)
  for (k = 1; k ≤ i; k++)
    for (w = 1; w ≤ j; w++)
    {a++}
if (n > 10) {Pippo= 2*Pippo (n-5)+a}
     else {Pippo=100}
}
```

11. Stimate **una** delle due equazione di ricorrenza con i metodi visti a lezione (ad es., albero delle decisioni, soluzione eq. ricorrenza lineare omogenea), e verificate la soluzione per induzione:

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + 4 & n \geq 2 \\ 1 & n = 1 \end{cases} \qquad T(n) = \begin{cases} 5T(n-1) - 6T(n-2) & n \geq 2 \\ 1 & n = 1 \\ 0 & n = 0 \end{cases}$$

12. Stimate l'ordine di grandezza dell'equazione che descrive la complessità computazionale del seguente algoritmo:

```
Pluto1 (n: integer);
if (n > 10)
    {Pluto1= 2*Pluto1(⌊n/2⌋) + 1:
     int k=1; int w=1;
     while (k ≤ n)
       {for (int j = 0; j ≤ n ; j++) w = 2;
        k = k + 2}
    }
   else Pluto1=100;
}
```

13. Si consideri il ciclo così definito:

$$\texttt{while}\ n > 2\ \texttt{do}\ n := f(n);$$

Trovate il costo computazionale dell'esecuzione di tale ciclo `while` per ciascuna funzione $f(n)$:

- $f(n) = n - 10$
- $f(n) = n/4$
- $f(n) = \log n$
- $f(n) = \sqrt{n}$
- $f(n) = \frac{n}{\log n}$

14. Per uno stesso problema sono stati disegnati tre diversi algoritmi le cui complessità in tempo sono:

- $f_1(n) = \sum_{k=1}^{n}(k^2 + k)$
- $f_2(n) = \sum_{k=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n^2}{2^k} \right\rceil$
- $f_3(n) = \begin{cases} T(n-2) + 4 & n \geq 2 \\ 1 & n = 1 \end{cases}$

Qual'è l'algoritmo più efficiente? Motivare adeguatamente la risposta.

15. Studiare la complessità in tempo della seguente procedura e provare la stima per induzione:

```
Pippo(n : integer):integer;
if n ≤ 3 then
    return: 3
else
    int k := 1;
    while k ≤ n do
        int j := 1;
        while j ≤ 2n do
            j := j * 2
        k := k + 1;
    Pippo := 2* Pippo (⌊n/2⌋) + 5
```

16. Limitare inferiormente e superiormente in ordine di grandezza l'espressione:

$$\Pi_{k=1}^{n} k$$

17. La ricorrenza $T(n) = 8T(n/2) + n\log(n)$ descrive il tempo di esecuzione di un algoritmo A. Un altro algoritmo A' ha un tempo di esecuzione $T'(n) = aT'(n/4) + n^2$. Confrontare le complessità dei due algoritmi al variare del parametro a.

18. Risolvere con l'albero della risorsione l'equazione di ricorrenza che rappresenta la complessità in tempo dell'algoritmo:

```
function Pippo(n : integer):integer;
int k:= 1; a:=0; i:=1;
while k ≤ n do
    for int j:=i, j ≤ n, j++ do
        a:=a+1
    i++; k:=2*k;
if n ≥ 10 then
    Pippo:=Pippo(⌊n/2⌋)+ Pippo(⌊n/2⌋)+2 * √n
else
    Pippo:= 100
```

19. • Applicando la definizione di ordine di grandezza, provate o confutate che esiste una costante $c > 0$ tale che:

$$2^{O(\log n)} \in O(n^c)$$

• Trovate un limite asintotico superiore per la complessità in tempo risolvendo l'equazione di ricorrenza con l'albero della ricorsione e provandone la soluzione per induzione.

```
Pippo(n : integer):integer;
if n ≤ 3 then
    return: 3
else
    int a := n;
    while a > 2 do
        a := ⌊a/2⌋
    Pippo := 4*Pippo(⌊n/2⌋) + Pippo (⌊n/2⌋))
```