Correzione del facsimile C

Esercizio 1. *Dire quali delle seguenti affermazioni sono vere. Ove false, fornire un controesempio.*

1. $f(n) = O(g(n))$ *implica* $\log_2(f(n)+2) = O(\log_2(g(n)+2))$.
2. $f(n) = O(g(n))$ *implica* $h(f(n)) = O(h(g(n))$.
3. $f(n) = O(g(n))$ *implica* $f(n) + g(n) = \Theta(g(n))$.
4. $f(n) = O(f(n+1))$.

Svolgimento Esercizio 1

1. $f(n) = O(g(n))$ implica $\log_2(f(n)+2) = O(\log_2(g(n)+1))$?
   Vero. La dimostrazione formale, non richiesta, sarebbe la seguente. Sappiamo che $\exists\, k,\ \exists\, n_0 \mid \forall n \geq n_0, 0 \leq f(n) \leq kg(n)$. Sia $h = \max(1,k)$; abbiamo

   $$f(n) + 2 \leq kg(n) + 2 \leq hg(n) + 2 \leq hg(n) + 2h = h(g(n)+2).$$

   Siccome $\log n$ è una funzione crescente, otteniamo

   $$\log_2(f(n)+2) \leq \log_2 h + \log_2(g(n)+2).$$

   Notiamo anche che $\log_2(f(n)+2) \geq 1$, essendo $f(n) \geq 0$; quindi la positività è assicurata per ogni $n \geq n_0$. Essendo, per i medesimi valori di n, anche $g(n) \geq 0$, abbiamo allo stesso modo $\log_2(g(n)+2) \geq 1$, da cui

   $$\log_2(f(n)+2) \leq (\log_2 h) \cdot \log_2(g(n)+2) + \log_2(g(n)+2) = (\log_2 h + 1)\log_2(g(n)+2).$$

   Essendo $\log_2 h$ positivo, la costante trovata è certamente positiva.

2. $f(n) = O(g(n))$ implica $h(f(n)) = O(h(g(n)))$.
   Falso, come si vede prendendo $f(n) = n, g(n) = n^2, h(x) = 1/x$. Ma risulta falso anche per $h(n) \geq 1$ per ogni n, come si vede prendendo $f(n) = n, g(n) = 2n, h(x) = 2^x$.

3. $f(n) = O(g(n))$ implica $f(n) + g(n) = \Theta(g(n))$.
   Vero. Abbiamo infatti che

   $\exists\, k,\ \exists\, n_0 \mid \forall n \geq n_0, 0 \leq f(n) \leq kg(n)$. Quindi, sommando membro a membro $g(n)$, otteniamo $g(n) \leq f(n) + g(n) \leq (k+1)g(n)$, il che implica la tesi.

4. $f(n) = O(f(n+1))$.
   Falso, come si vede prendendo $f(n) = 2^{-n^2}$. Detto infatti $g(n) = f(n+1)$, abbiamo

   $$\frac{f(n)}{g(n)} = \frac{2^{-n^2}}{2^{-(n+1)^2}} = 2^{(n+1)^2 - n^2} = 2^{2n+1}$$

   il cui limite è infinito, per cui $f(n) \in \omega(g(n))$. Se $f(n)$ è non decrescente, la tesi è invece senz'altro vera.

Esercizio 2. *Siano $f(n)$ e $g(n)$ due funzioni definitivamente non negative. Dimostrare che* $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$.

Svolgimento Esercizio 2

Sappiamo che $\exists\, n_1 \mid \forall n \geq n_1, f(n) \geq 0$. Analogamente, sappiamo che $\exists\, n_2 \mid \forall n \geq n_2, g(n) \geq 0$. A partire da $n_0 = \max\{n_1, n_2\}$, le due funzioni sono quindi entrambe non negative. Per ogni $n \geq n_0$ vale poi $f(n) + g(n) \leq 2\max\{f(n), g(n)\}$. Riassumendo

$$\forall n \geq n_0, 0 \leq \max\{f(n), g(n)\} \leq f(n) + g(n) \leq k \max\{f(n), g(n)\}$$

ove $k = 2$.

Esercizio 3. *Ordinare le seguenti funzioni per ordine di crescita asintotico non decrescente, ove k sia una costante positiva comune. Ve ne sono alcune che presentano lo stesso ordine di crescita?* $f(n) = n^k$, $f(n) = (n+5)^k$, $f(n) = \binom{n}{k}$, $f(n) = n^{\left(k-\frac{1}{k}\right)} \log^k n^k$, $f(n) = 2^{k \log_2 n}$.

Svolgimento Esercizio 3

Abbiamo

$$\begin{aligned} f_1(n) &= n^k, \\ f_2(n) &= (n+5)^k, \\ f_3(n) &= \binom{n}{k}, \\ f_4(n) &= n^{\left(k-\frac{1}{k}\right)} \log^k n^k, \\ f_5(n) &= 2^{k \log_2 n} = n^k. \end{aligned}$$

Abbiamo dunque $f_1(n) = f_5(n)$. Inoltre,

$$\lim_{n\to\infty} \frac{f_2(n)}{f_1(n)} = \lim_{n\to\infty} \frac{(n+5)^k}{n^k} = \left(\lim_{n\to\infty} \frac{n+5}{n}\right)^k = 1^k = 1$$

Segue poi

$$\begin{aligned} \lim_{n\to\infty} \frac{f_3(n)}{f_1(n)} &= \lim_{n\to\infty} \frac{n(n-1)\ldots(n-k+1)}{k!\, n^k} \\ &= \frac{1}{k!} \lim_{n\to\infty} \frac{n}{n} \frac{n-1}{n} \ldots \frac{n-k+1}{n} \\ &= \frac{1}{k!} \lim_{n\to\infty} \frac{n}{n} \lim_{n\to\infty} \frac{n-1}{n} \lim_{n\to\infty} \ldots \frac{n-k+1}{n} \\ &= \frac{1}{k!} \cdot 1 \cdot 1 \ldots \cdot 1 = \frac{1}{k!} \end{aligned}$$

Infine, notando che $(\log n^k)^k = (k \log n)^k = k^k \log^k n$, otteniamo

$$\begin{aligned} \lim_{n\to\infty} \frac{f_4(n)}{f_1(n)} &= \lim_{n\to\infty} \frac{k^k n^{\left(k-\frac{1}{k}\right)} \log^k n}{n^k} \\ &= k^k \lim_{n\to\infty} \frac{\log^k n}{n^{1/k}} = 0 \end{aligned}$$

poiché abbiamo visto che una qualunque potenza positiva del logaritmo cresce meno di una qualunque potenza positiva di n. Ricordiamo per completezza la dimostrazione di questo fatto: sia $\beta > 0$, allora

$$\lim_{n\to\infty} \frac{\ln n}{n^\beta} = \lim_{n\to\infty} \frac{D(\ln n)}{D(n^\beta)} = \lim_{n\to\infty} \frac{1}{n} \frac{1}{n^{\beta-1}} = \lim_{n\to\infty} n^{-\beta} = 0.$$

Se $\alpha > 0$,

$$\lim_{n\to\infty} \frac{\log^\alpha n}{n^\beta} = \lim_{n\to\infty} \left(\frac{\log n}{n^{\beta/\alpha}}\right)^\alpha = \left(\lim_{n\to\infty} \frac{\log n}{n^{\beta/\alpha}}\right)^\alpha = 0^\alpha = 0,$$

dove abbiamo usato il fatto che β/α è pur sempre una costante positiva cui si applica il fatto appena dimostrato. In definitiva, come ordine di crescita abbiamo

$$f_4(n) < f_1(n) = f_2(n) = f_3(n) = f_5(n).$$

Esercizio 4. *Si dimostri che* $\sum_{k=1}^{n} \left(\frac{1}{k} + \frac{1}{k+1}\right) = \Theta(\log n)$.

Svolgimento Esercizio 4
Sia

$$H(n) = \sum_{k=1}^{n} \frac{1}{k}.$$

$H(n)$ è detto *ennesimo numero armonico.* Abbiamo

$$\begin{aligned} S(n) = \sum_{k=1}^{n} \left(\frac{1}{k} + \frac{1}{k+1}\right) &= \sum_{k=1}^{n} \frac{1}{k} + \sum_{k=1}^{n} \frac{1}{k+1} \\ &= \sum_{k=1}^{n} \frac{1}{k} + \sum_{h=2}^{n+1} \frac{1}{h} \\ &= \sum_{k=1}^{n} \frac{1}{k} + \left(\sum_{h=1}^{n+1} \frac{1}{h}\right) - 1 \\ &= H(n) + H(n+1) - 1. \end{aligned}$$

Per trovare un limite inferiore a $H(n)$, notiamo che

$$\begin{aligned} H(n) &= \sum_{k=1}^{n} \frac{1}{k} \\ &= \int_1^{n+1} \frac{1}{\lfloor x \rfloor}\, dx \\ &\geq \int_1^{n+1} \frac{1}{x}\, dx \\ &= [\ln x]_1^{n+1} \\ &= \ln(n+1). \end{aligned}$$

È consigliato farsi il disegno con l"'istogramma" e la funzione che lo approssima per capire meglio le manipolazioni effettuate. L'istogramma viene poi buono anche per l'altra metà della dimostrazione. Analogamente, infatti,

$$\begin{aligned} H(n) &= \sum_{k=1}^{n} \frac{1}{k} \\ &= 1 + \int_2^{n+1} \frac{1}{\lfloor x \rfloor}\, dx \\ &\leq 1 + \int_2^{n+1} \frac{1}{x-1}\, dx \\ &= 1 + \int_1^{n} \frac{1}{x}\, dx \\ &= 1 + [\ln x]_1^{n} \\ &= 1 + \ln n. \end{aligned}$$

Quindi, avendo $\ln(n+1) \leq H(n) \leq 1 + \ln n$, otteniamo

$$\ln(n+1) + \ln(n+2) - 1 \leq H(n) + H(n+1) - 1 = S(n) \leq 1 + \ln n + \ln(n+1).$$

Essendo

$$\lim_{n\to\infty} \frac{\ln(n+1)+\ln(n+2)-1}{\ln n} = \lim_{n\to\infty} \frac{1+\ln n+\ln(n+1)}{\ln n} = 1,$$

otteniamo che $S(n) \in \Theta(\log n)$. Notare che la base del logaritmo diventa irrilevante nel passaggio alla notazione $\Theta()$.

Approfondimento. La tesi $\ln(n+1) \le H(n) \le 1+\ln n$, si può anche dimostrare per induzione - al solito, con un tantino di senno di poi.
Per $n = 1$, infatti, essa vale: $\ln 2 \le H(1) = 1 \le 1 + \ln 1$.
Supponendola vera per un certo $n \ge 1$, sommiamo membro a membro $1/(n+1)$ ad ottenere

$$\ln(n+1) + \frac{1}{n+1} \le H(n) + \frac{1}{n+1} = H(n+1) \le (1+\ln n) + \frac{1}{n+1}.$$

Se riuscissimo dunque a mostrare che, per ogni $n \ge 1$,

$$\ln(n+2) \le \ln(n+1) + \frac{1}{n+1}$$

e

$$1 + \ln n + \frac{1}{n+1} \le 1 + \ln(n+1),$$

saremmo giunti alla tesi.
Le due richieste di sopra si possono riscrivere, effettuando anche uno spostamento di indici nella prima, come segue:

$$\begin{aligned} \forall n \ge 2, \quad \ln(n+1) - \ln n &\le \frac{1}{n}, \\ \forall n \ge 1, \quad \ln(n+1) - \ln n &\ge \frac{1}{n+1}. \end{aligned}$$

I due fatti si possono dimostrare in un sol colpo ricordando il teorema di Lagrange, per il quale data una funzione reale derivabile in $[x, y]$,

$$\frac{h(y)-h(x)}{y-x} = f'(\xi),$$

ove $\xi \in (x, y)$. Nel nostro caso

$$\ln(n+1) - \ln n = \frac{\ln(n+1)-\ln n}{(n+1)-n} = \frac{1}{\xi},$$

ove $\xi \in (n, n+1)$. Da questo fatto le due tesi seguono immediatamente.

Si possono limitare i numeri armonici anche senza fare riferimento al calcolo integrale. Sia $T = \lfloor \log_2 n \rfloor$. Notiamo che

$$\sum_{k=1}^{2^T} \frac{1}{k} \le H(n) \le \sum_{k=1}^{2^{T+1}-1} \frac{1}{k}.$$

Trattiamo separatamente, per comodità, limite inferiore e superiore. Per il primo, notiamo che

$$\begin{aligned}
H(n) &\geq \sum_{k=1}^{2^T} \frac{1}{k} \\
&> \sum_{k=1}^{2^T-1} \frac{1}{k} \\
&= \sum_{k=2^0}^{2^1-1} \frac{1}{k} + \sum_{k=2^1}^{2^2-1} \frac{1}{k} + \ldots + \sum_{k=2^{T-2}}^{2^{T-1}-1} \frac{1}{k} + \sum_{k=2^{T-1}}^{2^T-1} \frac{1}{k} \\
&= \sum_{t=0}^{T-1} \left(\sum_{k=2^{t-1}}^{2^t-1} \frac{1}{k} \right) \\
&> \sum_{t=0}^{T-1} \left(\sum_{k=2^{t-1}}^{2^t-1} \frac{1}{2^t} \right) \\
&= \sum_{t=0}^{T-1} 2^{t-1} \frac{1}{2^t} \\
&= \sum_{t=0}^{T-1} \frac{1}{2} \\
&= \frac{T}{2} = \frac{\lfloor \log_2 n \rfloor}{2}.
\end{aligned}$$

Analogamente, per il limite superiore,

$$\begin{aligned}
H(n) &\leq \sum_{k=1}^{2^{T+1}-1} \frac{1}{k} \\
&= \sum_{k=2^0}^{2^1-1} \frac{1}{k} + \sum_{k=2^1}^{2^2-1} \frac{1}{k} + \ldots + \sum_{k=2^T}^{2^{T+1}-1} \frac{1}{k} \\
&= \sum_{t=0}^{T} \sum_{k=2^t}^{2^{t+1}-1} \frac{1}{k} \\
&\leq \sum_{t=0}^{T} \sum_{k=2^t}^{2^{t+1}-1} \frac{1}{2^t} \\
&= \sum_{t=0}^{T} 2^t \frac{1}{2^t} \\
&= \sum_{t=0}^{T} 1 \\
&= T + 1 = \lfloor \log_2 n \rfloor + 1
\end{aligned}$$

Abbiamo cosí mostrato, con mezzi del tutto elementari, che

$$\frac{\lfloor \log_2 n \rfloor}{2} < H(n) \leq \lfloor \log_2 n \rfloor + 1.$$

Esercizio 5. *Dato un intero $a > 0$, allo scopo di calcolare la potenza a^n, usiamo la seguente procedura iterativa:*

```
#include<iostream.h>

int main() {
  int a; cout << "Dammi a: "; cin >> a; cout << endl;
  int n; cout << "Dammi n: "; cin >> n; cout << endl;
  long long int P[n +1];
  P[0] = 1;
  for(int k=1; k<=n; k++)
    P[k] = a*P[k-1];
  cout << "P[" << n << "] = "  << P[n] << endl;
}
```

Si stabilisca l'ordine di crescita del tempo di calcolo della procedura proposta. Si stabilisca l'ordine di crescita della memoria impiegata dalla procedura proposta.

Svolgimento Esercizio 5

La ricorrenza che calcola, ad esempio, il numero di moltiplicazioni, è:

$$T(0) = 0,$$
$$T(n) = T(n-1) + 1$$

che ha l'ovvia soluzione $T(n) = n$.
Per la memoria, siccome riserviamo per il calcolo di a^n un array di dimensione $n+1$, abbiamo $M(n) = n+1$.

Esercizio 6. *Il professor Gonzalez, propone di utilizzare la seguente procedura ricorsiva allo scopo di calcolare la medesima funzione:*

```
#include<iostream.h>

long long int P(int a, int n) {
  if (n==0) return 1;
  if (n==1) return a;
  return P(a, n/2)* P(a, n- (n/2));
}

int main() {
  int a; cout << "Dammi a: "; cin >> a; cout << endl;
  int n; cout << "Dammi n: "; cin >> n; cout << endl;
  cout << "P[" << n << "] = "  << P(a, n) << endl;
}
```

Si stabilisca l'ordine asintotico di crescita per il tempo di calcolo della procedura proposta dal professor Gonzalez. Si stabilisca l'ordine di crescita della memoria impiegata dalla procedura di Gonzalez.

Svolgimento Esercizio 6

Conoscendo come funziona la divisione intera del `C`, vediamo che il procedimento di calcolo adottato è il seguente:

$$a^0 = 1,$$
$$a^1 = a,$$
$$a^n = a^{\lfloor \frac{n}{2} \rfloor} a^{\lceil \frac{n}{2} \rceil}.$$

A livello di moltiplicazioni abbiamo dunque

$$T(0) = T(1) = 0;$$

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1.$$

Ovviamente, possiamo applicare il Master Theorem ove $a = b = 2$, $f(n) = 1 \in O(n^{\log_b a - \epsilon})$, dove ad esempio $\epsilon = 1/2$, per ottenere $T(n) \in \Theta(n^{\log_b a}) = \Theta(n)$. Chi, giustamente, non s'accontenta, cercherà di arrivare a una soluzione precisa, data la facilità del caso. Procedendo a mano col seguente programmino:

```
#include <iostream.h>

int T[10];

main() {
  int i;
  T[0]= T[1]= 0;
  for (i=2; i<10; i++) {
    T[i]= T[i/2] + T[i - i/2] + 1;
    cout<<i<<"  "<<T[i]<<endl;
  }
}
```

constaterà che per $n \geq 1$ abbiamo $T(n) = n - 1$. Proviamo a dimostrarlo per induzione. Base, $T(0) = T(1) = 1$. Serve trattare il caso $n = 1$ entro la base poiché altrimenti otterremmo $T(1) = T(0) + T(1) + 1$ che non ha senso, non da ultimo, perché descrive la ricorsione di un programma che non termina. Usiamo dunque l'ipotesi induttiva $(T(0) = 0) \wedge (\forall k < n, T(k) = k - 1)$. Abbiamo, per $n \geq 2$,

$$\begin{aligned} T(n) &= T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1 \\ &= (\lfloor \frac{n}{2} \rfloor - 1) + (\lceil \frac{n}{2} \rceil - 1) + 1 \\ &= (\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil) - 1 = n - 1 \end{aligned}$$

dove abbiamo usato nell'ordine: la relazione di ricorrenza; l'ipotesi induttiva, ricordando che per $n \geq 2$ abbiamo sempre $\lfloor \frac{n}{2} \rfloor > 0$, per cui ci possiamo disinteressare al caso base $n = 0$; infine, il fatto che l'arrotondamento superiore e quello inferiore della metà di un numero sommano al numero stesso (distinguere i due casi pari e dispari per vederlo facilmente). Abbiamo anche, ovviamente, sfruttato il fatto che per $n \geq 2, \lceil n/2 \rceil < n$, ed è quindi lecito applicarvi l'ipotesi induttiva. La dimostrazione è quindi completa.

Per quanto riguarda la mamoria, notare che essa è proporzionale al numero di attivazioni della procedura che coesistono sullo stack, il quale a sua volta è al piú pari alla massimo numero di nodi su un ramo dell'albero di ricorsione (radice e foglia comprese). Tale distanza è governata dalla seguente ricorrenza:

$$D(1) = 1$$

$$D(n) = D(\lceil \frac{n}{2} \rceil) + 1.$$

Qui, ovviamente, gli uni stanno per unità di memoria consumate dalla singola attivazione. Per il Master theorem, con $a = 1, b = 2, f(n) = 1$, otteniamo che $D(n) \in \Theta(\log n)$. La soluzione esatta della ricorrenza è poi stata ricavata in una delle precedenti raccolte di esercizi.

Notare che $a = 1$, ben diversamente dalla ricorrenza che esprime il tempo di calcolo; questo è legato al fatto che, ad ogni istante, *al piú una* attivazione di procedura chiamata consiste con l'attivazione della procedura chiamante. Detto in altre parole, i comportamenti di $D(n)$ e di $T(n)$ differiscono cosí profondamente perché si può scrivere due volte nella stessa locazione di memoria, ma non si può vivere due volte lo stesso istante...

Complemento 6+ *Dire come il prof. Gonzalez possa, con un semplice accorgimento, ridurre il tempo di calcolo della sua procedura tanto da meritarsi l'appellativo di Speedy. Qualeè il nuovo tempo di calcolo?*

Svolgimento Complemento 6+

Si vede che la procedura può facilmente evitare di fare due chiamate ricorsive. Infatti, possiamo osservare che per n pari, $a^n = (a^{\lfloor \frac{n}{2} \rfloor})^2$; per n dispari, $a^n = a \cdot (a^{\lfloor \frac{n}{2} \rfloor})^2$. Questo suggerisce la scrittura della seguente funzione:

```
int power(int a, int n) {
  int tmp;
  if (n==0)
    return 1;
  if (n==1)
    return a;
  tmp= power(a, n/2);
  return (n % 2) ? (a*(tmp*tmp)) : (tmp*tmp);
}
```

Applicando il Master Theorem ove $a = 1, b = 2, f(n) \in \{1, 2\}$, otteniamo $T(n) \in \Theta(\log_2 n)$.

Approfondimento. Qui, il computo esatto delle moltiplicazioni segue la legge

$$T(0) = T(1) = 0;$$

$$T(2k) = T(k) + 1,$$

$$T(2k+1) = T(k) + 2,$$

la cui tabulazione, e.g. con il programmino

```
main() {
  int i;
  T[0]= T[1]= 0;
  for (i=2; i<20; i++) {
    T[i]= (i%2) ? (T[i/2]+2) : (T[i/2]+1);
    cout<<i<<"  "<<T[i]<<endl;
  }
}
```

dà i seguenti risultati:

```
  n =  2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
T(n)=  1   2   2   3   3   4   3   4   4   5   4   5   5   6   4
```

La decifrazione non è immediata come nei casi precedenti; ci si rende conto che tuttavia $T(n)$ è legato al numero $U(n)$ di uno che compaiono nello sviluppo binario di n, oltre che alla lunghezza $L(n)$ di detto sviluppo. Nella procedura *power* di sopra, infatti, la moltiplicazione per a viene eseguita se e solo se n è dispari, ovvero, se la sua cifra meno significativa è uno. Ricorsivamente, ci interesseremo poi alla parità di $\lfloor n/2 \rfloor$, e via dicendo. La cosa piú naturale è cercare una formuletta del tipo

$$M(n) = \alpha L(n) + \beta U(n) + \gamma.$$

Scegliamo tre valori a caso, e.g. $n = 6$, $n = 12$ e $n = 16$. Lo sviluppo di 6 è 110, quello di 12 è 1100, quello di 16 è 10000; vogliamo quindi

$$3\alpha + 2\beta + \gamma = 3$$

$$4\alpha + 2\beta + \gamma = 4$$

$$5\alpha + \beta + \gamma = 4$$

da cui $\alpha = 1, \beta = 1, \gamma = -2$. Proviamo a vedere se la formula funziona:

```
n     L(n)     U(n)    L(n)+U(n)-2

0       1        0         -1
1       1        1          0
2       2        1          1
3       2        2          2
4       3        1          2
5       3        2          3
6       3        2          3
7       3        3          4
8       4        1          3
```

Sembra che per $n > 0$ funzioni. Dimostriamolo dunque per induzione. Innanzitutto, $0 = T(1) = L(1) + U(1) - 2 = 1 + 1 - 2$. Supponiamo che per $k \in [1, n-1]$ sia $T(k) = L(k) + U(k) - 2$. Sia n pari. Allora $n/2$ ha un bit in meno, e lo stesso numero di uni.
Abbiamo

$$T(n) = T(n/2) + 1 = [L(n/2) + U(n/2) - 2] + 1 = [(L(n) - 1) + U(n) - 2] + 1 = L(n) + U(n) - 2.$$

Sia n dispari. Allora $\lfloor n/2 \rfloor$ ha un bit in meno, e anche un uno in meno.
Abbiamo

$$T(n) = T(n/2) + 2 = [L(n/2) + U(n/2) - 2] + 1 = [(L(n) - 1) + (U(n) - 1) - 2] + 2 = L(n) + U(n) - 2.$$

La dimostrazione è dunque completa. Sappiamo poi che, per $n > 0$, $L(n) = \lfloor \log_2 n \rfloor + 1$, e che ovviamente $1 \leq U(n) \leq L(n)$ (casi estremi: solo la cifra piú significativa dello sviluppo è uno, oppure tutte le cifre sono uno). Abbiamo quindi

$$\lfloor \log_2 n \rfloor \leq T(n) \leq 2 \lfloor \log_2 n \rfloor ,$$

dove il limite inferiore è toccato quando $n = 2^t$, quello superiore quando $n = 2^t - 1$.