

Università degli Studi di Camerino – Laurea in Informatica  
Prima Prova Parziale del corso di **Algoritmi e Strutture Dati**

Docente: Maria Rita Di Berardini

19 dicembre 2007

Nome:

Cognome:

N.Matricola:

Note: scrivere Nome, Cognome e N.Matricola su ogni foglio consegnato

1. Come misuriamo l'efficienza di un'algoritmo?
2. Quali sono i vantaggi derivanti dall'analisi di algoritmi in termine di tempo di esecuzione
3. Quali sono, in generale, gli algoritmi più efficienti, quelli ricorsivi o quelli iterativi?
4. Quale è lo scopo della notazione asintotica?
5. Fornire la definizione *formale* delle notazioni  $O$ ,  $\Omega$  e  $\Theta$ .

Siano  $f, g : \mathbf{N} \longrightarrow \mathbf{R}^{\geq 0}$ . Diciamo che:

- $f(n) = O(g(n))$  se esistono delle costanti positive  $c, n_0$  tali che  $0 \leq f(n) \leq cg(n)$  per ogni  $n \geq n_0$ ;
  - $f(n) = \Omega(g(n))$  se esistono delle costanti positive  $c, n_0$  tali che  $0 \leq cg(n) \leq f(n)$  per ogni  $n \geq n_0$ ;
  - $f(n) = \Theta(g(n))$  se esistono delle costanti positive  $c_1, c_2, n_0$  tali che  $0 \leq c_1g(n) \leq f(n) \leq c_2g(n)$  per ogni  $n \geq n_0$ .
6. Vero o falso (fornire una breve spiegazione per ogni risposta fornita):
    - (i)  $n = \Omega(\log n)$ . Vero.  
È possibile dimostrare per induzione su  $n$  che  $\log n \leq n$  per ogni  $n \geq 1$ .
    - (ii)  $3n^2 + 12n + 5 = \Theta(n^2)$ . Vero.  
Per ogni  $n \geq n_0 = 1$ ,  $n^2 \leq 3n^2 + 12n + 5$  – e quindi  $3n^2 + 12n + 5 = \Omega(n^2)$ .  
Inoltre,  $3n^2 + 12n + 5 \leq 3n^2 + 12n^2 + 5n^2 = 20n^2$ , per ogni  $n \geq n_0 = 1$  – e quindi  $3n^2 + 12n + 5 = O(n^2)$ .
    - (iii)  $3^n = \Theta(4^n)$ . Falso, infatti non è vero che  $3^n = \Omega(4^n)$ . Lo dimostriamo per assurdo.  
Assumiamo che  $3^n = \Omega(4^n)$ . Allora esistono delle costanti positive  $c, n_0$  tali che  $c4^n \leq 3^n$ , ossia tali che  $c \leq (3/4)^n$  per ogni  $n \geq n_0$ . Impossibile, perchè,  $c$  è una costante mentre il valore della funzione  $(3/4)^n$  decresce costantemente al variare di  $n$ .
  7. Assumendo  $T(1) = 1$ , risolvere le seguenti ricorrenze (usare il metodo indicato di fianco)

(i)  $T(n) = 2T(\frac{n}{4}) + n^2$  (metodo iterativo).

$$\begin{aligned}
 T(n) &= n^2 + 2T(n/4) \\
 &= n^2 + 2(n/4)^2 + 4T(n/16) \\
 &= n^2 + 2(n/4)^2 + 4(n/16)^2 + 8T(n/64) \\
 &= n^2 + 2(n/4)^2 + 4(n/16)^2 + 8(n/64)^2 + 16T(n/256) \\
 &\dots \\
 &= \sum_{i=0}^{k-1} 2^i (n/4^i)^2 + 2^k T(n/4^k)
 \end{aligned}$$

Ci fermiamo quando  $n/4^k = 1$  e quindi quando  $k = \log_4 n$ . Inoltre

$$2^i \left( \frac{n}{4^i} \right)^2 = 2^i \left( \frac{n^2}{(4^i)^2} \right) = 2^i \left( \frac{n^2}{(4^2)^i} \right) = 2^i \left( \frac{n^2}{16^i} \right) = \left( \frac{2}{16} \right)^i n^2 = \left( \frac{1}{8} \right)^i n^2$$

Allora

$$\begin{aligned}
 T(n) &= \sum_{i=0}^{\log_4 n - 1} \left( \frac{1}{8} \right)^i n^2 + 2^{\log_4 n} \cdot T(1) = n^2 \sum_{i=0}^{\log_4 n - 1} \left( \frac{1}{8} \right)^i + n^{\log_4 2} \\
 &= n^2 \cdot \frac{1 - (1/8)^{\log_4 n}}{1 - (1/8)} + \sqrt{n} = (7/8)n^2 \cdot \left( 1 - \frac{1}{8^{\log_4 n}} \right) + \sqrt{n} \\
 &= (7/8)n^2 \cdot \left( 1 - \frac{1}{n^{\log_4 8}} \right) + \sqrt{n} = (7/8)n^2 \cdot \left( 1 - \frac{1}{n^{3/2}} \right) + \sqrt{n} \\
 &= (7/8)n^2 - (7/8)\sqrt{n} + \sqrt{n} = (7/8)n^2 + (1/8)\sqrt{n} = \Theta(n^2)
 \end{aligned}$$

(ii)  $T(n) = 2T(\frac{n}{4}) + n$  (metodo dell'albero di ricorsione). Procediamo alla costruzione dell'albero di ricorsione

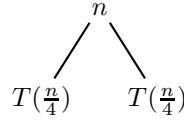


Figura 1: Albero di ricorsione – livello 1

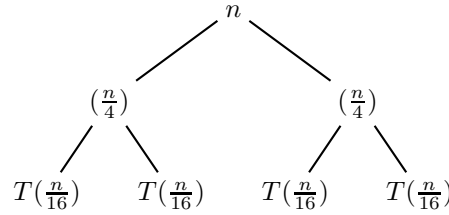


Figura 2: Albero di ricorsione – livello 2

L'albero completo (vedi Figura 3) ha  $k+1$  livelli con  $k = \log_4 n$ ; l'ultimo livello corrisponde ad una chiamata di  $T(n/4^k)$  con  $n/4^k = 1$  e quindi  $k = \log_4 n$ .

Sia  $i$  un generico livello dell'albero (con  $i = 0, \dots, \log_4 n$ ). Questo livello ha  $2^i$  nodi (il livello 0 ha un nodo e, ogni volta che scendiamo di livello, moltiplichiamo per due il numero di nodi del livello precedente) ciascun dei quali ha un costo pari a  $n/4^i$ . Il costo complessivo del livello  $i$  è  $2^i \cdot n/4^i = (1/2)^i n$ . Allora

$$T(n) = \sum_{i=0}^{\log_4 n} (1/2)^i n = n \sum_{i=0}^{\log_4 n} (1/2)^i = n \cdot 2 \left( 1 - \frac{1}{2^{\log_4 n + 1}} \right) = 2n - 2\sqrt{n} = \Theta(n)$$

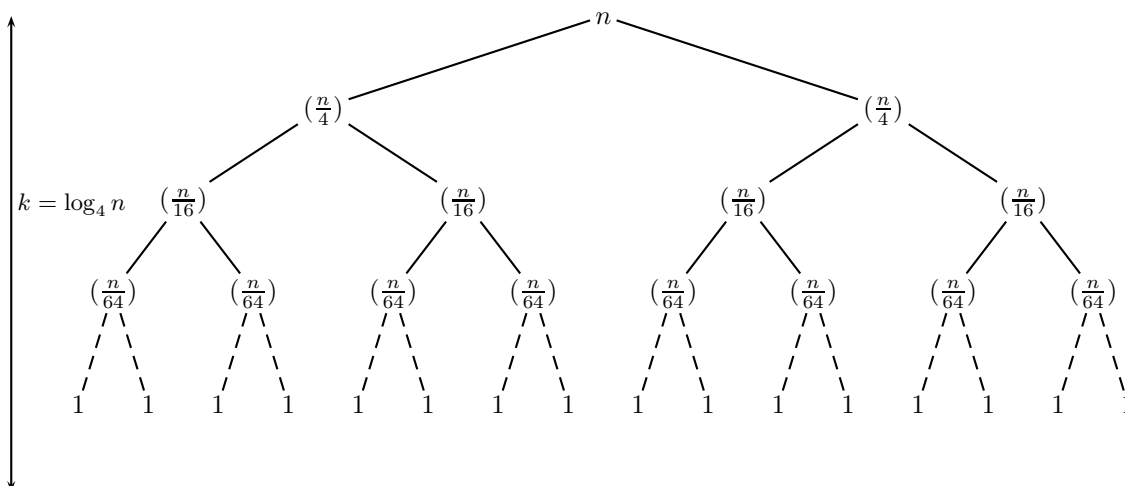


Figura 3: Albero di ricorsione (completo)

(iii)  $T(n) = 2T(\frac{n}{4}) + n^3$  (teorema del master).

$a = 2$  e  $b = 4$ ; inoltre  $\log_b a = \log_4 2 = 1/2$ . Allora  $f(n) = n^3 = \Omega(n^{\log_b a + \epsilon})$  con  $\epsilon > 0$  e  $\log_b a + \epsilon = 1/2 + \epsilon \leq 3$  (e quindi  $0 < \epsilon \leq 5/2$ ). Inoltre,  $af(n/b) = 2(n/4)^3 = 2(n^3/4^3) = (2/64)n^3 = (1/32)n^3 \leq cf(n)$  per un qualche  $(1/32) \leq c < 1$ . Terzo caso del teorema del master. Possiamo concludere che  $T(n) = \Theta(f(n)) = \Theta(n^3)$

8. È possibile realizzare il tipo di dato astratto *coda* mediante una struttura collegata come segue. Usiamo una lista collegata di records del tipo  $(value, next)$  per memorizzare tutti gli elementi della coda, un riferimento  $f$  all'elemento testa alla coda, ed un riferimento  $\ell$  all'elemento in fondo alla coda. Si discuta il costo di ciascuna delle operazioni supportate dal tipo astratto coda (`isEmpty()`, `enqueue()`, `dequeue()`, `first()`).
9. Assumete che un docente voglia utilizzare il numero di matricola (5 cifre decimali) come chiave in una tabella per memorizzare i suoi 200 studenti. Quale è il fattore di carico della tabella?
10. Discutere i vantaggi e gli svantaggi (in termini di occupazione di memoria e costo delle singole operazioni) della gestione di un tipo di dato dizionario mediante tabella hash e mediante tabella ad indirizzamento diretto?
11. Dare una definizione del fenomeno conosciuto come *agglomerazione primaria* e discutere il suo impatto sul numero medio di accessi per una ricerca senza successo. Indicare (almeno) un metodo di scansione che consente di eliminare questo problema.
12. Cos'è un *max-heap*? Quale è la caratteristica di un max-heap che rende la sua altezza logaritmica nel numero dei nodi? Quali sono le implicazioni del fatto che  $h = O(\log n)$ , dove  $h$  ed  $n$  sono rispettivamente l'altezza ed il numero di nodi di un max-heap?
13. Si consideri la seguente sequenza di numeri

34 12 36 19 16 24 17 27 25 15

Assumendo che siano memorizzati in un array secondo la notazione posizionale, discutere motivando la risposta se sequenza rappresenta un *max-heap*. Nel caso in cui la sequenza non sia un max-heap, si applichi la procedura BUILD-MAX-HEAP per costruire un max-heap contenente gli

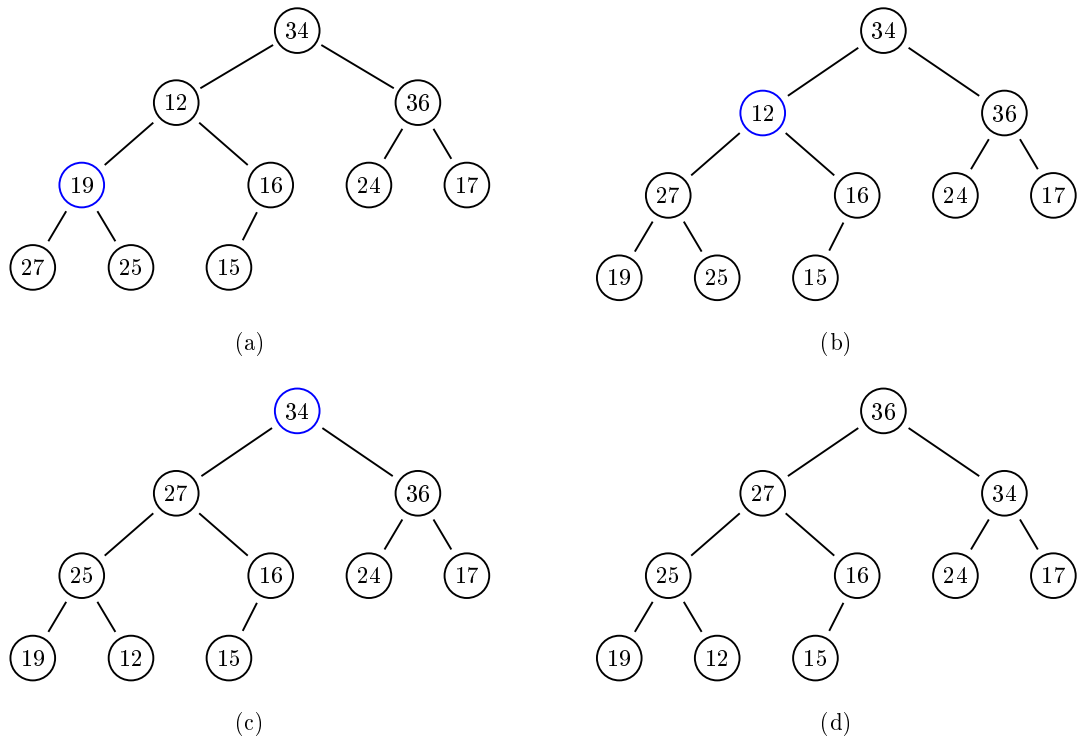


Figura 4: Creazione dello heap (i nodi in blue rappresentano le radici dei sottoalberi da ri-heapizzare)

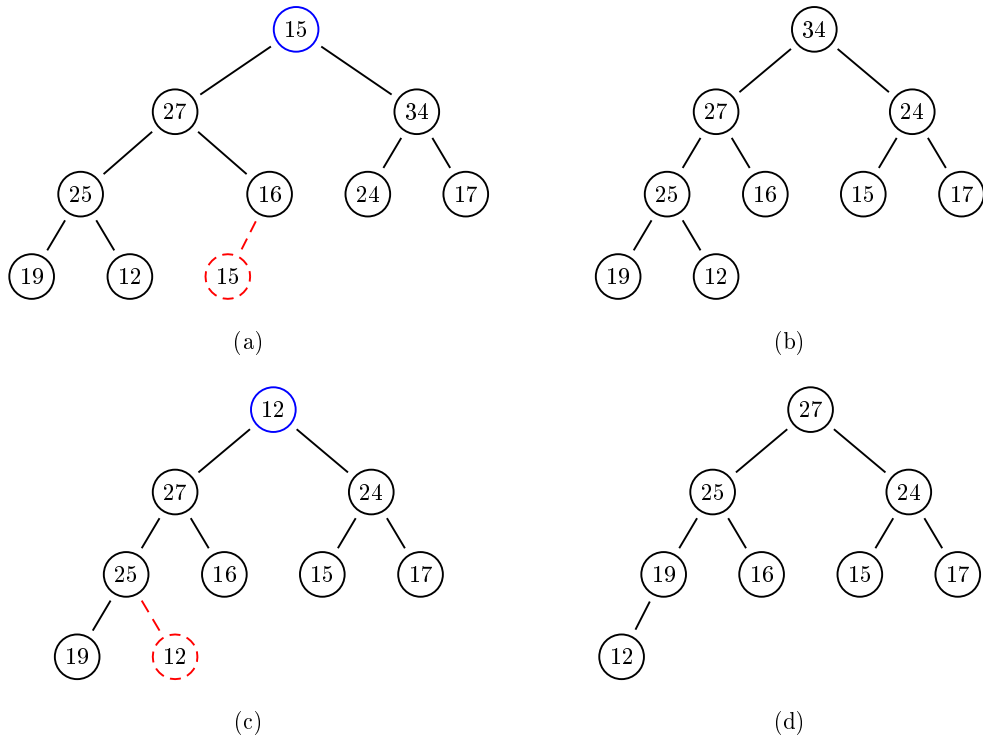


Figura 5: Eliminazione di 36 e 34 (i nodi in blue rappresentano le radici dei sottoalberi da ri-heapizzare, quelli in rosso rappresentano i nodi da eliminare)

stessi elementi. Si illustri lo stato del max-heap così ottenuto dopo aver estratto, nell'ordine, gli elementi 36 e 34.

14. Quale è il numero di *nodi interni* (che non sono né la root né delle foglie) di un albero binario completo di altezza  $h \geq 1$ ? Fornire una prova per induzione su  $h$ .

Sappiamo che:

- (a) il numero di nodi di un albero binario completo di altezza  $h$  è  $n = 2^{h+1} - 1$ ,
- (b) il numero di foglie è  $2^h$

Otteniamo il numero di nodi interni sottraendo ad  $n$  il numero delle foglie e poi uno (poiché la root non è un nodo interno). Allora il numero di nodi interni è

$$n_i = (2^{h+1} - 1) - 2^h - 1 = 2^h(2 - 1) - 2 = 2^h - 2$$

Lo dimostriamo per induzione su  $h$

- Caso base  $h = 1$ . Un albero completo di altezza 1 non ha nodi interni. Inoltre  $2^1 - 2 = 0$
- Sia  $T$  un albero completo di altezza  $h > 1$ . Il numero dei nodi interni di  $T$  è pari al numero dei nodi interni del sottoalbero sinistro  $T_\ell$  più il numero dei nodi interni del sottoalbero destro  $T_r$ , più due (le root del sottoalbero sinistro e destro sono dei nodi interni di  $T$  ma non di  $T_\ell$  e  $T_r$ ). Possiamo concludere che  $n_i = n_i(T_\ell) + n_i(T_r) + 2$ . Per ipotesi induttiva  $n_i(T_\ell) = n_i(T_r) = 2^{h-1} - 2$  (poiché sia  $n_i(T_\ell)$   $n_i(T_r)$  sono degli alberi completi di altezza  $h - 1$ ). Allora:

$$n_i = n_i(T_\ell) + n_i(T_r) + 2 = 2(2^{h-1} - 2) + 2 = 2^h - 4 + 2 = 2^h - 2$$

15. Dare una definizione di *albero binario di ricerca* (BST). Se un insieme dinamico  $S$  viene rappresentato mediante un BST come è possibile identificare l'elemento minimo e massimo in  $S$ . Quale è il costo di queste operazioni.
16. Quale è il risultato dell'applicazione della procedura **Tree-Walk** ad un qualsiasi BST, ossia quale è l'output di una chiamata di **Tree-Walk**(*root*)

**Tree-Walk**( $x$ )

▷  $x$  è un generico nodo dell'albero

**if**  $x \neq \text{NIL}$

**then**

**Tree-Walk**(*right*[ $x$ ])

**stampa**(*key*[ $x$ ])

**Tree-Walk**(*left*[ $x$ ])

Dare una misura della complessità di questo algoritmo.

*L'algoritmo stampa i valori associati ai nodi dell'albero ordinati in maniera crescente. La sua complessità è  $\Theta(n)$ .*

17. Si consideri l'insieme di numeri

$$A = \{34, 12, 36, 19, 16, 24, 17, 27, 25, 15\}$$

ed si illustri il comportamento delle procedura di **MergeSort**( $A, 1, n$ ), dove  $n = 10 = |A|$ . Fornire la ricorrenza che descrive il costo del MergeSort in funzione del numero di elementi dell'insieme da ordinare.

18. Si consideri l'algoritmo di ordinamento **QuickSort** illustrato di seguito. Si discuta il ruolo della procedura **Partition** e in quale maniera questa procedura può influenzare la complessità dell'algoritmo. In particolare si identifichino il caso peggiore ed il caso ottimo per il **QuickSort**.

```
QuickSort( $A, p, r$ )  
  if  $p < r$   
    then  $q \leftarrow \mathbf{Partition}(A, p, r)$   
        QuickSort( $A, p, q - 1$ )  
        QuickSort( $A, q + 1, r$ )
```

# 1 Altri esercizi

## 1.1 Problemi su alberi

**Esercizio 1.1** Dato un albero rappresentato con puntatori al primo figlio/fratello scrivere due algoritmi per calcolare il numero totale di nodi dell'albero ed il numero delle sue foglie

**Leaf( $r$ )**

```
1  ▷ restituisce true se  $r$  è una foglia, false altrimenti
2  if  $r = \text{NIL}$ 
3      then return false
4  return ( $r.\text{son} = \text{NIL}$ )
```

**ContaNodi( $r$ )**

```
1  if  $r = \text{NIL}$ 
2      then return 0
1  if Leaf( $r$ )
2      then return 1
4   $p \leftarrow r.\text{son}$ 
4   $x \leftarrow 1$ 
5  while  $p \neq \text{NIL}$ 
6  do
7       $x \leftarrow x + \text{ContaNodi}(p)$ 
8       $p \leftarrow p.\text{brother}$ 
9  return  $x$ 
```

**ContaFoglie( $r$ )**

```
1  if  $r = \text{NIL}$ 
2      then return 0
1  if Leaf( $r$ )
2      then return 1
4   $p \leftarrow r.\text{son}$ 
4   $x \leftarrow 0$ 
5  while  $p \neq \text{NIL}$ 
6  do
7       $x \leftarrow x + \text{ContaFoglie}(p)$ 
8       $p \leftarrow p.\text{brother}$ 
9  return  $x$ 
```

## 1.2 Problemi su array

**Esercizio 1.2** Si supponga di avere in input un array  $A$  contenente  $n$  interi distinti e ordinato in maniera crescente, ma il cui contenuto è stato shiftato in avanti di  $k$  posizioni, con  $1 \leq k < n$ . Mostrare un algoritmo che trova l'elemento massimo sfruttando l'ordinamento (Soluzioni che impiegano in tempo  $\Theta(n)$  non sono accettate perchè hanno lo stesso comportamento asintotico di una ricerca sequenziale)

Si consideri il seguente array. Confrontiamo l'elemento centrale (quello in posizione,  $m = \lfloor n/2 \rfloor$  con il primo elemento. Se  $A[m]$  è maggiore del elemento in prima posizione, ci spostiamo a destra, altrimenti ci spostiamo a sinistra.

|    |    |    |    |    |    |   |    |    |    |
|----|----|----|----|----|----|---|----|----|----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7 | 8  | 9  | 10 |
| 20 | 25 | 32 | 43 | 54 | 61 | 5 | 13 | 16 | 18 |

**Search-Max( $A, i, j$ )**

```
1  if  $i = j$  return  $A[i]$ 
2   $m \leftarrow \lfloor (i + j)/2 \rfloor$ 
3  if  $A[m] > A[i]$ 
4      then Search-Max( $A, m + 1, j$ )
5      else Search-Max( $A, i, m - 1$ )
```

## 1.3 Distanza minima

Dati due interi  $x$  ed  $y$  definiamo la distanza tra  $x$  e  $y$  come  $d(x, y) = |x - y|$ . Sia  $A$  un vettore di interi. Si descriva un algoritmo per determinare i due elementi di  $A$  aventi distanza minima

```

MIN-DISTANCE( $A$ )
1   $sort(A)$ 
2   $min \leftarrow -\infty$ 
3   $k \leftarrow 0$ 
4  for  $i \leftarrow 1$  to  $length[A] - 1$ 
5      do if  $DISTANCE(A[i], A[i + 1]) < min$ 
6          then  $min \leftarrow DISTANCE(A[i], A[i + 1])$ 
7               $k \leftarrow i$ 
8  Stampa “Gli elementi aventi minima distanza sono  $A[k]$  e  $A[k + 1]$ ”

```

Qui,  $DISTANCE(x, y)$  è un algoritmo che, dati due interi  $x$  ed  $y$  restituisce la loro distanza.

**Esercizio 1.3** Sia  $A$  un insieme di  $n$  numeri reali ed  $x \in \mathbf{R}$ . Scrivere un algoritmo, la cui complessità nel caso peggiore è  $O(n \log n)$ , che restituisca vero solo se  $A$  contiene due elementi la cui somma è  $x$ .

```

SEARCH-SUM( $A, x$ )
1   $sort(A)$ 
2   $n \leftarrow length[A]$ 
3  for  $i \leftarrow 1$  to  $n$ 
4      do if  $BINARY-SEARCH(A, x - A[i])$ 
5          then return true
6  return false

```

## 1.4 Un algoritmo misterioso

Si consideri il seguente algoritmo:

```

Mystery( $A, i, j, k$ )
1  if  $i > j$  return 0
2  if  $A[i] = k$  return  $i$ 
3  if  $A[j] = k$  return  $j$ 
4  Mystery( $A, i + 1, j - 1, k$ )

```

- Cosa restituisce  $\mathbf{Mystery}(A, 1, length[A], k)$ ?
- Trovare un limite asintotico (il più stretto possibile) alla complessità dell'algoritmo

L'algoritmo cerca  $k$  nel vettore  $A$  confrontandolo, di volta in volta, con l'elemento in prima e ultima posizione. Se  $k$  appartiene ad  $A$  restituisce la sua posizione in  $A$ ; altrimenti restituisce zero. la complessità dell'algoritmo è descritto dalla seguente ricorrenza:

$$T(n) = \begin{cases} T(n-2) & \text{se } n > 2 \\ c & \text{altrimenti} \end{cases}$$

Risolviamo questa ricorrenza con il metodo iterativo:

$$\begin{aligned}
 T(n) &= c + T(n-2) \\
 &= c + c + T(n-4) \\
 &= c + c + c + T(n-6) \\
 &\quad \dots \\
 &= kc + T(n-2k)
 \end{aligned}$$



Se  $n$  è pari, terminiamo quando  $n - 2k = 2$  e quindi, quando  $k = (n - 2)/2$ . Allora,  $T(n) = c(n - 2)/2 + c = (c/2)n$ . Se  $n$  è dispari terminiamo quando  $n - 2k = 1$  e quindi, quando  $k = (n - 1)/2$ . In questo caso  $T(n) = c(n - 1)/2 + c = (c/2)n + (c/2)$ . Possiamo concludere che  $T(n) = \Theta(n)$ .

## 1.5 Calcolo della moda

*Scrivere un algoritmo per trovare la moda di un insieme di dati (ovvero il valore che compare più spesso). Nel caso in cui più valori possono essere definiti come moda, ritornare il valore numericamente più alto.*