Esercizi – Esami Passati

Anno Accademico 2022/2023

Dott. Staccone Simone

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Esercizio 1 - 26/01/2022

Si indichi un limite asintotico per la ricorrenza
$T(n) = 2\,T(n/8) + \sqrt{n}$.

Soluzione:
Provo ad applicare il Master Theorem.
→ Ip. $a \geq 1$ e $b \geq 2$ sono costanti intere, $c > 0$ e $\beta \geq 0$ sono costanti reali. ($\alpha = \log_b a$)

→ $a = 2$; $b = 8$; $c = 1$; $\beta = 1/2$; (Ipotesi verificate)

→ $\alpha = \log_8 2 = \log_2 2 \,/\, \log_2 8 = 1/3$; $\beta = 1/2$; Terzo caso del master theorem → $\alpha < \beta$
$\Theta(n^{\beta}) = \Theta(n^{1/2})$

Esercizio 1 - 24/02/2022

Si indichi un limite asintotico per la ricorrenza
$T(n) = 4\,T(n/2) + n^2$.

Soluzione:
Provo ad applicare il Master Theorem.
→ Ip. $a \geq 1$ e $b \geq 2$ sono costanti intere, $c > 0$ e $\beta \geq 0$ sono costanti reali. ($\alpha = \log_b a$)

→ $a = 4$; $b = 2$; $c = 1$; $\beta = 2$; (Ipotesi verificate)

→ $\alpha = \log_2 4 = 2$; $\beta = 2$; Secondo caso del master theorem → $\alpha < \beta$
$\Theta(n^{\alpha} \log n) = \Theta(n^2 \log n)$

Esercizio 1 - 15/06/2022

Si risolva la seguente ricorrenza per ottenere i migliori limiti asintotici per T(n) usando la notazione O(). $T(n) = T(n / 4\log_2(n)) + 2n$, $n \geq 1$, $T(1) = 1$. É possibile assumere che tutti i numeri possano essere arrotondati al più vicino intero.

Soluzione:
Forniamo un limite superiore ed uno inferiore per mostrare che il miglior limite asintotico per la ricorrenza è proprio $\Theta(n)$.

Un limite inferiore di $\Omega(n)$ segue dal secondo termine in T(n)
$T(n) > 2n > n$

Un limite superiore può essere raggiunto osservando che $4\log(n) > 4$ per $n > 1$, quindi $T(n / 4\log(n)) \leq T(n/4)$.

Quindi $T(n) \leq T(n/4) + 2n$

Applico il master theorem e ottengo $T(n) \leq \Theta(n)$.

Quindi $\Omega(n) \leq T(n) \leq \Theta(n) \rightarrow T(n) = \Theta(n)$

Esercizio 1 - 15/06/2022

Sia risolva la ricorrenza: $T(n) = 3\,T(\lceil n/2 \rceil) + cn$ con $T(1) = 4$.

Soluzione:

Innanzitutto notiamo che la parte intera superiore non altera il risultato in quanto possiamo sempre considerare un n di partenza che sia una potenza di 2 ,visto che siamo interessati all'andamento asintotico, e la stessa potenza di 2 la possiamo utilizzare per maggiorare il tempo di calcolo corrispondente ad n.

Applico il Master Theorem e vado a valutare il risultato supponendo $c > 1$:

$\alpha = \log_2 3$; $\beta = 1$; $\alpha > \beta$ allora siamo nel terzo caso del teorema

$T(n) = \Theta(n^{\log(3)})$

Esercizio 1 - 16/07/2021

Trovare limiti superiore e inferiore per la seguente equazione di ricorrenza, utilizzando il metodo di sostituzione:

$$T(n) = \begin{cases} T(n/2) + 2^n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Soluzione:
È facile osservare che un limite inferiore per T(n) è $\Omega(2^n)$, in quanto la componente non ricorsiva è pari a 2^n. Proviamo quindi a dimostrare che $T(n) = O(2^n)$

Caso base: $T(1) = 1 \leq 2c$, il che è vero per $c \geq ½$ (n = 1)
Ipotesi induttiva: $T(n') \leq 2^{n'}$ per ogni n' < n.
Passo induttivo:
$T(n) = T(n/2) + 2^n \leq c \cdot 2^{n/2} + 2^n \leq c \cdot 2^n$

Consideriamo ora la disequazione $c \cdot 2^{n/2} + 2^n \leq c \cdot 2^n$
$c*(2^n - 2^{n/2}) \geq 2^n$

Quindi: $c \geq 2^n / (2^n - 2^{n/2})$ -> 1 per n -> infinito

Esercizio 1 - 16/07/2021

Trovare limiti superiore e inferiore per la seguente equazione di ricorrenza, utilizzando il metodo di sostituzione:

$$T(n) = \begin{cases} T(n/2) + 2^n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Soluzione:

dove abbiamo $c > 1$ per qualsiasi $n > 1$. Scegliendo $c = 2$ abbiamo:

$2 \cdot 2^{n/2} + 2^n \leq 2 \cdot 2^n$

Quindi: $2 \cdot 2^{n/2} \leq 2^n$

Ovvero: $n/2 + 1 \leq n$ che è vero per qualsiasi $n \geq 2$, il che completa la dimostrazione.

Esercizio 3 - 16/07/2021

Si consideri il problema della ricerca di una chiave memorizzata in una matrice bidimensionale quadrata di dimensione N × N, in cui gli elementi sono ordinati in modo decrescente sia lungo le righe che lungo le colonne. Si scriva una procedura (in pseudocodice) che restituisca gli indici a cui si trova la chiave cercata, oppure dei valori nulli (p.es. [− 1,− 1]) se non viene trovata. Si stimi la complessità dell'algoritmo proposto.

Soluzione:

Per eseguire la ricerca in modo efficiente si deve sfruttare l'ordinamento che, come detto
nel testo, si applica sia alle righe che alle colonne. L'idea dell'algoritmo è che:

1. Se la chiave si trova in posizione [1, n] allora la ricerca ha successo;

2. Altrimenti, se la chiave è minore del valore in posizione [1, n], allora nessuno degli elementi della riga 1 può essere minore o uguale alla chiave, e quindi la riga 1 può essere trascurata, e si continua la ricerca sulla sottomatrice di dimensione (n − 1) × n che inizia dalla riga 2;

3. Altrimenti, se la chiave è maggiore del valore in posizione [1, n], allora nessuno degli elementi della colonna n può essere maggiore o uguale alla chiave, e quindi la colonna n può essere trascurata, e si prosegue la ricerca sulla sottomatrice di dimensione n × (n − 1) che nisce alla colonna n − 1.

Esercizio 3 - 16/07/2021

Poiché ad ogni passo viene eliminata una intera riga oppure una intera colonna, la procedura termina in al più 2 N passi, a partire dalla chiamata MatrixSearch (A, k, 1, 1, N, N).

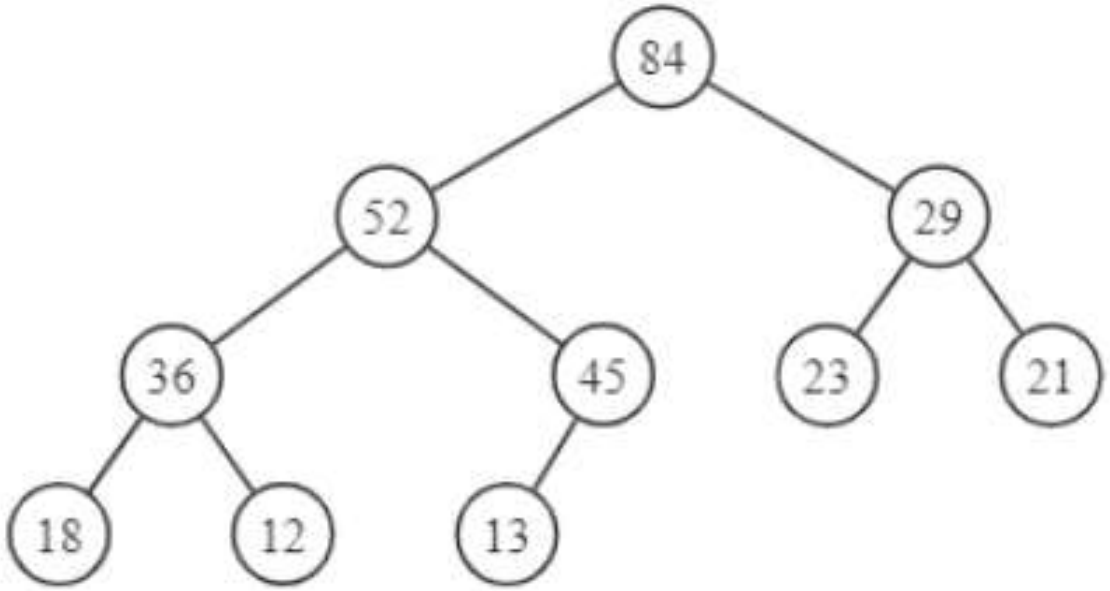
Procedure MatrixSearch(A, k, i, j, m, n)

Input: Array ordinato $A[i, j]$, $i, j = 1, \ldots, N$, chiave k, estremi della sottomatrice corrente i, j, m, n;
if $i > m$ *or* $j > n$ **then**
    Result: $[-1, -1]$
else
    if $A[i, n] = k$ **then**
        Result: $[i, n]$
    else
        if $A[i, n] > k$ **then**
            Result: MatrixSearch$(A, k, i + 1, j, m, n)$
        else
            Result: MatrixSearch$(A, k, i, j, m, n - 1)$

Si noti che sarebbe stato possibile effettuare una scansione lineare della matrice, ottenendo però una complessità di $O(n^2)$

Esercizio 3 - 16/07/2021

Dato il seguente max-heap, si disegni l'heap ottenuto dopo aver aggiunto la chiave 64.



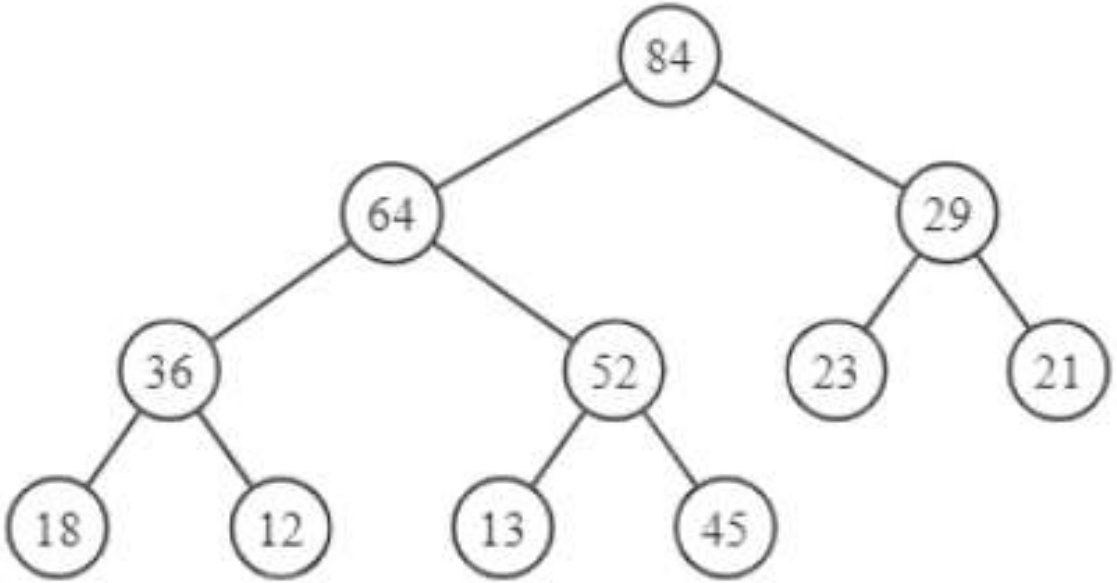
Esercizio 3 – 16/07/2021

Dato il seguente max-heap, si disegni l'heap ottenuto dopo aver aggiunto la chiave 64.

Soluzione:

Ricavo l'array corrispondente all'heap: [84,52,29,36,45,23,21.18,12,13]. Aggiungo 64 in fondo all'array: [84,52,29,36,45,23,21.18,12,13, 64]. In questo modo le proprietà di maxHeap non sono rispettate, quindi inverto 64 con il suo nodo padre fino a garantire che 64 sia il nodo dal valore più alto nel suo sotto-albero.
[84,52,29,36, 64,23,21.18,12,13, 45] ; [84,64,29,36,52,23,21.18,12,13, 45]

ottenendo:



Esercizio 5 - 15/07/2022

Si consideri il problema di calcolare la potenza n -esima di un numero reale. Quale complessità si può ottenere, considerando significative le operazioni di moltiplicazione?

Soluzione:

Il calcolo della potenza n -esima si può chiaramente effettuare con un codice del tipo

```
int n_pow(int n,int x){
        unsigned int pow = x;
        for(int i = 1;i<n;i++){
                pow = pow * x;
        }
        return pow;
}
```

con una complessità evidentemente lineare.

Esercizio 5 - 15/07/2022

Tuttavia è possibile ottenere una complessità logaritmica basandosi su una decomposizione ad esempio della potenza n = 5 come 4 + 1, ed a sua volta la potenza 4 si ottiene con due moltiplicazioni, una per generare x^2 ed una per generare $x^4 = x^2\ x^2$, per un totale di 3 moltiplicazioni. Il numero di moltiplicazioni nel risultato p corrisponde al numero di bit pari ad 1 nella rappresentazione binaria di n, e si preparano le potenze facendo tante moltiplicazioni quante sono le cifre totali di n. In definitiva, si può usare l'algoritmo:

Esempio:
$7 = (0111)_2$, quindi servono 3 moltiplicazioni x^7 = x*6 * x = x^4 * x^2 * x
Quindi calcolo x^2 = x*x;
Calcolo x^4 = x^2 * x^2
Moltiplico x^4 * x^2 * x e ottengo il risultato

Costo?
$T(n) \leq 2 \lceil \log(n) \rceil$, perché abbiamo al massimo $2 \lceil \log(n) \rceil$ moltiplicazioni. Per cui
$T(n) = O(\log(n))$

Esercizio 5 - 15/07/2022

Un implementazione in C potrebbe essere la seguente (Vedi codice allegato)

```c
int log_n_pow(int n, int x){
        unsigned int k;
        unsigned int p = 1;

        while(n>0){
                k = n&1;
                if(k > 0)
                        p = p*x;
                x = x*x;
                n = n>>1;
        }
        return p;
}
```

Esercizio 3 - 15/07/2022

Dato il codice C della funzione enigma si risponda alle seguenti domande:
1. si descriva quale sia lo scopo della funzione e quale computazione essa compia con i dati input;
2. se si volesse utilizzare la funzione enigma come funzione hash per il tipo di dato "string", 0 sarebbe un valore adeguato per x? Si motivi la risposta.

```
int x = ...;

int enigma(const char * s, int n) {
  int h = 0;
  for (int k = n - 1; k >= 0; --k)
    h = x * h + int(s[k]);
  return h;
}
```

Esercizio 3 - 15/07/2022

1. si descriva quale sia lo scopo della funzione e quale computazione essa compia con i dati input;

```
int x = ...;

int enigma(const char * s, int n) {
  int h = 0;
  for (int k = n - 1; k >= 0; --k)
    h = x * h + int(s[k]);
  return h;
}
```

Soluzione:

Data una stringa s di dimensione n, la funzione enigma valuta il valore intero dei caratteri in s come un polinomio in x: $\sum_{i=0}^{n-1} s[i[* x^i$

Esercizio 3 - 15/07/2022

2. se si volesse utilizzare la funzione enigma come funzione hash per il tipo di dato "string", 0 sarebbe un valore adeguato per x? Si motivi la risposta.

```
int x = ...;

int enigma(const char * s, int n) {
  int h = 0;
  for (int k = n - 1; k >= 0; --k)
    h = x * h + int(s[k]);
  return h;
}
```

Soluzione:
Il valore 0 non sarebbe adeguato per x se si volesse usare enigma come funzione hash perché la funzione restituirebbe sempre s[0]; in altre parole, tutte le stringhe che iniziano con lo stesso carattere avrebbero la stessa hash.

Esercizio 4 - 24/02/2022

Si consideri il metodo per verificare se un numero sia primo che consiste nel tentare di dividere per tutti i fattori primi possibili. L'algoritmo è in P?

Soluzione:

L'algoritmo è pseudopolinomiale. Infatti, dato il numero n, esso deve provare tutti i divisori possibili $k \leq n/2$, e quindi il suo costo è O (n). Sembrerebbe quindi un problema polinomiale, ma siccome la dimensione dell'input (numero di cifre necessarie per rappresentare n) è pari al log(n), il problema non è polinomiale, ma pseudopolinomiale (in quanto il suo tempo di esecuzione è un esponenziale nella dimensione dell'input); se si impone un limite superiore al valore di n, allora diventa a tutti gli effetti polinomiale.

Esercizio 4 - 16/07/2021

Si considerino due insiemi con chiavi numeriche A e B rappresentati utilizzando degli alberi rosso-neri. Si stimi la complessità delle due operazioni:

- Unione
- Differenza simmetrica (insieme di tutti gli elementi che sono contenuti solo in uno dei due insiemi).

Soluzione:

Per realizzare gli operatori di unione e differenza simmetrica è suciente scorrere i due insiemi nell'ordine indotto dalla chiave numerica, confrontando ad ogni passo gli elementi considerati. Per l'unione si usa la procedura di merge, per la differenza simmetrica si adatta la procedura di merge per aggiungere l'elemento all'insieme di uscita se presente in solo uno dei due insiemi A e B. Poichè le operazioni di ricerca del minimo (per il valore iniziale) e di ricerca del successore (per l'avanzamento) sono entrambe di complessità O (h) dove h è l'altezza dell'albero, che è logaritmica nel numero di elementi, si ha una complessità O (max(|A|, |B|) · log(max(|A|, |B|))).