

Esercitazioni di Algoritmi e Strutture Dati

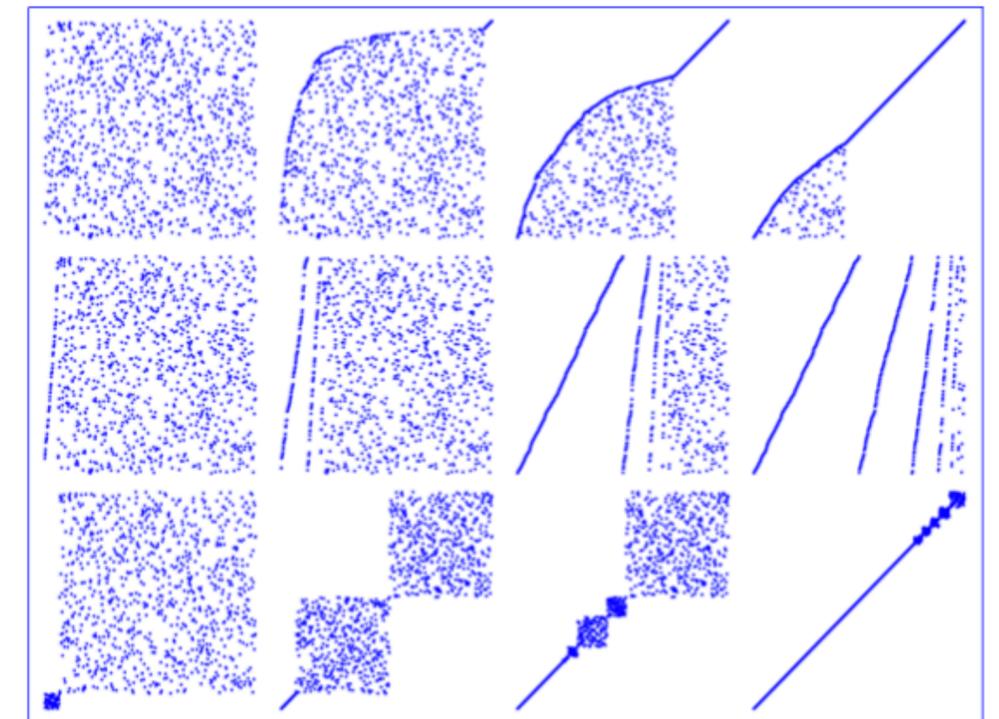
Esercizi di Algoritmi e Strutture Dati

Gli esercizi presentati a lezione sono presi da

<https://moreno.marzolla.name/teaching/ASD/>

Le slides con gli esercizi sono scaricabili da

<https://saveriogiallorenzo.com/teaching/#asd>



Moreno Marzolla
<http://www.moreno.marzolla.name/>

Esercitazione(**Esercizio e) {**
leggere il testo di e insieme;
timeout(10' , { Provare a svolgere e singolarmente });
for((Studente, Soluzione**) (st, so) : studenti . soluzioni) {**
 st presenta brevemente la so che ha trovato;
}
contestualizzazione di e , soluzione da  e discussione;
}

1. Notazioni Asintotiche

Esercizio 1.1.0

Determinare il costo asintotico dell'algoritmo seguente

```
ALGB( integer n ) → integer  
integer a ← 0;  
Integer s, t;  
for s ← 1 to n do  
    for t ← s to n do  
        a ← a + s + t;  
    endfor  
endfor  
return a;
```

Esercizio 1.1

Sia $f(n) = n(n+1)/2$. Utilizzando la definizione di $O(\)$, dimostrare o confutare le seguenti affermazioni:

1. $f(n) = O(n)$
2. $f(n) = O(n^2)$

Esercizio 1.2 Si consideri la funzione $FUN(n)$, con $n \geq 1$ intero, definita dal seguente algoritmo ricorsivo:

```
algoritmo FUN( int n ) → int  
  if (n ≤ 2) then  
    return n;  
  else  
    return FUN(n-1) - 2*FUN(n-2);  
  endif
```

1. Determinare un limite inferiore sufficientemente accurato del tempo di esecuzione $T(n)$
2. Determinare un limite superiore sufficientemente accurato del tempo di esecuzione $T(n)$

Esercizio 1.3 Scrivere un algoritmo il cui costo computazionale $T(n)$ sia dato dalla seguente relazione di ricorrenza:

$$T(n) = \begin{cases} O(1) & \text{se } n \leq 10 \\ nT(n-1) + O(1) & \text{altrimenti} \end{cases}$$

dove n è un parametro intero positivo passato come input all'algoritmo. Non è richiesto il calcolo della soluzione della ricorrenza, né è richiesto che l'algoritmo produca un risultato di una qualsivoglia utilità pratica.

Esercizio 1.4

Supponiamo di disporre di un “coprocessore speciale” in grado di fondere due array ordinati, aventi complessivamente n elementi, in tempo $O(\sqrt{n})$. Determinare il costo computazionale dell'algoritmo di ordinamento MERGESORT visto a lezione, in cui la fase di fusione dei (sotto-)vettori ordinati viene realizzata mediante il coprocessore speciale di cui sopra.

Esercizio 1.5

Si consideri il seguente algoritmo ricorsivo:

```
algoritmo FUN( array A[1..n] di double, int i, int j ) → double  
  if ( i > j ) then  
    return 0;  
  elseif ( i == j ) then  
    return A[i];  
  else  
    int m := FLOOR( ( i + j ) / 2 );  
    return FUN(A, i, m) + FUN(A, m+1, j);  
  endif
```

L'algoritmo accetta come parametri un array $A[1..n]$ di n numeri reali e due interi, i e j ; l'algoritmo viene inizialmente invocato con $\text{FUN}(A, 1, n)$ e restituisce un numero reale.

1. Scrivere la relazione di ricorrenza che descrive il costo computazionale di FUN in funzione di n ;
2. Risolvere la ricorrenza di cui al punto 1;
3. Cosa calcola $\text{FUN}(A, 1, n)$ (spiegare a parole)

1. Strutture Dati Elementari

Esercizio 2.0a

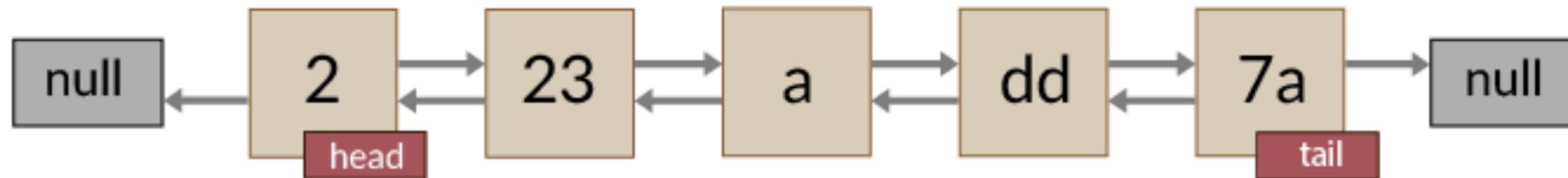
Che differenza c'è la LinkedList e ArrayList in Java (come forma e i relativi costi asintotici delle operazioni)

	LinkedList	ArrayList
Inserimento in testa		
Inserimento in coda		
Inserimento dopo un elemento di posizione/riferimento dati		
Cancellazione di un elemento di posizione/riferimento dati		
Accesso diretto al k -esimo elemento		

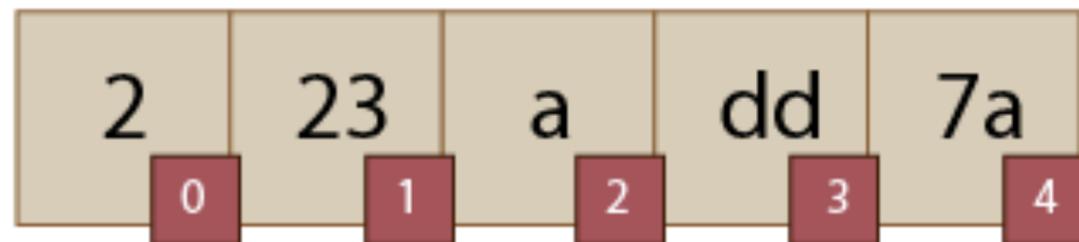
Esercizio 2.0a

Che differenza c'è la LinkedList e ArrayList in Java (come forma e i relativi costi asintotici delle operazioni)

LinkedList



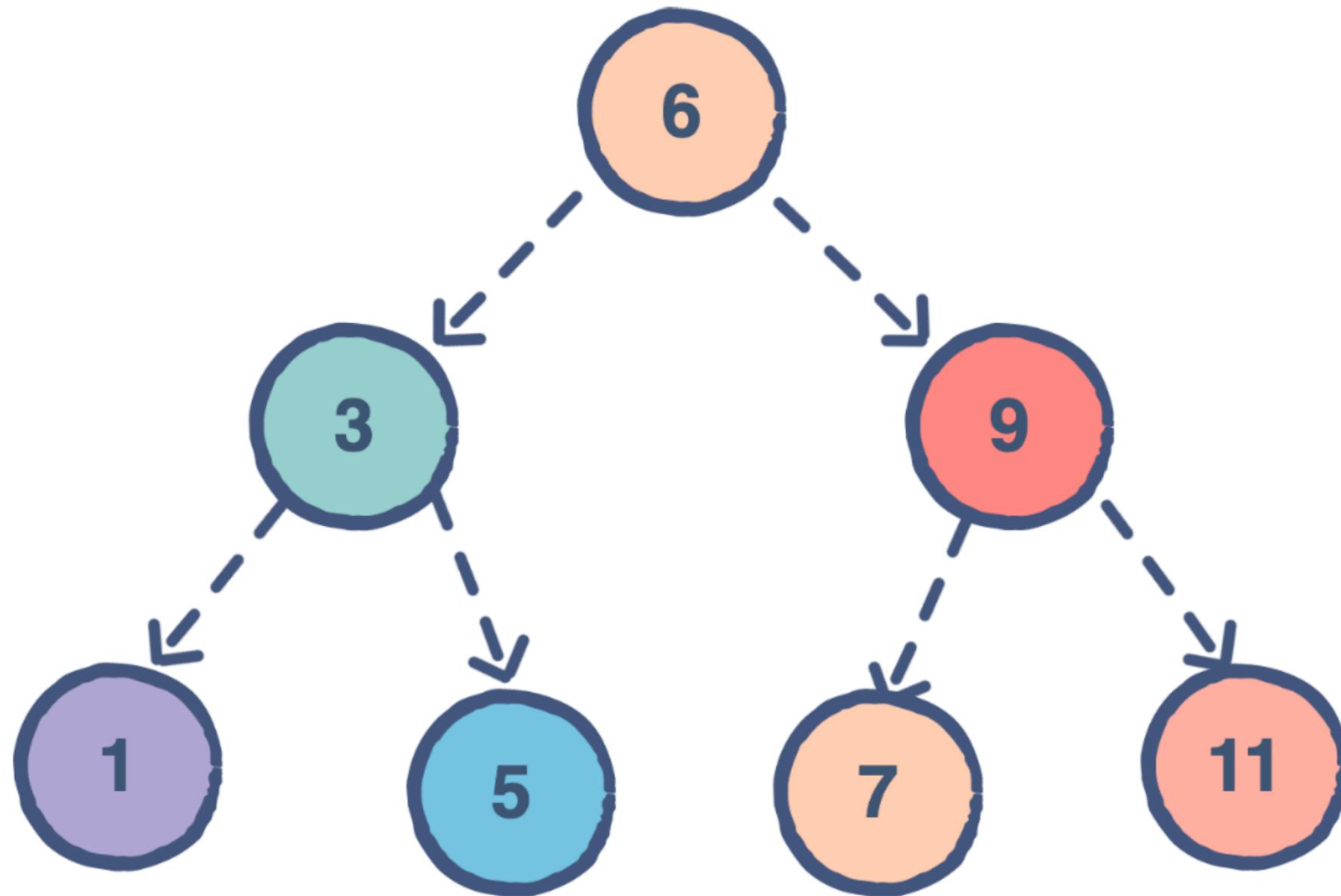
Array and ArrayList



Esercizio 2.0b

Considerando un albero binario non bilanciato con n nodi

- qual'è il costo asintotico dell'operazione di ricerca nel caso pessimo?
- qual'è il costo asintotico dell'operazione di ricerca nel caso ottimo?



Esercizio 2.1

Scrivere un algoritmo efficiente per risolvere il seguente problema: dato un array $A[1..n]$ di $n > 0$ valori reali, restituire *true* se l'array A rappresenta un min-heap binario, *false* altrimenti. Calcolare la complessità nel caso pessimo e nel caso ottimo dell'algoritmo proposto, motivando le risposte.

Esercizio 2.3

Si consideri un albero binario di ricerca non bilanciato, inizialmente vuoto. Disegnare gli alberi che risultano dopo l'inserimento di ciascuna delle seguenti chiavi numeriche: 17, 7, 9, -3, 20, 19, 5, 2, 6.

Esercizio 2.4a

Si consideri un albero binario B in cui a ciascun nodo t è associata una chiave numerica (reale) $t.key$. Non ci sono chiavi ripetute.

1. Scrivere un algoritmo efficiente che dato in input l'albero B e due valori reali a e b , con $a < b$, restituisce true se e solo se B rappresenta un albero binario di ricerca le cui chiavi siano tutte comprese nell'intervallo $[a, b]$. Si noti che è necessario controllare esplicitamente che i valori delle chiavi appartengano all'intervallo dato, poiché in generale l'albero B può contenere chiavi arbitrarie. Non è consentito usare variabili globali.

Esercizio 2.4b

Si consideri un albero binario B in cui a ciascun nodo t è associata una chiave numerica (reale) $t.key$. Non ci sono chiavi ripetute.

2. Calcolare il costo computazionale nel caso ottimo e nel caso pessimo dell'algoritmo di cui al punto 1. Disegnare un esempio di albero che produce un caso pessimo, e un esempio di albero che produce il caso ottimo.

Esercizio 2.5

Si consideri un albero binario di ricerca B , non necessariamente bilanciato, in cui a ciascun nodo t è associata una chiave numerica (reale) $t.key$. Non ci sono chiavi ripetute.

1. Scrivere un algoritmo efficiente che dato in input l'albero B e due valori reali a e b , con $a < b$, restituisce il numero di nodi la cui chiave appartiene all'intervallo $[a, b]$ (estremi inclusi)
2. Calcolare il costo computazionale dell'algoritmo di cui al punto 1 nel caso ottimo e nel caso pessimo.

Esercizio 2.6

Si consideri un albero binario di ricerca non bilanciato, inizialmente vuoto.

1. Disegnare la struttura degli alberi risultanti dopo l'inserimento di ognuna delle seguenti chiavi, nell'ordine indicato: 20, 3, 37, 30, 1, 15, 25, 32.
2. Disegnare la struttura dell'albero dopo la rimozione del valore 37.

Esercizio 2.7

Si consideri un albero binario T , avente struttura arbitraria, in cui ciascun nodo t contenga un numero reale $t.val$;

1. Scrivere un algoritmo efficiente che, dato in input T , restituisca la somma dei valori contenuti in tutte le foglie; se l'albero è vuoto l'algoritmo restituisce zero. È vietato usare variabili globali.
2. Assumendo che l'albero T sia completo e tutte le foglie si trovino sullo stesso livello, determinare formalmente il costo computazionale dell'algoritmo in funzione del numero n di nodi.