DELIMITAZIONI SUPERIORI E INFERIORI DI UN PROBLEMA

UPPERBOUND IN UN PROBLEMA

SIA P UN PROBLEMA, P HA UNA COMPLESSITÀ $\forall O(f(m))$ (DELIMITAZ. SUPERIORE) RISPETTO AD UNA RISORSA DI CALCOLO SE ESISTE UN ALGORITMO CHE RISOLVE IL PROBLEMA IL CUI COSTO DI ESECUZIONE RISPETTO A QUELLA RISORSA È PROPRIO $O(f(m))$.

ESEMPIO

PROBLEMA DELL'ORDINAMENTO

UPPER BOUND $O(m^2)$:
INSERTION SORT, SELECTION SORT, BUBBLE SORT...

UPPER BOUND MIGLIORE $O(m \log m)$
MERGE SORT, HEAP SORT

LOWERBOUND IN UN PROBLEMA

UN PROBLEMA P HA UNA COMPLESSITÀ $\forall \Omega(f(m))$ (DELIMIT. INFERIORE) RISPETTO AD UNA DI CALCOLO SE OGNI ALGORITMO CHE RISOLVE P HA COSTO DI ESECUZIONE NEL CASO PEGGIORE $\Omega(f(m))$.

ESEMPIO

PROBLEMA DELL'ORDINAMENTO

LOWER BOUND: $\Omega(m) \longrightarrow$ OGNI ALGORITMO DEVE ALMENO LEGGERLI

OTTIMALITÀ DI UN ALGORITMO

DATO UN PROBLEMA P CON COMPLESSITÀ $\Omega(f(m))$, UN ALGORITMO CHE RISOLVE P È ASINTOTICAMENTE OTTIMO SE HA COSTO DI ESECUZIONE $O(f(m))$.

TEOREMA

OGNI ALGORITMO **BASATO SU CONFRONTI** CHE ORDINA n ELEMENTI DEVE FARE NEL CASO PEGGIORE $\Omega(n \log n)$ CONFRONTI

↓

IL MERGE SORT E L'HEAP SORT SONO ALGORITMI OTTIMI POICHÉ IMPIEGANO $O(n \log n)$.

↳ GUARDA DEF.

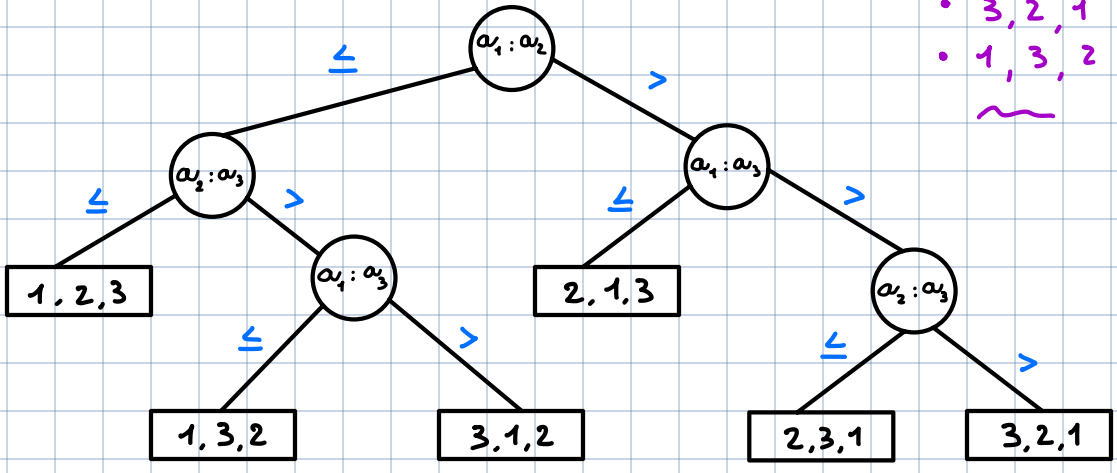
ALBERI DI DECISIONE

DESCRIVE I CONFRONTI CHE L'ALGORITMO ESEGUE SU UN INPUT DI UNA DETERMINATA DIMENSIONE.

INPUT: a_1, a_2, a_3

TEST CASE:
- 1, 2, 3
- 3, 2, 1
- 1, 3, 2

} PROVA



OGNI ALBERO DI DECISIONE È ASSOCIATO AD UN:
- ALGORITMO
- DIMENSIONE DELL'ISTANZA

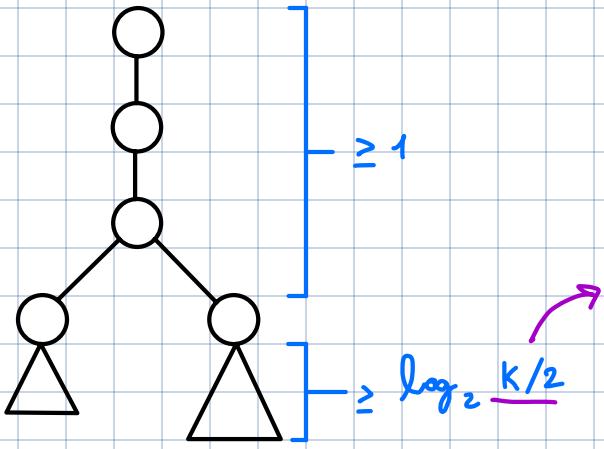
DESCRIVE LE DIVERSE SEQUENZE DI CONFRONTI.
UN ALGORITMO CHE ORDINA n ELEMENTI AVRÀ UN ALBERO DI DECISIONE CON ALMENO $n!$ FOGLIE.

↳ PERMUTAZIONI

LEMMA

SIA T UN ALBERO BINARIO CON K FOGLIE, LA SUA ALTEZZA È ALMENO: $(h \geq \dots)$

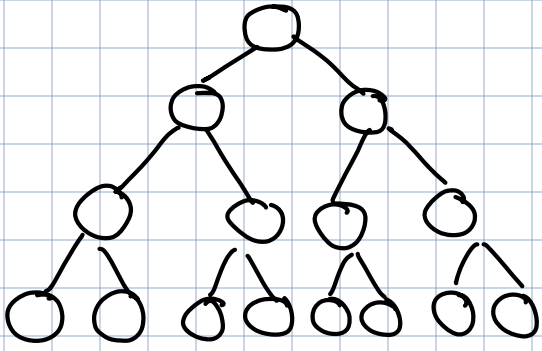
$$\log_2 k$$!!



K/2 È IL LOWER BOUND, AVENDO 2 SOTTO ALBERI LA DIMENSIONE PIÚ PICCOLA CHE POSSONO AVERE INSIEME È PROPRIO K/2.

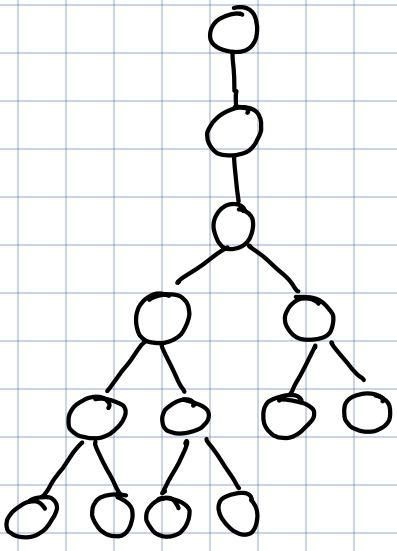
SE DOVESSIMO AVERE NEL SOTTOALB. DI SX 2 FOGLIE E NEL SOTTOALB. DI DX K-2 FOGLIE, K-2 È MAGGIORE DI K/2 CHE È IL LOWER BOUND.

ESEMPIO



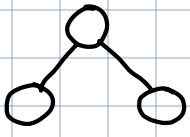
$K = 8 \qquad h \geq \log_2 k = \log_2 8 = 3$ ✓

VERIFICATA INFATTI L'ALTEZZA È 3



$K = 6 \qquad h \geq \log_2 6 = 2{,}5\dots$ ✓

VERIFICATA INFATTI L'ALTEZZA È 5



$K = 2 \qquad h \geq \log_2 2 = 1$ ✓

VERIFICATA

INTEGER SORT (NON BASATO SU CONFRONTI)

SUPPONIAMO DI AVERE UN ARRAY DI M ELEMENTI DA ORDINARE CON VALORI CHE VANNO DA 1 A K.

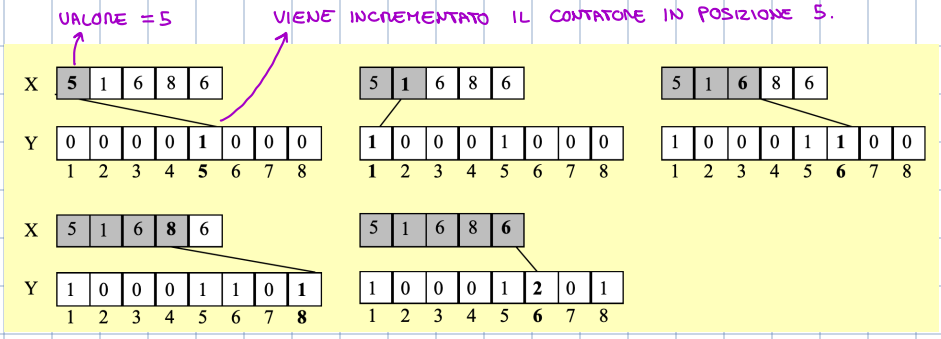
$X =$ | 5 | 1 | 6 | 8 | 6 |

VIENE POI CREATO UN SECONDO ARRAY DI CONTATORI LUNGO K, INIZIALIZZATO A 0.

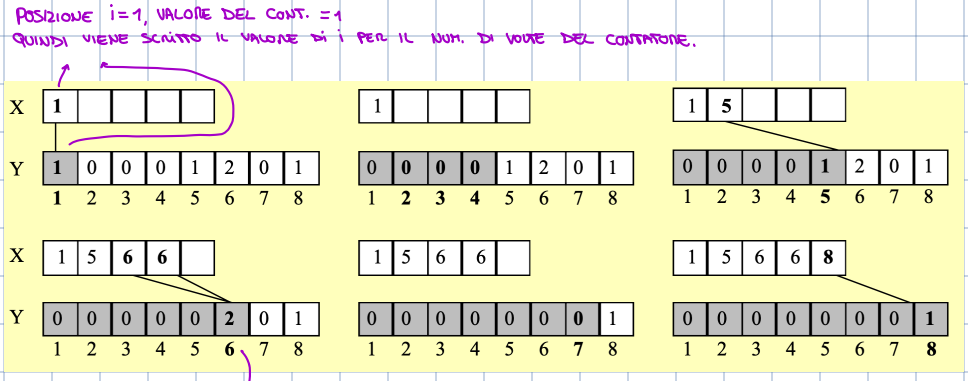
$Y =$

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

SCORRENDO L'ARRAY X, OGNI VOLTA CHE SI PRENDE UN ELEMENTO SI INCREMENTA IL CONTATORE NEL SECONDO ARRAY CHE SI TROVA NELLA POSIZIONE PARI AL VALORE DELL'ELEMENTO PRESO.



FATTO CIÓ, SCORRE L'ARRAY Y E IN BASE AL VALORE DEL CONTATORE IN POSIZIONE i-ESIMA, SCRIVE IL NUMERO i NELL'ARRAY X TOT. VOLTE IN BASE AL VALORE DEL CONTATORE, SE E SOLO SE IL VALORE DEL CONTATORE È >0.



BANALMENTE POSSIAMO DIRE CHE IN Y AVREMO IN OGNI POSIZIONE J, IL NUMERO DI VOLTE CHE J COMPARE DENTRO X.

IntegerSort (X, k)

```
1.  Sia Y un array di dimensione k          } O(1) - tempo costante
2.  for i=1 to k do Y[i]=0                  } O(k)
3.  for i=1 to n do incrementa Y[X[i]]      } O(n)
4.  j=1                                     } O(1)
5.  for i=1 to k do                         } O(k)
6.      while (Y[i] > 0) do                 }
7.          X[j]=i                          } per i fissato
8.          incrementa j                    } #volte eseguite è al più 1+Y[i]  ⇒ O(k+n)
9.          decrementa Y[i]                 }
```

+1 PERCHÉ SE AVESSIMO IL CONTATORE Y[i]=0 ESEGUIREMMO COMUNQUE 1 VOLTA LA RIGA N°6.

$$\sum_{i=1}^{k}(1+Y[i]) = \sum_{i=1}^{k} 1 + \sum_{i=1}^{k} Y[i] = k + n$$

k → AVENDO K ITERAZIONI (XCHÉ Y È LUNGO K) LA RIGA 6 VIENE ESEGUITA K VOLTE

m → SICCOME IL NOSTRO ARRAY PRIMARIO HA m ELEMENTI, SE CI PENSIAMO LA SOMMA DI TUTTI I CONTATORI È UGUALE AD m (x)

ESEMPIO [5,1,1,1,4], CONTATORI → Y[1]=3, Y[4]=1, Y[5]=1

(m ELEMENTI)

SOMMANDO I CONTATORI ABBIAMO PROPRIO m.

ABBIAMO QUINDI UNA COMPLESSITÀ DI $O(m+k)$, POSSIAMO DIRE CHE L'EFFICIENZA DIPENDE UN PÓ DAL VALORE DI K POICHÉ IL SECONDO ARRAY AVRÀ DIMENSIONE FINO A K.

CASO IN CUI NON È EFFICIENTE

SE DOVESSIMO AD ESEMPIO AVERE $K = m^2$ AVREMMO UNA COMPLESSITÀ DI:

$$O(m + m^2) = O(m^2)$$

E AVENDO m^2 A QUESTO PUNTO POTREMMO CONCLUDERE CHE NON CI CONVIENE USARE QUESTO ALGORITMO MA UN ALTRO COME AD ESEMPIO MERGE SORT.

CASO IN CUI È EFFICIENTE

COME ABBIAMO VISTO, IL LOWER BOUND DEGLI ALGORITMI BASATI SU CONFRONTO È $\Omega(m \log m)$.

SE DOVESSIMO AVERE $K = O(m)$ AVREMMO UNA COMPLESSITÀ TOTALE DI:

$$O(m + m) = O(m)$$

CHE È MIGLIORE DEL $\Omega(m \log m)$ DEGLI ALGORITMI BASATI SU CONFRONTO.

↓

QUINDI ABBIAMO "ABBATTUTO" IL LOWER BOUND $\Omega(m \log m)$? NO! PERCHÉ IL NOSTRO INTEGER SORT NON È UN ALGO BASATO SUI CONFRONTI !!

INTEGER SORT

↘

ABBIAMO UN ARRAY X DI M ELEMENTI CHE VANNO DA 1 A K.
CREIAMO UN SECONDO ARRAY LUNGO K, OGNI CELLA È UN CONTATORE INIZIALIZZATO A 0.
SCORRIAMO X, PER OGNI NUMERO INCREMENTIAMO IL CONTATORE IN Y NELLA POSIZIONE UGUALE AL NUMERO INCONTRATO.
SCORRIAMO Y E PER OGNI CONTATORE INSERIAMO IN X (SOVRASCRIVENDO) IL NUMERO DELLA POSIZIONE PER TOT VOLTE QUANTO È IL CONTATORE. $O(m+k)$