Università di Pisa
Dipartimento di Ingegneria dell'informazione

Ingegneria Informatica

Algoritmi & Strutture dati

Raccolta delle diapositive del prof.Virdis

A.A 2019-2020 - Secondo semestre

File realizzato da Gabriele Frassi, disponibile presso la copisteria *OneCent*

Indice

Algoritmi e Strutture Dati

Lezione 1

www.iet.unipi.it/a.virdis

Antonio Virdis

antonio.virdis@unipi.it

“Ma io so già programmare!”

Fondamenti I	Fondamenti II
Sia dato un array contenente delle **frasi**. Scrivere un programma che prenda in input una **stringa** e restituisca tutte le frasi che la contengono.	Realizzare un motore di ricerca che abbia complessità O(log n) sulle operazioni di lettura.

Algoritmi e **Strutture Dati**

Informazioni

- Lezioni teorico-pratiche
- Esercizi assegnati per casa
- Esercitazioni tramite portale online
- Esame in laboratorio

Pre Requisiti

- Fondamenti I
- Utilizzo compilatore
- Comandi base unix

Sommario

- Ordinamento: Insertion Sort
- Standard Template Library
- Debug

esperimenti

- Implementare **InsertionSort**
- Chiedere a **InsertionSort** come funziona
- Analizzare casi limite
- Prestazioni

Ordinamento

Stampa

```
// stampa i "len" valori contenuti in arr,
// separati da tabulazioni
     stampaArray( int arr[] , int len );



/*Scrivere un programma che:
  - possa memorizzare 10 interi
  - memorizzi 10 interi
  - usi una funzione per stampare i 10 interi     */
int main()
{
    const int sSize = 10;
    int sArray[sSize] = { 9,5,1,14,0,9,5,1,14,0 };


}
```


Stampa

```
1  #include <iostream>
2  Using namespace std;
3
4  void stampaArray( int arr[] , int len )
5  {
6      for( int i=0 ; i < len ; ++i )
7          cout << arr[i] << "\t" ;
8      cout << endl;
9  }
10
11
12 int main( )
13 {
14     const int sSize = 10;
15     int sArray[sSize] = { 9,5,1,14,0,9,5,1,14,0 };
16     stampaArray(sArray,sSize);
17     return 1;
18 }
```


Nomi delle Variabili

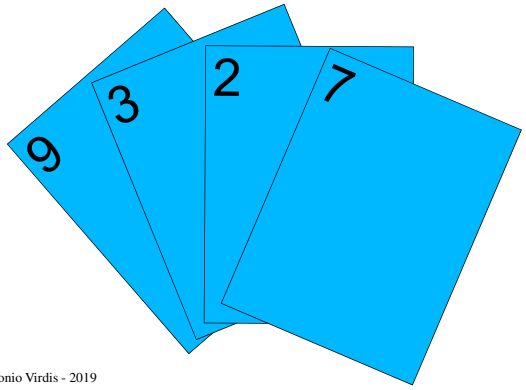
prova pippo

```
1
2
3
4
5
6
7  int main( ){const int a = 10;
8  int pippo[a] = { 9,5,1,14,0,9,5,1,14,0 };
9  prova(pippo,a);
10 return 1;}
11
12
13
14
15
16
17
18
```

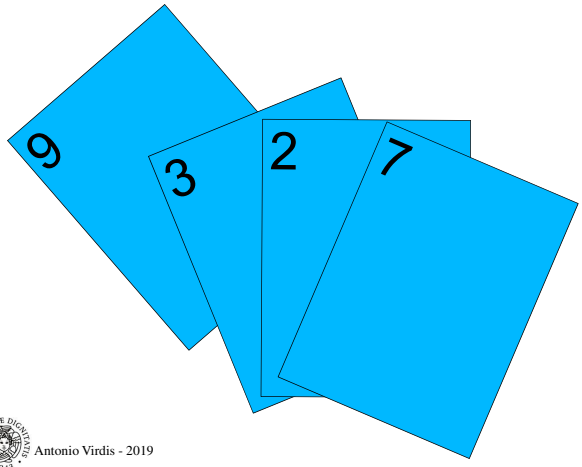
?

Insertion Sort: rappresentazione

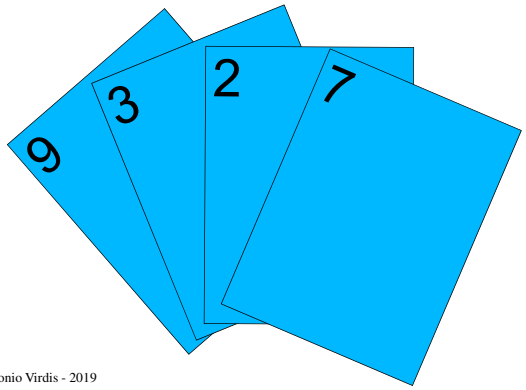
Ordinare Carte da Gioco



Ordinare Carte da Gioco

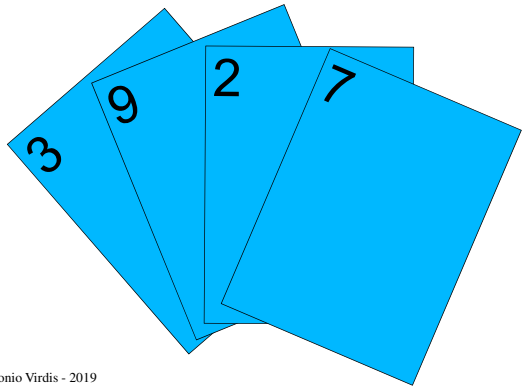


Ordinare Carte da Gioco

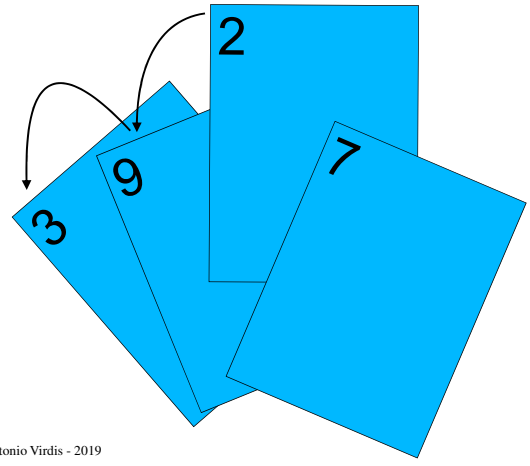


Ordinare Carte da Gioco

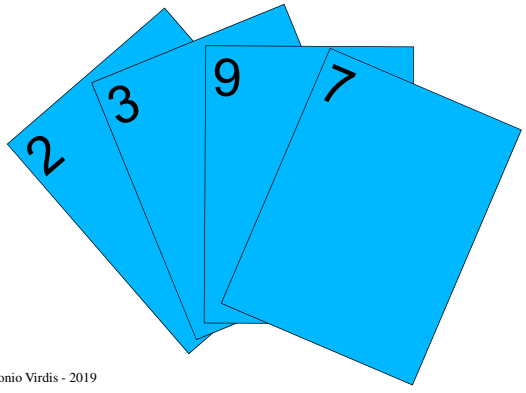
Ordinare Carte da Gioco



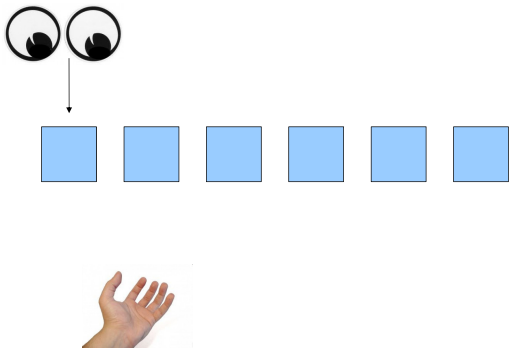
Ordinare Carte da Gioco



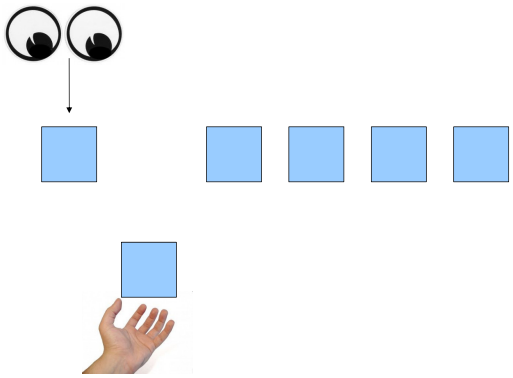
Ordinare Carte da Gioco



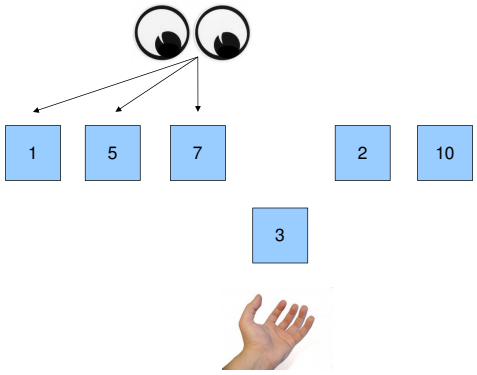
Insertion Sort: rappresentazione



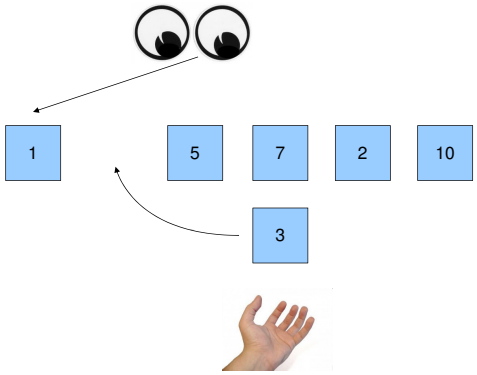
Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Insertion Sort: rappresentazione



Sorting

```
1  void sortArray( int arr[] , int len )
2  {
3
4
5      for( PER OGNI ELEMENTO DELLA FILA )
6      {
7          // INIZIALIZZO MANO E OCCHIO
8
9
10         while( TROVO POSIZIONE CORRETTA )
11         {
12             // SPOSTO OGGETTO
13             // SPOSTO OCCHIO
14         }
15
16         // LIBERO MANO
17     }
18 }
```


Sorting

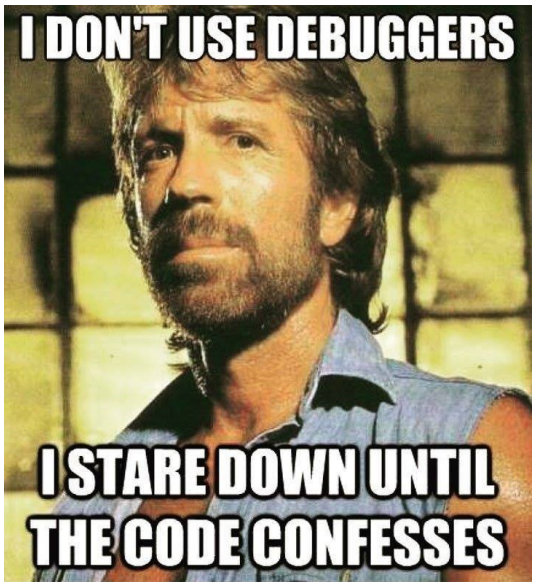
```
1  void sortArray( int arr[] , int len )
2  {
3      int mano = 0;
4      int occhio = 0;
5      for(          )
6      {
7
8
9
10         while(       )
11         {
12
13
14         }
15
16
17     }
18 }
```

Sorting

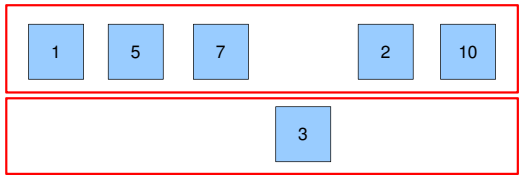
```
1  void sortArray( int arr[] , int len )
2  {
3      int mano = 0;
4      int occhio = 0;
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7          mano = arr[iter];
8          occhio = iter-1;
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12             arr[occhio+1] = arr[occhio];
13             --occhio;
14         }
15
16         arr[occhio+1] = mano;
17     }
18 }
```


Debugging

> “ If **debugging** is the process of **removing** software bugs,
>
> then **programming** must be the process of **putting** them in „
>
> **E. Dijkstra**



Debug Visuale



Tecniche

- Testo
- Visuale
- “Debugger” (es **GDB**)
- Compilatore
- Analisi Memoria (**Valgrind**)

Debug Visuale

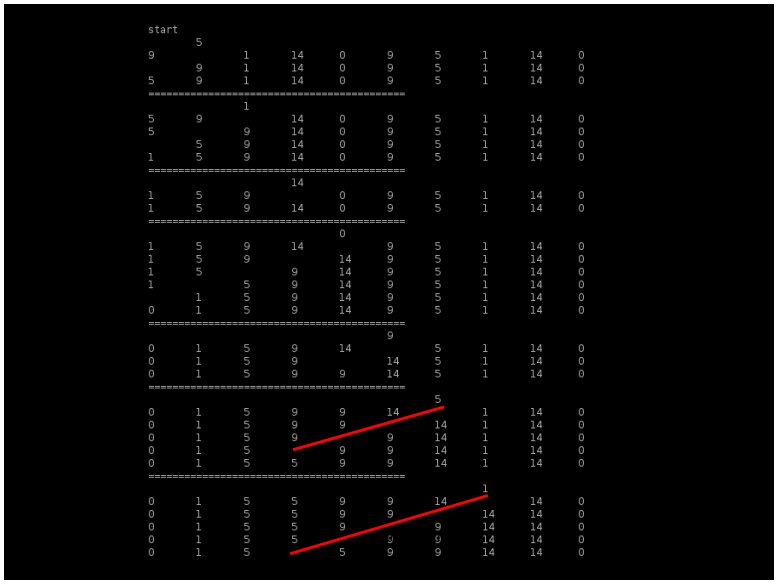
```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3
4
5
6
7
8
9
10
11
12
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16
17
18
19
20
```

Debug Visuale

```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3
4         // PER OGNI ELEMENTO
5
6         // SE SONO IN POSIZIONE buco, SALTO
7
8         // ALTRIMENTI STAMPO ELEMENTO
9
10
11
12
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16
17        // SALTO TUTTI GLI ELEMENTI FINO A posizione
18
19        // STAMPO IL SEGNO
20
```


Debug Visuale

```
1  // stampa con buco
2  void stampaArray( int arr[] , int len , int buco)
3  {
4      for( int i=0 ; i < len ; ++i )
5      {
6          if(i==buco)
7              cout << "\t";
8          else
9              cout << arr[i] << "\t" ;
10     }
11     cout << endl;
12 }
13
14 // stampa "segno" in "posizione"
15 void stampaSegno( int posizione , int segno )
16 {
17     for( int i = 0 ; i < posizione ; ++i )
18         cout << "\t";
19     cout << segno << "\n";
20 }
```

Analisi InsertionSort

```
1  void sortArray( int arr[] , int len )
2  {
3
4
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7
8
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12
13
14         }
15
16
17     }
18 }
```

$$\frac{n(n-1)}{2}$$

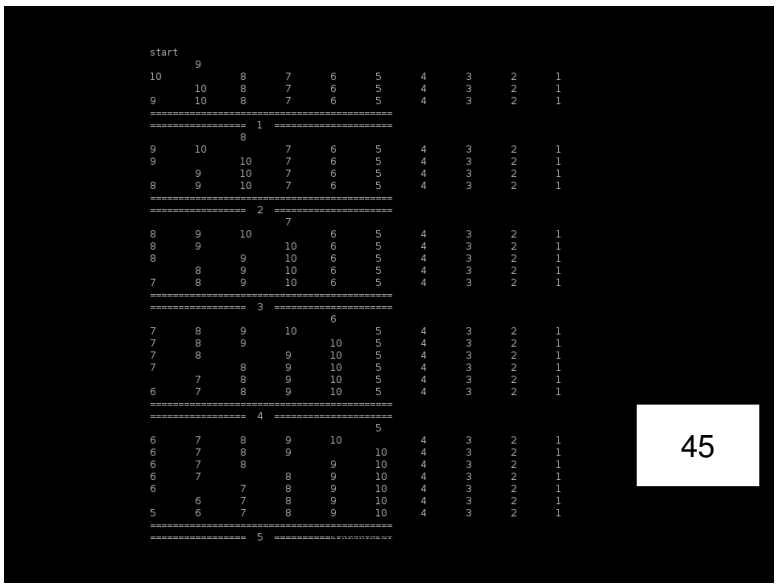
$$\Theta(n^2)$$

Esempio Worst Case

- Input di worst case?

10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1

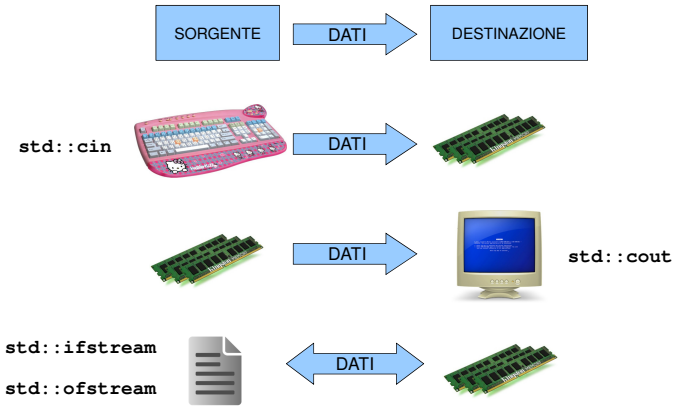

```
1  void sortArray( int arr[] , int l )
2  {
3      // init total e counter
4
5      for( int iter = 1 ; iter < len ; ++iter )
6      {
7
8          counter = 0;
9
10         while( occhio >= 0 && arr[occhio] > mano )
11         {
12             counter++;
13
14         }
15
16         total += counter;
17     }
18 }
```



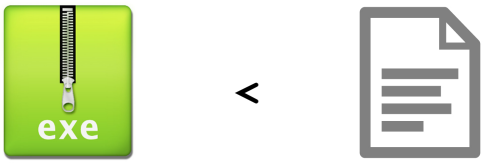
Memoria dinamica

- Quantità di dati **NON** nota a tempo di compilazione
- Quantità di dati **VARIABILE** durante l'esecuzione

Lettura Input



Redirezione DA File




```
./stlSort < reqFile
```

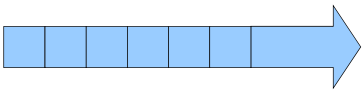

Lettura Input

```
1
2
3       leggiInput( )
4
5
6         // 1) LEGGO PRIMO VALORE (numero elementi)
7
8
9         // 2) ALLOCAZIONE MEMORIA
10
11
12        // 3) LETTURA CARATTERE PER CARATTERE
13
14
15
16
17
18
```


Lettura Input

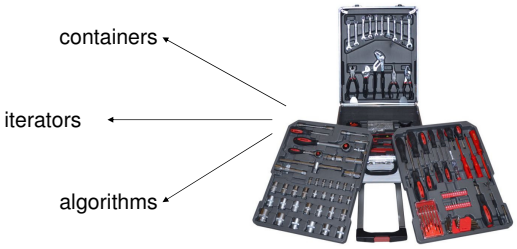
```
1
2   int * leggiInput( )
3   {
4
5       cin >> len;
6
7       int * arr = new int[len];
8
9
10      for( int i = 0 ; i < len ; ++i )
11          cin >> arr[i];
12
13
14      return arr;
15  }
16
17
18
```

Vettore



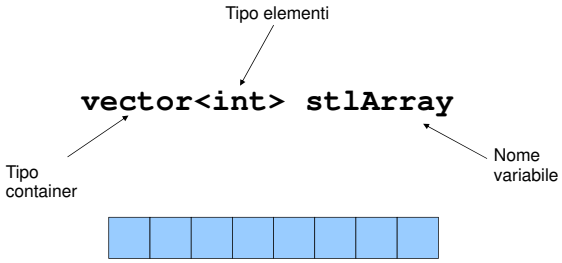
- Struttura dati di dimensione estendibile
- Accesso efficiente
- Algoritmi
- Magari già pronta?!?

Standard Template Library (STL)



Documentazione:
www.cplusplus.com/reference/stl/

Uso Vector

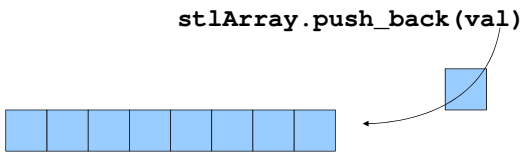


Uso Vector

```
vector<int> stlArray
```

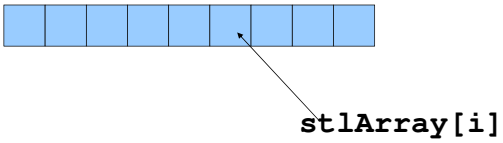

Uso Vector

```
vector<int> stlArray
```



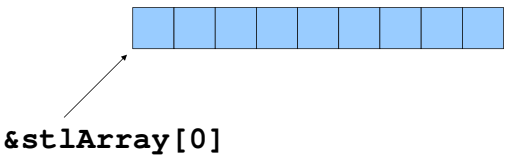
Uso Vector

```
vector<int> stlArray
```



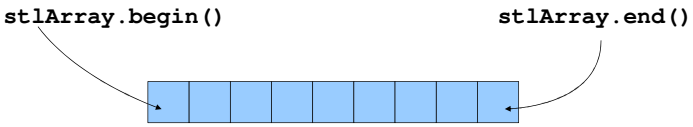
Uso Vector

vector<int> stlArray



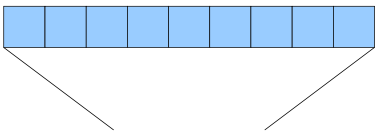
Uso Vector

vector<int> stlArray



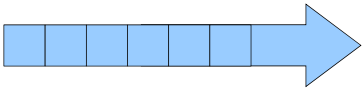
Uso Vector

vector<int> stlArray



Uso Vector

- Dinamico
  - Allocazione dinamica dimensione



- Contiguo
  - Accesso Random con costo costante
  - Gestione array-like ← **(con prudenza)**

File ⇨ Vector

```
1  #include <vector>
2
3  void leggiInput( std::vector<int> & arr )
4  {
5
6      cin >> len;
7
8      int val;
9      for( int i = 0 ; i < len ; ++i )
10     {
11         cin >> val;
12         arr.push_back(val);
13     }
14
15     return;
16 }
17
18
```


Sort STL

```
sort( stlArray.begin(),stlArray.end() );
```

restituisce un iteratore

$$\Theta(n \log n)$$

Qualche Test

Insertion Sort	vs	STL Sort
$\Theta(n^2)$		$\Theta(n \log n)$

```
time ./stlSort
```


Qualche Test

- Casi limite
  - Tutti uguali
  - Già ordinato
  - Ordine inverso
- Valori random
  - srand(seed) rand()%maxVal
- Comando time
  - time nomeEseguibile

Come Esercitarsi

- Input: input.txt Output: output.txt

LETTURA
```
cin >> valore;
```

INPUT
```
./eseguibile < input.txt
```

GENEZIONE OUTPUT
```
cout << uscita;
```

VERIFICA
```
./eseguibile < input.txt | diff - output.txt
```


Esercizio

Input: `input.txt`

```
3
1
9
15
```

Output: `output.txt`

```
25
135
yes
```

Input

- Il primo carattere indica il numero di valori da leggere
- Un valore per riga

Output

- Somma dei valori
- Prodotto dei valori
- I valori sono positivi? Rispondere yes o no

Algoritmi e Strutture Dati

Lezione 2

www.iet.unipi.it/a.virdis

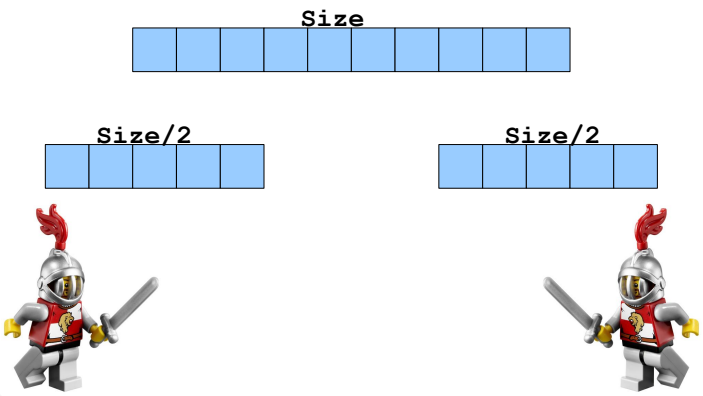
Antonio Virdis

antonio.virdis@unipi.it

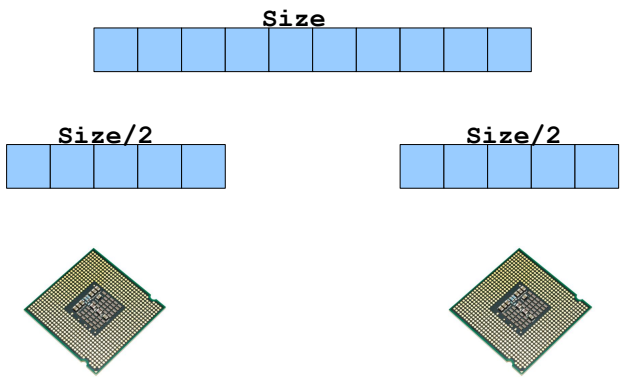
Sommario

- Merge Sort
- Ordinamento STL
- Gestione Liste
- Esercizi

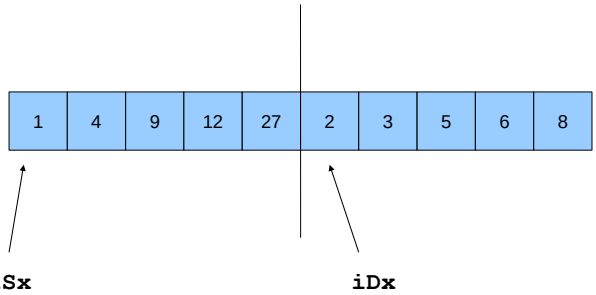
A metà



A metà



Unire gli array



combina

```
1  void combina( int arr[] , int start , int mid , int end )
2  {
3      // init Variabili di stato + buffer appoggio
4
5      while(1)
6      {
7          // se arr[iSx] più piccolo
8          {
9              // Inserisco arr[iSx]
10
11         }
12         // se arr[iDx] più piccolo
13         {
14             // Inserisco arr[iDx]
15
16         }
17     }
18
19
20 }
```

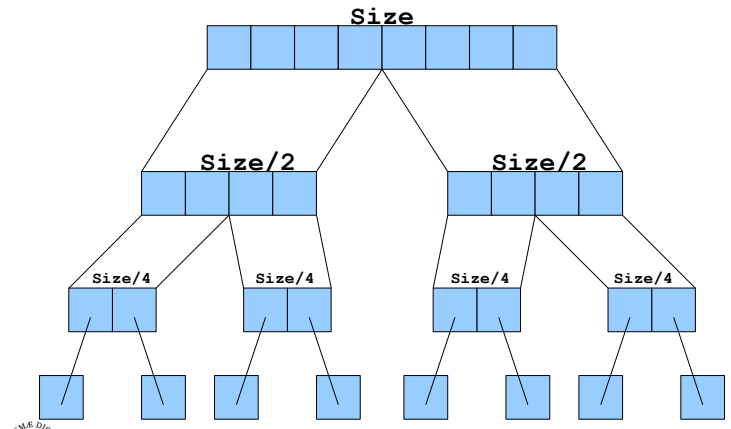

combina

```
1  void combina( int arr[] , int start , int mid , int end )
2  {
3      int iSx = start , iDx = mid; // stato
4      std::vector<int> tempResult; // buffer
5      while(1)
6      {
7          if(arr[iSx] < arr[iDx])
8          {
9              tempResult.push_back(arr[iSx++]);
10             // CONDIZIONE USCITA
11         }
12         else
13         {
14             tempResult.push_back(arr[iDx++]);
15             // CONDIZIONE USCITA
16         }
17     }
18     // GESTISCO ULTIMI
19     // RICOPIO da buffer a arr
   }
```



```
1    2    5    6    8    12   18   26   78
3    6    9    99   100  120  150  168  300
sx 1
sx 2
dx 3
sx 5
dx 6
sx 6
sx 8
dx 9
sx 12
sx 18
sx 26
sx 78
1    2    3    5    6    6    8    9    _ 12
18   26   78   99   100  120  150  168  300
```

divide



Lista ordinata Triviale

Divide , Conquer , Combine

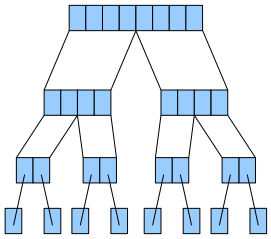
```
1      conquer ( int * arr , int start , int end )
2  {
3      int mid;
4      if( start<end )
5      {
6          mid = (start+end)/2; // DIVIDE
7          conquer( arr , start , mid ); // CONQUER
8          conquer( arr , mid+1 , end ); // CONQUER
9          combina( arr , start , mid+1 , end );
10     }
11 }
12
```

Divide , Conquer , Combine

```
1  void mergeSort( int * arr , int start , int end )
2  {
3      int mid;
4      if( start<end )
5      {
6          mid = (start+end)/2; // DIVIDE
7          mergeSort( arr , start , mid ); // CONQUER
8          mergeSort( arr , mid+1 , end ); // CONQUER
9          combina( arr , start , mid+1 , end );
10     }
11 }
```


Complessità mergesort

- elementi n
- Livelli $log\,(n) + 1$
- Costo livello n



$$n\,(log\,(n) + 1) \longrightarrow n\,log\,(n) + n$$

Complessità mergesort

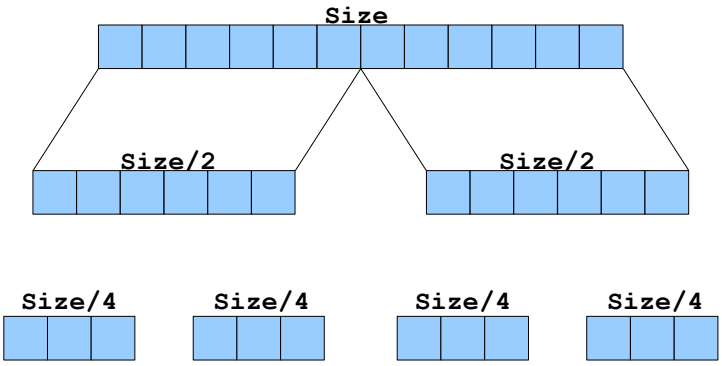
$$\Theta\,(n\,log\,(n))$$

BEST CASE

WORST CASE

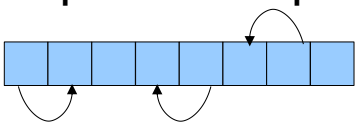
	Worst Case	Best Case	Average Case
Merge Sort	$\Theta\,(n\,log\,n)$	$\Theta\,(n\,log\,n)$	$\Theta\,(n\,log\,n)$
Insertion Sort	$\Theta\,(n^2)$	$\Theta\,(n)$	$\Theta\,(n^2)$

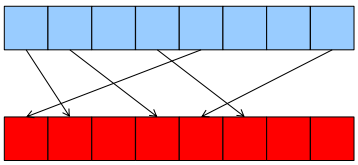
Ibrido



Complessità?

- Tempo di esecuzione: **worst** vs **best** vs **avg**
- Memoria: **in-place** or **not in-place**?

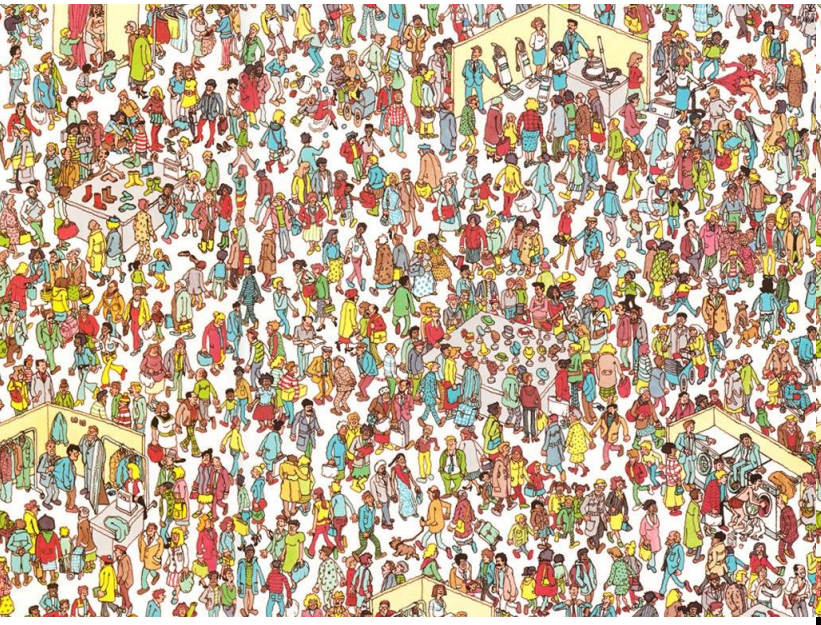




Test

- Insertionsort
- Mergesort
- Casi
  - Random
  - Ordinata
  - Inversa
- Variare quantità linearmente (10 , 40 , 80…)

Where is Wally?



Find Bug-Wally

```
1  [..]
2       int iSx = start , iDx = mid;
3      int stop , iRim;
4      std::vector<int> tempResult;
5      while(1)
6      {if(arr[iSx] < arr[iDx]);
7          {tempResult.push_back(arr[iSx++]);
8              if(iSx == mid)
9              { iRim = iDx;
10               stop = end;
11               break;}
12             continue;
13         }if(arr[iSx] >= arr[iDx])
14         {tempResult.push_back(arr[iDx++]);
15             if(iDx == end+1)
16             { iRim = iSx;
17               stop = mid;
18               break;
19             }
20 [..]
```

```
start
merge sort
33      36      27      15      43      35      36      42      49      21      12      27      40      9       13
6       11      18      17      29      32      30      12      23      17      35      29      2       22      8
1       42      29      23      21      19      34      37      48      24      15      20      13      26      41
0       46      31      5       25      34      27      36      5       46      29      13      7       24      45
4       14      43      0       37      8       26      28      38      34      3       1       4       49      32
2       26      36      44      39
```

Compiler Flags

- g++ **-W** -o test test.cpp
- g++ **-Wall** -W -o test test.cpp

```
start
merge sort
33      36      27      15      43      35      36      42      49      21      12      27      40      9       13
6       11      18      17      29      32      30      12      23      17      35      29      2       22      8
1       42      29      23      21      19      34      37      48      24      15      20      13      26      41
0       46      31      5       25      34      27      36      5       46      29      13      7       24      45
4       14      43      0       37      8       26      28      38      34      3       1       4       49      32
2       26      36      44      39
kruviser@ilMioComputer:~/Dropbox/lezioni algoritmi/lezione 2$ g++ -W -o testMergeSortBug testMergeSortBug.cpp
testMergeSortBug.cpp: In function 'void combina(int*, int, int, int)':
testMergeSortBug.cpp:135:32: warning: suggest braces around empty body in an 'if' statement [-Wempty-body]
```

Warning: suggest braces around empty body in an 'if' statement

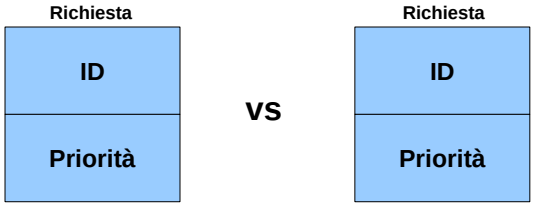
Find Bug-Wally

```
1   [..]
2        int iSx = start , iDx = mid;
3       int stop , iRim;
4       std::vector<int> tempResult;
5       while(1)
6       {if(arr[iSx] < arr[iDx]);
7           {tempResult.push_back(arr[iSx++]);
8               if(iSx == mid)
9               { iRim = iDx;
10                stop = end;
11                break;}
12              continue;
13          }if(arr[iSx] >= arr[iDx])
14          {tempResult.push_back(arr[iDx++]);
15              if(iDx == end+1)
16              { iRim = iSx;
17                stop = mid;
18                break;
19              }
20  [..]
```


Find Bug-Wally

```
1   [..]
2        int iSx = start , iDx = mid;
3       int stop , iRim;
4       std::vector<int> tempResult;
5       while(1)
6       {if(arr[iSx] < arr[iDx]);
7           {tempResult.push_back(arr[iSx++]);
8               if(iSx == mid)
9               { iRim = iDx;
10                stop = end;
11                break;}
12              continue;
13          }if(arr[iSx] >= arr[iDx])
14          {tempResult.push_back(arr[iDx++]);
15              if(iDx == end+1)
16              { iRim = iSx;
17                stop = mid;
18                break;
19              }
20  [..]
```


Ordinamenti multi-valore



- Richieste servite in ordine di ID crescente
- A parità di ID, si serve in ordine di priorità decrescente

Richiesta

```
1   struct Richiesta
2   {
3       int id_;
4       int prio_;
5
6       // costruttore con lista init
7       Richiesta(int id, int prio):
8                 id_(id) , prio_(prio){}
9   };
10
11
```

STL: sort()

```
sort ( first, last, comparatore );
```

Estremi del vettore da ordinare

Funzione di confronto
- True
- False

Richiesta

```
1  bool confrontaRichieste( Richiesta r1 , Richiesta r2)
2  {
3      // SE ID1 < ID2
4         // VINCE 1
5
6      // SE ID1 == ID2
7      {
8          // SE PRIO1 > PRIO2
9              // VINCE 1
10
11
12     }
13     // TUTTI GLI ALTRI CASI
14         // VINCE 2
15 }
```


Richiesta

```
1  bool confrontaRichieste( Richiesta r1 , Richiesta r2)
2  {
3      if( r1.id_<r2.id_ )
4          return true;
5
6      else if(r1.id_ == r2.id_)
7      {
8          if(r1.prio_>r2.prio_)
9              return true;
10
11             ?
12     }
13     else
14         return false;
15 }
```


Richiesta

```
1  bool confrontaRichieste( Richiesta r1 , Richiesta r2)
2  {
3      if( r1.id_<r2.id_ )
4          return true;
5
6      else if(r1.id_ == r2.id_)
7      {
8          if(r1.prio_>r2.prio_)
9              return true;
10         else
11             return false;
12     }
13     else
14         return false;
15 }
```

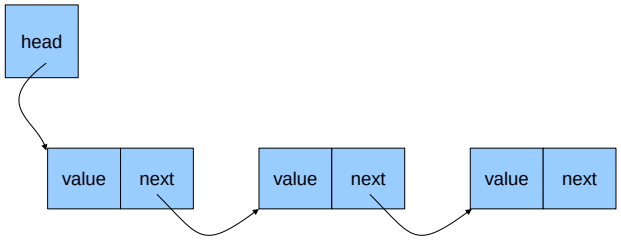

Tipo Accessi

vs

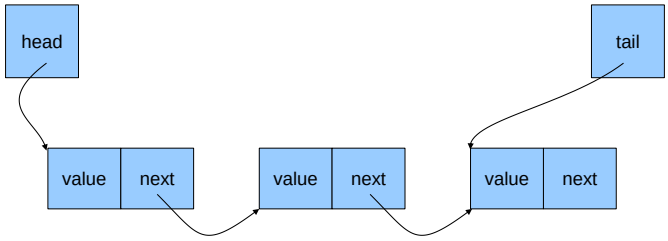
Tipo Accessi

vs

liste



liste



Solo inserimento in coda

Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      // LEGGO LUNGHEZZA
4
5
6      // VARIABILI DI APPOGGIO
7
8      // PER TUTTA LA LUNGHEZZA
9      {
10         // LEGGO VALORE
11
12         // CREO E INIZIALIZZO OGGETTO
13
14
15         // AGGIORNO TESTA
16     }
17     // RITORNO TESTA
18 }
```

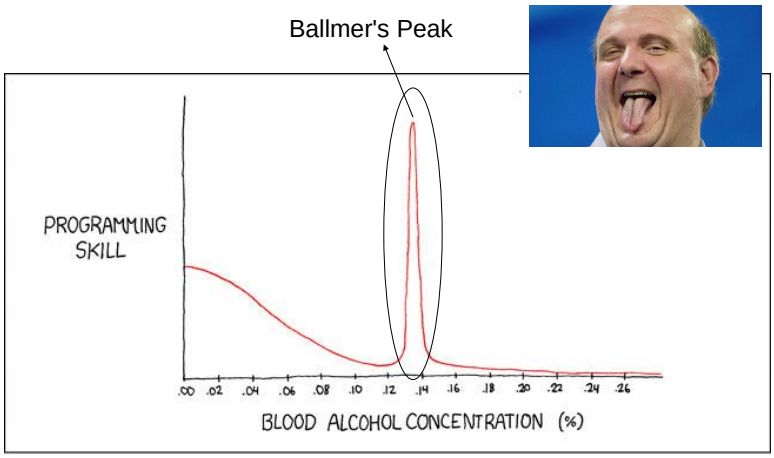

Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      int value , l;
4      cin >> l;
5
6      Obj * head , * newObj;
7
8      for( int i = 0 ; i < l ; ++i )
9      {
10         cin >> value;
11         newObj = new Obj();
12         newObj->next_ = head;
13         newObj->value_ = value;
14
15         head = newObj;
16     }
17     return head;
18 }
```


Stampa Lista

```
1  void stampaLista( Obj * head )
2  {
3      Obj * pointer = head;
4      while( pointer != NULL )
5      {
6          cout << pointer->value_ << endl ;
7          pointer = pointer->next_;
8      }
9      cout << endl;
10 }
11
12
```


Birra!



Fonte: https://xkcd.com/323/


```
Hello world

Segmentation fault
```

```
kruviser@ilMioComputer:~/Dropbox/lezioni algoritmi/lezione 2$ ./testList < listFile
letto 10
5       1       26      8       12      78      6       2       18      3
3
18
2
6
78
12
8
26
1
5
Segmentation fault (core dumped)
```

Valgrind

- Babysitter Memoria
- Controlla accessi
- Conta accessi


```
valgrind ./eseguibile
```

```
12
8
26
1
5
==3307== Conditional jump or move depends on uninitialised value(s)
==3307==    at 0x8048719: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==
==3307== Use of uninitialised value of size 4
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==
==3307== Invalid read of size 4
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==  Address 0xffff is not stack'd, malloc'd or (recently) free'd
==3307==
==3307==
==3307== Process terminating with default action of signal 11 (SIGSEGV)
==3307==  Access not within mapped region at address 0xFFFF
==3307==    at 0x80486E5: stampaLista(Obj*) (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==    by 0x804883D: main (in /home/kruviser/Dropbox/lezioni algoritmi/lezione 2/testList)
==3307==  If you believe this happened as a result of a stack
==3307==  overflow in your program's main thread (unlikely but
==3307==  possible), you can try to increase the size of the
==3307==  main thread stack using the --main-stacksize= flag.
==3307==  The main thread stack size used in this run was 8388608.
==3307==
==3307== HEAP SUMMARY:
==3307==     in use at exit: 80 bytes in 10 blocks
==3307==   total heap usage: 10 allocs, 0 frees, 80 bytes allocated
==3307==
==3307== LEAK SUMMARY:
==3307==    definitely lost: 0 bytes in 0 blocks
==3307==    indirectly lost: 0 bytes in 0 blocks
==3307==      possibly lost: 0 bytes in 0 blocks
==3307==    still reachable: 80 bytes in 10 blocks
```

```
g++ -g -o eseguibile eseguibile.cpp

valgrind ./eseguibile
```

```
12
8
26
1
5
==3288== Conditional jump or move depends on uninitialised value(s)
==3288==    at 0x8048719: stampaLista(Obj*) (testList.cpp:21)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==
==3288== Use of uninitialised value of size 4
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==
==3288== Invalid read of size 4
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==  Address 0xffff is not stack'd, malloc'd or (recently) free'd
==3288==
==3288==
==3288== Process terminating with default action of signal 11 (SIGSEGV)
==3288==  Access not within mapped region at address 0xFFFF
==3288==    at 0x80486E5: stampaLista(Obj*) (testList.cpp:23)
==3288==    by 0x804883D: main (testList.cpp:58)
==3288==  If you believe this happened as a result of a stack
==3288==  overflow in your program's main thread (unlikely but
==3288==  possible), you can try to increase the size of the
==3288==  main thread stack using the --main-stacksize= flag.
==3288==  The main thread stack size used in this run was 8388608.
==3288==
==3288== HEAP SUMMARY:
==3288==     in use at exit: 80 bytes in 10 blocks
==3288==   total heap usage: 10 allocs, 0 frees, 80 bytes allocated
==3288==
==3288== LEAK SUMMARY:
==3288==    definitely lost: 0 bytes in 0 blocks
==3288==    indirectly lost: 0 bytes in 0 blocks
==3288==      possibly lost: 0 bytes in 0 blocks
==3288==    still reachable: 80 bytes in 10 blocks
```

Stampa Lista

File testList.cpp

```
18 void stampaLista( Obj * head )
19 {
20     Obj * pointer = head;
21     while( pointer != NULL )
22     {
23         cout << pointer->value_ << endl ;
24         pointer = pointer->next_;
25     }
26     cout << endl;
27 }
28
29
```

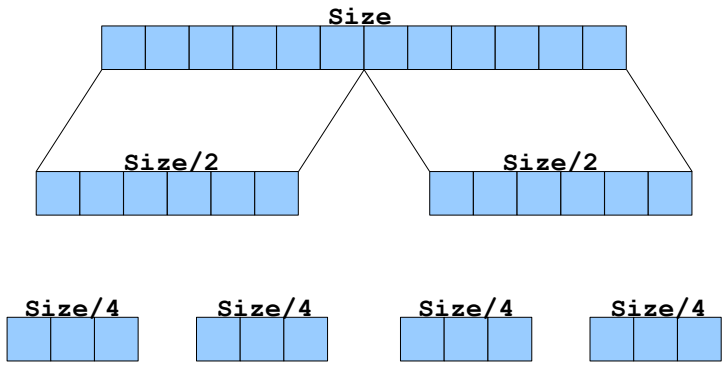

Lettura su Lista

```
1  Obj * leggiInput()
2  {
3      int value , l;
4      cin >> l;
5
6      Obj * head , * newObj;
7
8      for( int i = 0 ; i < l ; ++i )
9      {
10         cin >> value;
11         newObj = new Obj();
12         newObj->next_ = head;
13         newObj->value_ = value;
14
15         head = newObj;
16     }
17     return head;
18 }
```


Operazioni su Lista

- Ricerco un elemento e lo sposto in testa
  - Scorrere
  - Estrazione
  - Inserzione testa
- Ricerco un elemento e lo sposto in coda
  - Scorrere
  - Estrazione
  - Inserimento in coda...

Merge Sort Ibrido

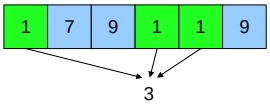


ESERCIZI:

Distinti in Array

- Input: elementi array
- Output: array senza duplicati

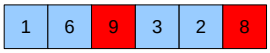
K interi più frequenti



- Input: elementi array , intero k
- Output: primi k valori più frequenti

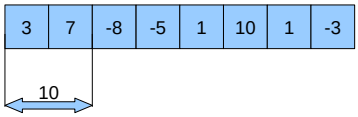
K interi più grandi

K=2

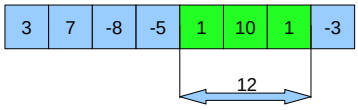


- Input: elementi array , intero k
- Output: primi k valori ordinati in maniera decrescente

Somma Massima



- Input: array
- Output: somma massima
- Esempio



Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3
4
5
6      for(i=0; i<size; i++)                    // n
7      {
8
9
10
11
12
13
14
15
16
17     }
18     return max;
   }
```


Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3
4
5
6      for(i=0; i<size; i++)                    // n
7      {
8          for(j=i; j<size; j++)                // n
9          {
10
11
12
13
14
15
16         }
17     }
18     return max;
   }
```


Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3      int somma;
4      int i,j,k;
5      int max=a[0];
6      for(i=0; i<size; i++)                    // n
7      {
8          for(j=i; j<size; j++)                // n
9          {
10             somma=0;
11             for(k=i; k<=j; k++)              // n
12             {
13                 somma+=a[k];
14             }
15             if(somma > max) max=somma;
16         }
17     }
18     return max;
   }
```

Soluzione 1

```
1  int somme1(int a[] , int size )
2  {
3      int somma;
4      int i,j,k;
5      int max=a[0];
6      for(i=0; i<size; i++)                  // n
7      {
8          for(j=i; j<size; j++)              // n
9          {
10             somma=0;
11             for(k=i; k<=j; k++)            // n
12             {
13                 somma+=a[k];
14             }
15             if(somma > max) max=somma;
16         }                                  Θ ( n^3 )
17     }
18     return max;
   }
```


Soluzione 2

```
1  int somme2(int a[] , int size )
2  {
3      int somma;
4      int i,j;
5      int max=a[0];
6      for(i=0; i<size; i++)
7      {
8          somma=0;
9          for(j=i; j<size; j++)
10         {
11             somma+=a[j];
12             if(somma > max) max=somma;
13         }
14     }
15     return max;
16 }
17
18
```


Soluzione 2

```
1  int somme2(int a[] , int size )
2  {
3      int somma;
4      int i,j;
5      int max=a[0];
6      for(i=0; i<size; i++)                  // n
7      {
8          somma=0;
9          for(j=i; j<size; j++)              // n
10         {
11             somma+=a[j];
12             if(somma > max) max=somma;
13         }
14     }
15     return max;                            Θ ( n^2 )
16 }
17
18
```


Esercizi

- Esperimenti
  - Merge vs Insertion sort vs Ibrido
  - Soluzioni array somma massima
  - Input critici (array inverso, array ordinato)
- Esercizi
  - Inserimenti testa/coda liste
  - Distinti
  - Più frequenti
  - Più grandi

Algoritmi e Strutture Dati

Lezione 3

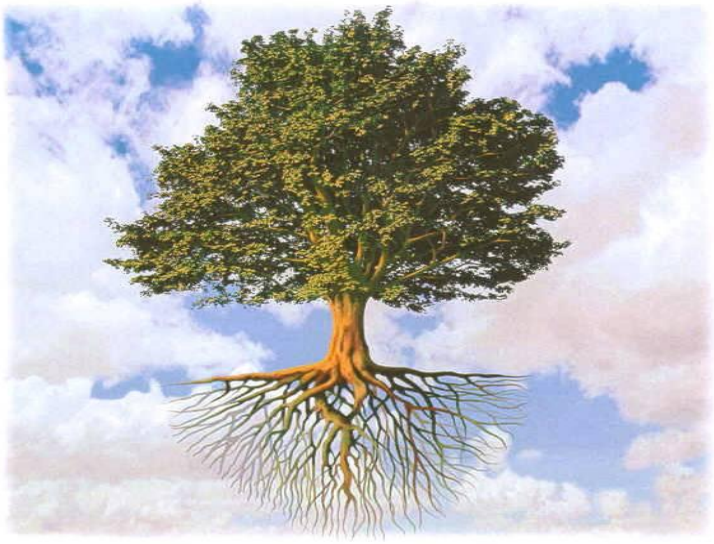
www.iet.unipi.it/a.virdis

Antonio Virdis

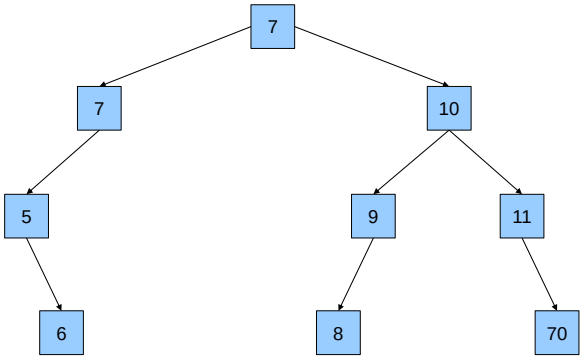
antonio.virdis@unipi.it

Sommario

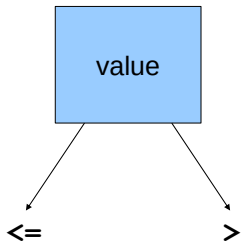
- Alberi Binari di Ricerca
- Gestione Stringhe
- Progettazione
- Esercizi



Alberi Binari



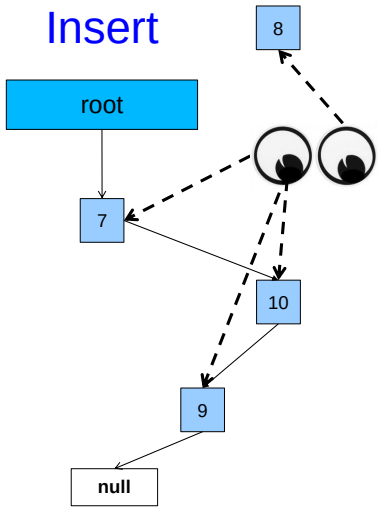
Alberi Binari di Ricerca



binTree

```
1  struct Node
2  {
3      int value;
4      Node * left;
5      Node * right;
6
7      Node(int val):
8          value(val) , left(NULL) , right(NULL) {}
9  };
10
11 class BinTree
12 {
13     Node * root_;
14
15 public:
16
17     BinTree() { root_ = NULL; }
18
19     Node * getRoot() { return root_; }
20 }
```


Insert



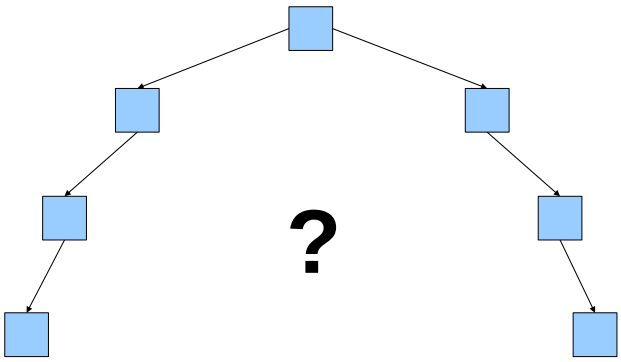
Insert

```
1  void insert( int val )
2  {
3      // inizializzo nuovo elemento          INIZIALIZZAZIONE
4      // inizializzo variabili appoggio
5
6
7      // finchè non arrivo ad una foglia
8      {
9          // aggiorno variabili
10         // se <=                           INDIVIDUO
11             // vado a sinistra             POSIZIONE
12         // altrimenti
13             // vado a destra
14     }
15     // se albero vuoto                     INSERIMENTO
16         // aggiorno radice
17
18     // decido se diventare figlio left o right
19
20
21 }
```

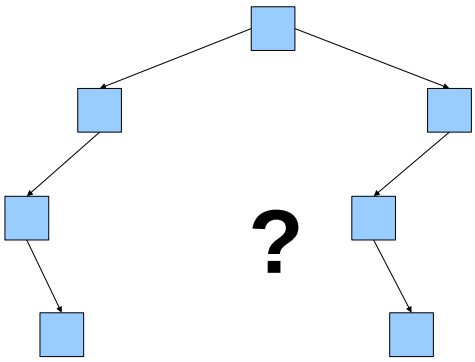

Insert

```
1  void insert( int val )
2  {
3      Node * node = new Node(val);            INIZIALIZZAZIONE
4      Node * pre = NULL;
5      Node * post = root_;
6
7      while( post != NULL )
8      {
9          pre = post;
10         if( val <= post->value )            INDIVIDUO
11             post = post->left;              POSIZIONE
12         else
13             post = post->right;
14     }
15     if( pre == NULL )                       INSERIMENTO
16         root_ = node;
17     else if( val <= pre->value )
18         pre->left = node;
19     else
20         pre->right = node;
21 }
```


min & MAX



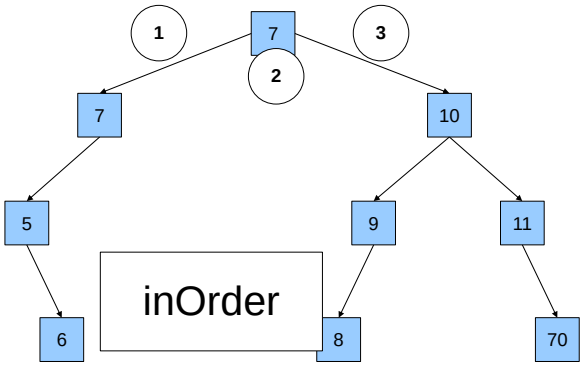
min & MAX



Min/Max

```
1  Node * min()
2  {
3      Node * temp = root_;
4      while( temp->left != NULL )
5          temp = temp->left;
6      return temp;
7  }
8
9
10 Node * max()
11 {
12     Node * temp = root_;
13     while( temp->right != NULL )
14         temp = temp->right;
15     return temp;
16 }
17
18
19
20
```


visite



In-Order

```
1
2
3
4  void inOrder( Node * tree )
5  {
6      // se non sono in una foglia
7      {
8
9          // visito verso left
10
11         // stampo questo valore
12
13         // visito verso right
14
15     }
16 }
17
18
19
20
```


In-Order

```
1
2
3
4  void inOrder( Node * tree )
5  {
6      if(tree!=NULL)
7      {
8
9          visitaNodo(tree->left);
10
11         cout << tree->value << "\t";
12
13         visitaNodo (tree->right);
14
15     }
16 }
17
18
19
20
```


In-Order

```
1
2
3
4  void inOrder( Node * tree )
5  {
6      if(tree!=NULL)
7      {
8
9          inOrder(tree->left);
10
11         cout << tree->value << "\t";
12
13         inOrder(tree->right);
14
15     }
16 }
17
18
19
20
```


Sort vs BinTree

- Albero alto: **log(n)**
- Inserimento: **n * log(n)**
- Sort/visita: **n**

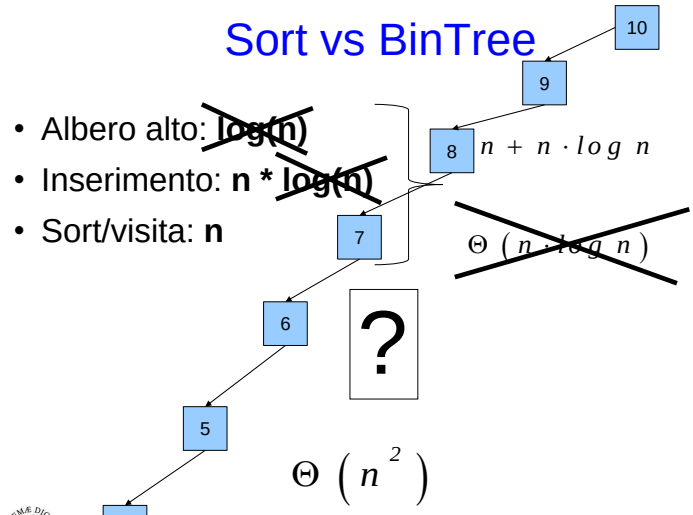
$$n + n \cdot log\ n$$

$$\Theta\left(n \cdot log\ n\right)$$

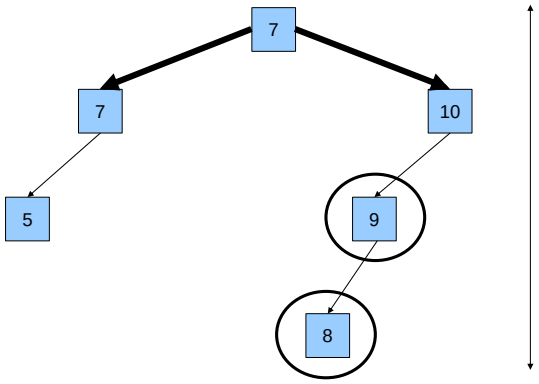
giusto?

Sort vs BinTree

- Albero alto: ~~**log(n)**~~
- Inserimento: ~~**n * log(n)**~~
- Sort/visita: **n**



Height



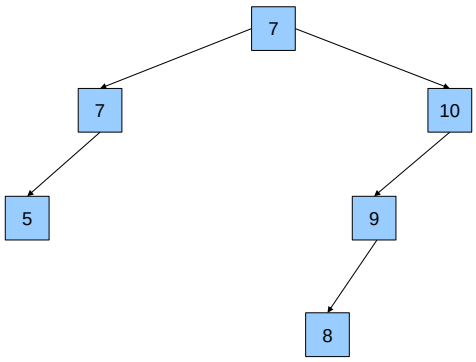
Altezza albero

```
1  int height( Node * tree )
2  {
3      int hLeft;
4      int hRight;
5
6      if( tree == NULL )
7          return 0;
8
9      hLeft = height(tree->left);
10
11     hRight = height(tree->right);
12
13     return 1 +  max(hLeft,hRight);
14 }
15
16
17
18
19
20
```


Trova chiave

- Dato
  - Un albero binario con valori distinti
  - Un valore K
- Trovare
  - Se il valore esiste

Search K=9



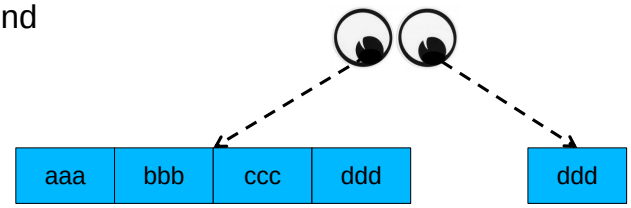
Search

```
1  bool search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return false;
5
6     bool found;
7
8      if( tree->value == val )
9          return true;
10
11     else if( val <= tree->value )
12        found = search( tree->left , val );
13     else
14        found = search( tree->right , val );
15
16     return found;
17 }
18
19
20
```

INDIVIDUO DIREZIONE

stringhe

- Creazione
- Concatenazione
- Compare
- Find



Stringhe

```
1   #include <string>
2
3   String parola = "liste";
4
5   String frase = "mi piacciono le liste";
6
7   String parola2 = "non ";
8
9   String frase2 = parola2 + frase;
10
11  frase.find(parola);
12  // se fallisce -> string::npos
13
14  parola.compare(parola2);
```

http://www.cplusplus.com/reference/string/string/

Esercizio Stringhe

- Input
  - Una testo T formato da più parole
  - Un insieme S di N parole
- Output:
  1. Le parole di S *contenute* in T, ordinate per posizione in T (insieme R1)
  2. Le parole di S *non contenute* in T, in ordine lessicografico (insieme R2)

Analisi

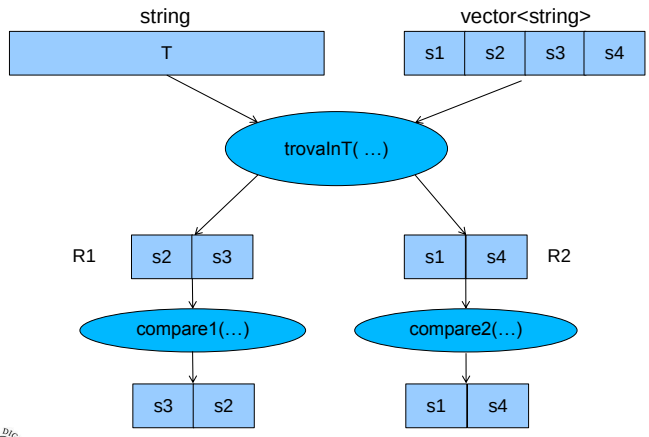
Strutture Dati:

- Dove salvo T?
- Dove salvo S?

Operazioni:

- Come ottengo gli elementi di 1?
- Come ottengo gli elementi di 2?
- Come ordino 1?
- Come ordino 2?

Analisi (2)



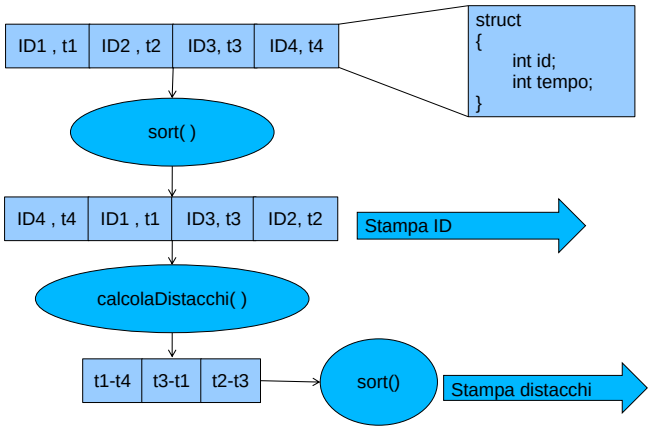
Implementazione

```
1   // cerca stringhe di S dentro T
2   void trovaInT( … ){   }
3
4   // implementa confronto per posizione
5   bool compare1(string a, string b){  }
6
7   // implementa confronto lessicografico
8   bool compare2(string a, string b){  }
9
10  int main()
11  {
12      string T;
13      vector <string> S, R1, R2;
14
15      // lettura T ed S
16
17      trovaInT( … );
18      sort( R1.begin(), R1.end(), compare1 )
19      sort( R2.begin(), R2.end(), compare2 )
20      print();
}
```

Gara

- Ad una gara partecipano N concorrenti
- Ogni concorrente e' caratterizzato da:
  - Un ID intero
  - Un tempo di arrivo espresso in secondi
- Calcolare:
  - Classifica
  - K distacchi più ampi di utenti **consecutivi**

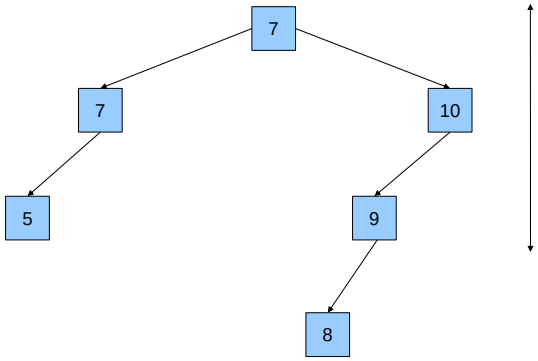
Analisi



Trovare altezza chiave K

- Input
  - Una sequenza di N interi positivi
  - Chiave K
- Output
  - L'altezza della chiave K dentro l'albero (se esiste)

Search K=9



Altezza

```
1  int search( Node * tree , int val )
2  {
3      if( tree == NULL )
4          return 0;
5
6      int cont = 0;
7      if( tree->value == val )
8          return 1;
9
10     else if( val <= tree->value )
11         cont = search( tree->left , val );
12 SEARCH
13 else
14         cont = search( tree->right , val );
15
16     if( cont != 0 )
17         return cont+1;
18 HEIGHT
19     else return 0;
20 }
```


Esercizi

- Esperimenti
  - Sort vs Binary Tree
  - Sort vs Min/Max
- Esercizi
  - Visite pre- e post-order
  - Esercizi di progettazione

Algoritmi e Strutture Dati

Lezione 4

www.iet.unipi.it/a.virdis

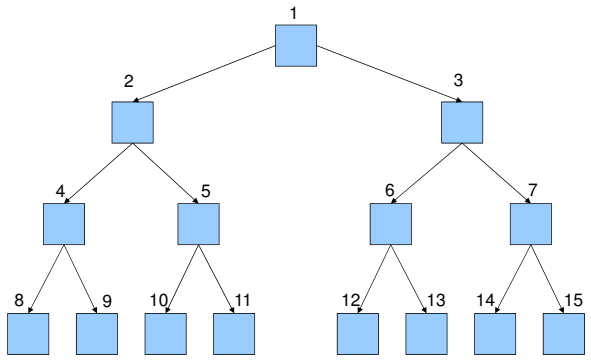
Antonio Virdis

a.virdis@iet.unipi.it

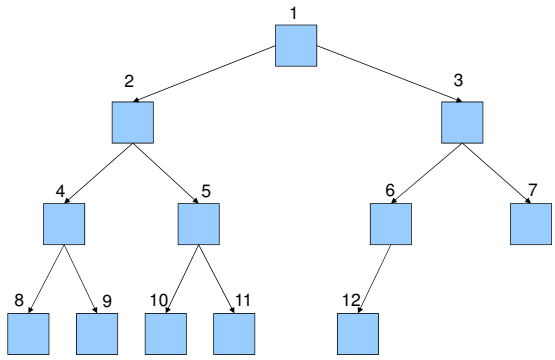
Sommario

- Heap
- Ordinamento tramite Heap
- Soluzioni
- Esercizi

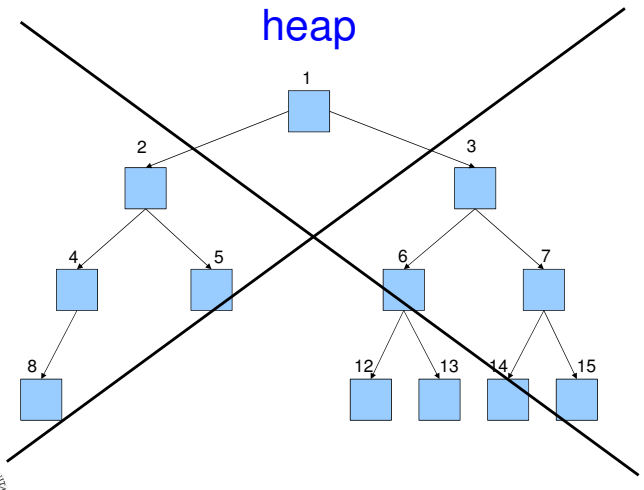
heap



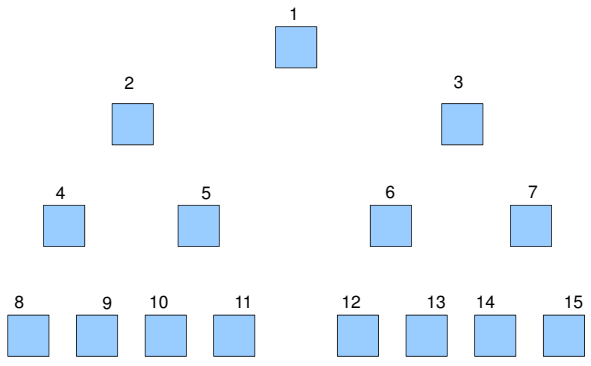
heap



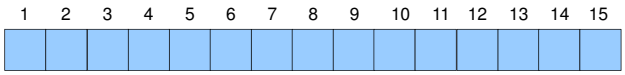
heap



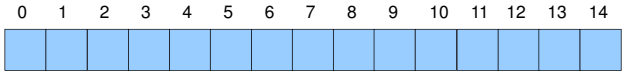
heap



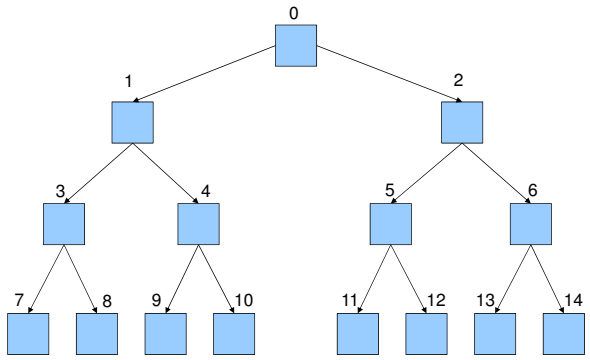
heap



heap



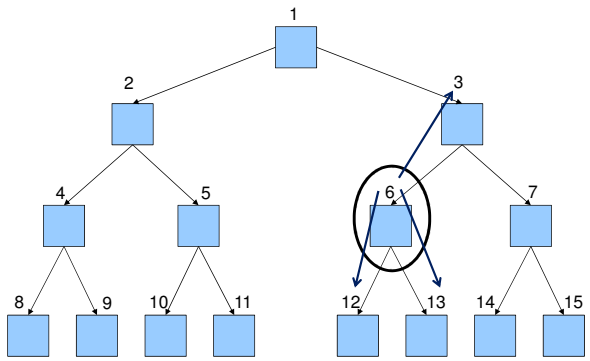
heap



heap

```
1   class Heap
2   {
3       std::vector<int> data_;
4
5       int length_;     // lunghezza array
6       int size_;   // dimensione Heap
7
8   public:
9       Heap() {};
10
11      void fill( int l );
12      void printVector();
13
14     ...
15  }
```

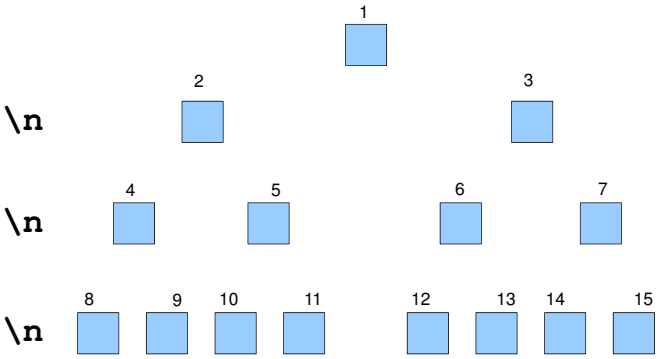

heap



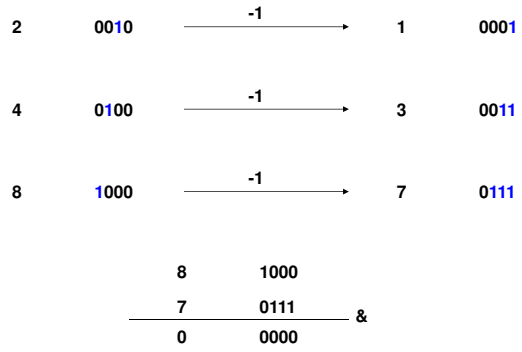
heap

```
1   int parent(int i)
2   {
3       return floor((i-1)/2);    // floor(i/2)
4   }
5
6   int getLeft(int i)
7   {
8       return (i*2) + 1;         // i*2
9   }
10
11  int getRight(int i)
12  {
13      return (i*2)+2;           // (i*2)+1
14  }
15
16
17
18
```

stampa



stampa



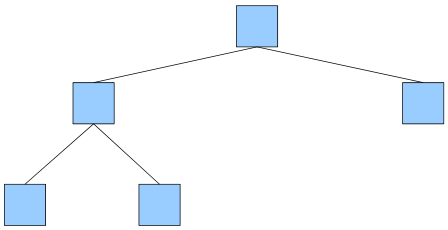
Print

```
1  bool isFirstChild( int i )
2  {
3      if( ( i!=0 ) && ( (i&(i-1)) == 0) )
4          return true;
5      else
6          return false;
7  }
8
9
10
11
12
13
14
15
16
17
18
```

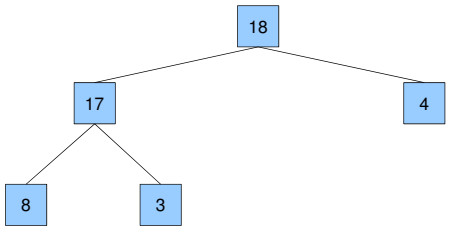

Print

```
1  bool isFirstChild( int i )
2  {
3      if( ( i!=0 ) && ( (i&(i-1)) == 0) )
4          return true;
5      else
6          return false;
7  }
8
9  void print()
10 {
11     for( int i=0 ; i < length_ ; ++i )
12     {
13         if( isFirstChild(i+1) )
14             cout << endl;
15         cout << data_[i] << "\t";
16     }
17     cout << endl;
18 }
```

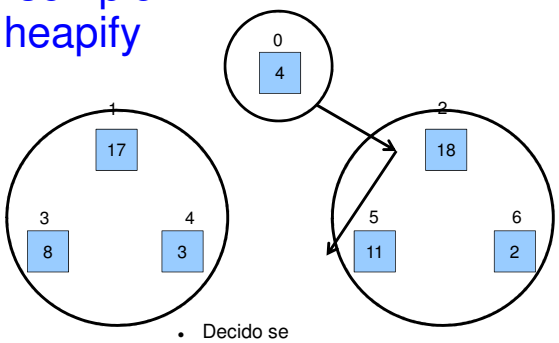

Heap Property



Heap Property



Esempio heapify



- Decido se
  - già ok?
  - andare a destra
  - andare a sinistra

heapify

```
1  void maxHeapify(int i)
2  {
3      // ottengo left e right
4
5
6
7      // (se ho figlio left) AND (left > i)
8          // left é più grande
9      // altrimenti
10         // i é più grande
11
12     // (se ho figlio right) AND (right > largest)
13         // right è più grande
14
15     // se i viola la proprietà di max-heap
16     {
17         // scambio i e il più grande
18         // controllo se l'albero che ho cambiato va bene
19     }
20 }
```


heapify

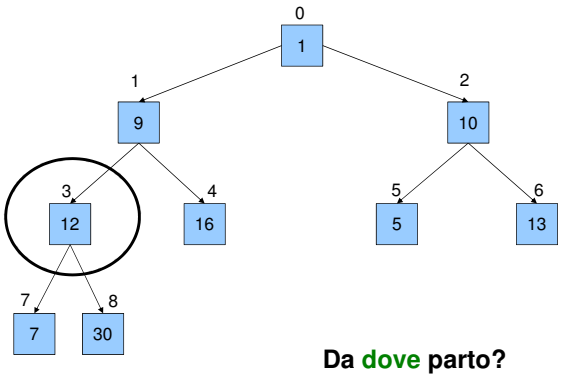
```
1  void maxHeapify(int i)
2  {
3      int left = getLeft(i);
4      int right = getRight(i);
5      int largest;
6
7      if((left < size_)&&(data_[left] > data_[i]))
8          largest = left;
9      else
10         largest = i;
11
12     if((right < size_)&&(data_[right] > data_[largest]))
13         largest = right;
14
15     if( largest != i )
16     {
17         scambia(i,largest);
18         maxHeapify(largest);
19     }
20 }
```

inizializzazione

Identifico + grande

Aggiorno albero

Build Heap

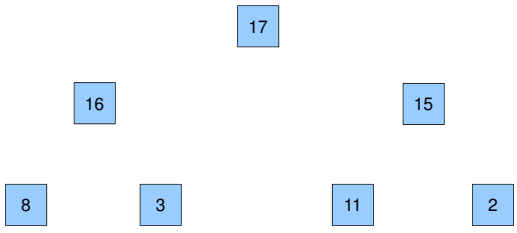


Da dove parto?

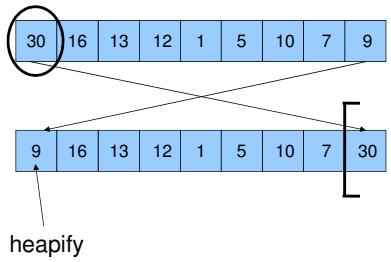
Build Heap

```
1  void buildMaxHeap()
2  {
3      size_ = length_;
4
5      int i = floor(length_/2)-1
6
7      for( ; i>=0 ; --i )
8      {
9          maxHeapify(i);
10         print();
11     }
12 }
13
14
15
16
17
18
19
20
```


Utilizzo

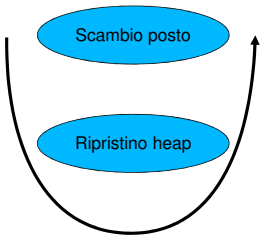


Esempio heapsort



heapsort

```
1  void heapSort()
2  {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 }
20
```



heapsort

```
1  void heapSort()
2  {
3
4      int i = length_-1
5
6      for( ; i>0 ; --i)
7      {
8
9
10         scambia(0,i);
11
12
13
14         --size_;
15
16         maxHeapify(0);
17     }
18
19 }
20
```


Programma completo

```
1  int main()
2  {
3      Heap hp;
4
5      hp.fill();
6      hp.print();
7
8
9      hp.buildMaxHeap();
10     hp.print();
11
12     hp.heapSort();
13     hp.printArray();
14
15     return 0;
16
17 }
18
```


Esercizi (per casa)

- Aggiunta nodo
- Eliminazione nodo
- Aumento Valore

Heap STL

```
#include <algorithm>
make_heap( inizio , fine )
pop_heap( inizio , fine )

#include <queue>
priority_queue<int> prioQ
prioQ.push(val)
prioQ.top()
prioQ.pop()
```

Algorithms

```
1  #include <vector>
2  #include <algorithm>
3
4
5  vector<int> vect;
6
7  for( int i = 0 ; i<quanti ; ++i )
8  {
9      cin >> val;
10     vect.push_back(val);
11 }
12
13 make_heap(vect.begin(),vect.end());
14
15 while(!vect.empty())
16 {
17     cout << "top " << *vect.begin() << endl;
18     pop_heap(vect.begin(),vect.end());
19     vect.pop_back();
20 }
```

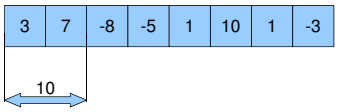

priority_queue

```
1  #include <queue>     // std::priority_queue
2
3  priority_queue<int> prioQ;
4
5  for( int i = 0 ; i<quanti ; ++i )
6  {
7      cin >> val;
8      prioQ.push(val);
9  }
10
11 while(!prioQ.empty())
12 {
13     cout << "top " << prioQ.top() << endl;
14     prioQ.pop();
15 }
16
17
18
19
20
```

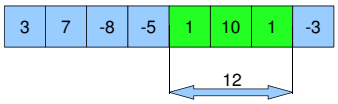

Esercizi

- Esperimenti
  - Utilizzo Heap fatto a mano
  - Heapsort VS MergeSort
  - Priority_queue

Somma Massima

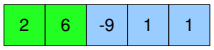


- Input: array
- Output: somma massima
- Esempio

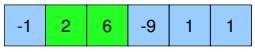


proprietà

la somma degli elementi del sotto array di somma massima è sempre positiva



Il valore precedente al primo valore del sotto array di somma massima è negativo

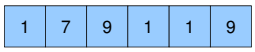


Soluzione 3

```
1  int somme3(int a[] , int size )
2  {
3      int somma;
4      int i;
5      int max=a[0];
6      somma = 0;
7      for(i=0; i<size; i++)
8      {
9          if(somma > 0) somma+=a[i];
10         else somma=a[i];
11
12         if(somma > max) max=somma;
13     }
14     return max;
15 }
```

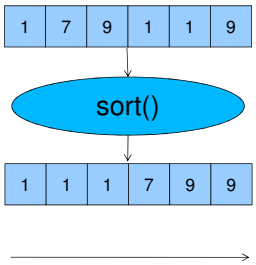
$\Theta(n)$

Distinti in Array

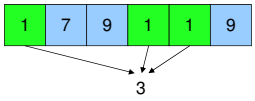


- Input: elementi array
- Output: array senza duplicati

Distinti in Array (2)

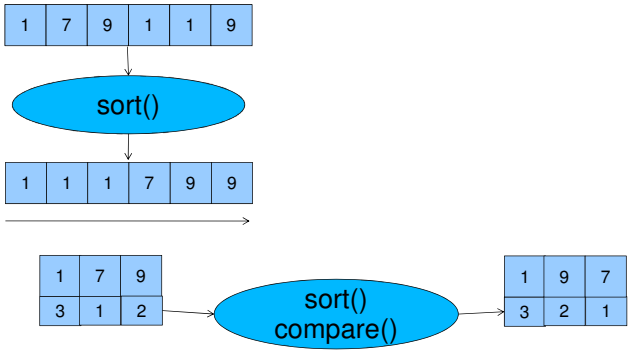


K interi più frequenti



- Input: elementi array , intero k
- Output: primi k valori più frequenti

K interi più frequenti



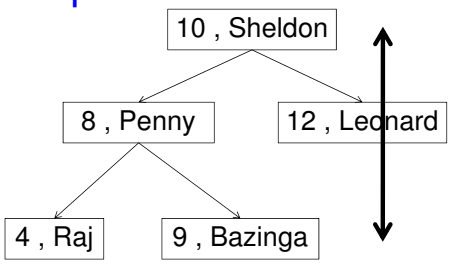
Albero Binario a etichette complesse

- Input:
  - Un intero **N**
  - Un intero **H**
  - N coppie **[intero,stringa]**
- Operazioni:
  - Inserire le N coppie in un albero binario di ricerca (usando il valore intero come chiave)
- Output:
  - stringhe che si trovano in nodi ad altezza H, stampate in ordine lessicografico

Albero Binario a etichette complesse

input

```
5
2
10
Sheldon
12
Leonard
8
Penny
4
Raj
9
Bazinga
```



output

```
Bazinga
Raj
```

Analisi

- Input:
  - Un intero N
  - Un intero H
  - N coppie [intero,stringa]
- Operazioni:
  - Inserire le N coppie in un albero binario di ricerca
- Output:
  - stringhe che si trovano in nodi ad altezza H, stampate in ordine lessicografico

Analisi

Implementare struttura dati che supporti
- Albero binario
- Etichette multi valore

```
struct node
{
        int key;
        string str;
        struct node* right;
        struct node* left;
} Node;
```

Funzioni
- Insert su albero binario → insert()
- Trovare nodi ad altezza H → visita+altezza
- Sort su string → sort+compare

Trova nodi ad altezza H

```
1  void getStringList(    Node* node,
2                         int curr_h ,
3                         int H,
4                         vector<string> & strList )
5  {
6     if(node==NULL) return;
7
8     if(curr_h==H)
9     {
10        strList.push_back(node->str);
11        return;
12    }
13
14    getStringList(node->left,curr_h+1,H,strList);
15    getStringList(node->right,curr_h+1,H,strList);
16
17    return;
18 }
19
20
```

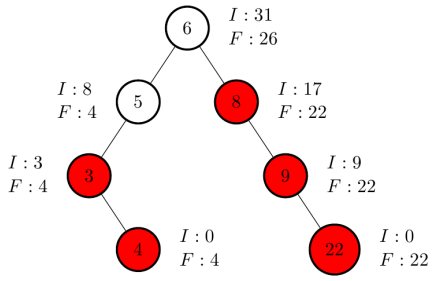

Somma Nodi

- Input:
  - Un intero N
  - N interi
- Operazioni:
  - Inserire gli N interi in un albero binario di ricerca
  - Per ogni nodo *u*, calcolare *I(u)* e *F(u)*
- Output:
  - Stampare le etichette dei nodi tali che *I(u)* <= *F(u)*

Somma Nodi (2)

I(u)*: somma delle chiavi dei nodi interni del sottoalbero radicato in *u

F(u)*: somma delle chiavi delle foglie del sottoalbero radicato in *u



Calcolo I(u) e F(u)

- Devo visitare tutto l'albero.
- I valori di I(u) e F(u) di un nodo padre, dipendono dagli stessi valori calcolati per i nodi figli.
- Di quali nodi posso calcolare I(u) e F(u) "al volo"?
- **Suggerimento:** come facevamo a calcolare l'altezza di un nodo? (relazione padre/figli)

Algoritmi e Strutture Dati

Lezione 5

www.iet.unipi.it/a.virdis

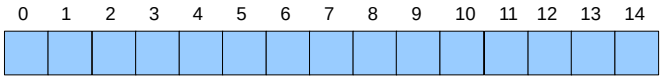
Antonio Virdis

antonio.virdis@unipi.it

Sommario

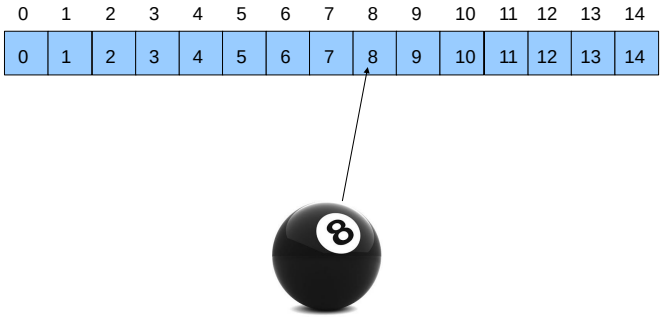
- Hashing
- Hashing e tipi di input
- Esercizi

Array ++

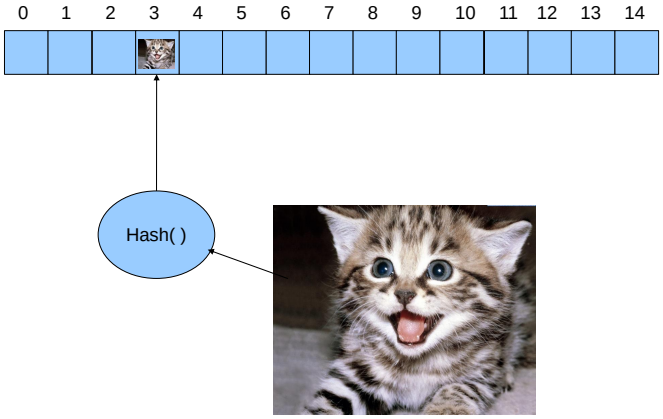


Indirizzamento diretto

Indirizzamento Diretto

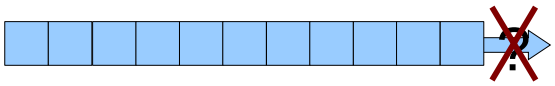


Hashing



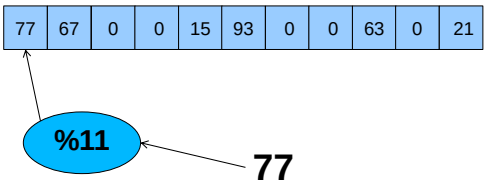
Strutture Dati

- Array
- Vector
- ...



Simple Hash Table

- Trattiamo interi >0
- Chiave coincide con valore
- La funzione HASH e' la funzione modulo
- Convenzione 0 per vuoto



Class

```
1  class HashTable
2  {
3      int * table_;
4      int size_;
5
6  public:
7      HashTable( int size );
8
9      bool insert( int key );
10
11     void print();
12
13     int hash( int key );
14 };
15
16
17
18
```


Costruttore

```
1  HashTable::HashTable( int size )
2  {
3      table_ =  new int[size];
4
5      size_ = size;
6
7
8  }
9
10     memset( address , value , size );
11
12
13
14
15
16
17
18
```


Hashing

```
1  int HashTable::hash( int key )
2  {
3
4      return key % size_;
5
6  }
7
8
9
10
11
12
13
14
15
16
17
18
```

Insert

```
1  bool HashTable::insert( int key )
2  {
3      // trova indice tramite hashing
4
5      // se posizione già occupata
6      {
7          // non posso inserire
8
9      }
10
11
12     // inserisco
13
14 }
15
16
17
18
```

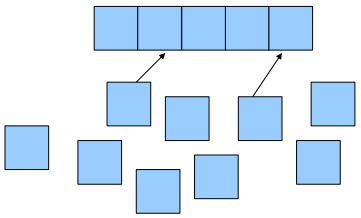

Insert

```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key);
4
5
6
7
8
9
10     table_[index] = key;
11
12     cout << "key stored" << endl;
13     return true;
14 }
15
```

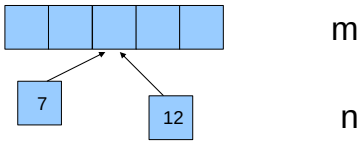

Insert

```
1  bool HashTable::insert( int key )
2  {
3      int index = hash(key);
4
5      if( table_[index] != 0 )
6      {
7          cout << "already occupied" << endl;
8          return false;
9      }
10     table_[index] = key;
11
12     cout << "key stored" << endl;
13     return true;
14 }
15
```


Collisioni

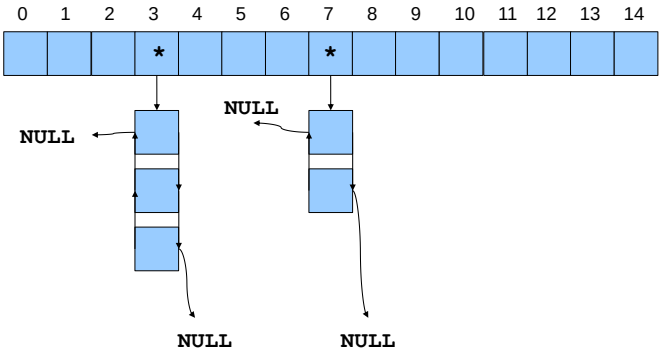


Collisioni



- Liste di trabocco
- Indirizzamento aperto

Array di puntatori

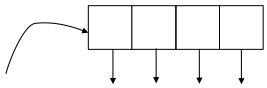


Elem

```
1  struct Elem
2  {
3     int key;
4     Elem * next;
5     Elem * prev;
6
7     Elem(): next(NULL) , prev(NULL) {}
8  };
9
10
11
12
13
14
15
16
17
18
```


Hash con trabocco

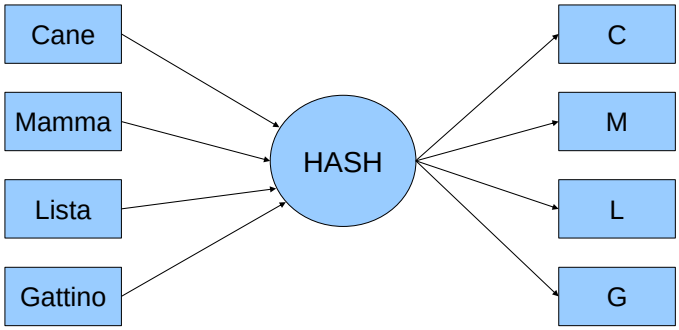
```
1  class HashTable
2  {
3      Elem ** table_;
4      int size_;
5
6  public:
7
8
9
10
11
12
13
14
15
16
17
18 };
```



Implementazione

- Insert / Print / Find
- Stiamo trattando liste
- Facciamo inserimento in testa

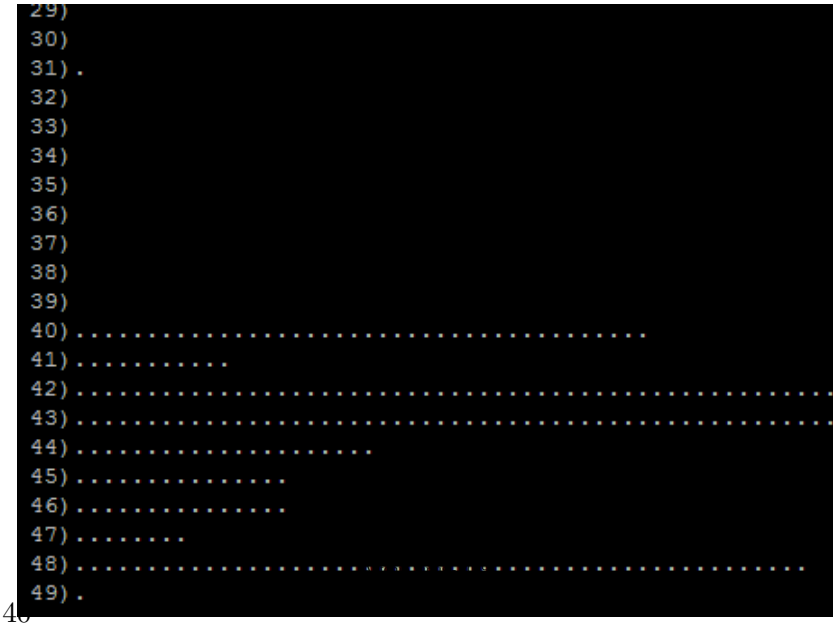
Hashing Stringhe



Prima lettera

```
int hash(string key)
{
    int index = key[0] % size_;
}
```

?



Somma caratteri

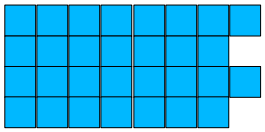
```
for( int i = 0 ; i < key.length() ; ++i )
{
  index = ( index + key[i] )% size_;
}
```

?


```
29).....
30)................
31)............
32)..............................
33)....
34).........................................
35)................
36)...
37)........
38)................
39)................
40)..............
41)..........
42).....................................
43)........
44)...........................
45)..............
46)............................
47)..........
48)............
49).....................
```

Good HASH

- Dipende fortemente dal tipo di applicazione
- Per applicazioni di indexing e' fondamentale l'**uniformità**



- Lavorano sulla **rappresentazione binaria**
- E.g. MurmurHash, CityHash, FarmHash …

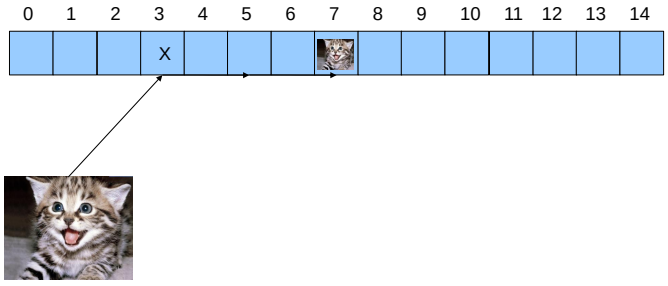
stl::map

```
std::map < key_T , obj_t > table;
```

Tipo chiave — Tipo dati

```
table['uno']="valore uno";
table.find('uno');
```

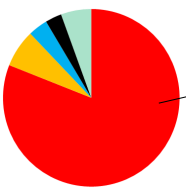

Open Addressing



Quiz: trova la cretinata

- Software di emulazione

valgrind → **pow(x,y)**



Trova la cretinata

```
value =
v1*pow(x,6) + v2*pow(x,5) + v3*pow(x,4)…;
```

$$v_1 \cdot x^6 + v_2 \cdot x^5 + v_3 \cdot x^4 + v_4 \cdot x^3 + v_5 \cdot x^2$$

Esercizio 1

- Sorting e tabelle hash
- Classifica videogame online
  - Salvo coppie <nome , punteggio>
- Interrogazioni
  - Sapere i primi K
  - Sapere posizione di Pippo

Esercizio 2

- Motore di ricerca tematico:
  - Ogni sito ha un
    - Tipo (sport , news , musica …)
    - Nome (gazzetta.it , lercio.it , amicidimaria.it)
    - Dati accessori (#pagine, statistiche…)
    - Numero di accessi
  - Operazioni
    - Accesso ad un sito
    - Dato un tipo, ottenere il nome e i dati del sito con più accessi

Altezza Figli

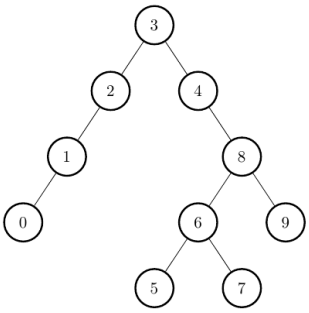
Input:

- N interi da inserire in un albero binario di ricerca

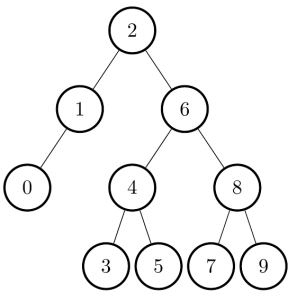
Verificare che :

- Per ciascun nodo, l'altezza dei suoi sottoalberi sinistro e destro deve differire al massimo di uno

Esempi

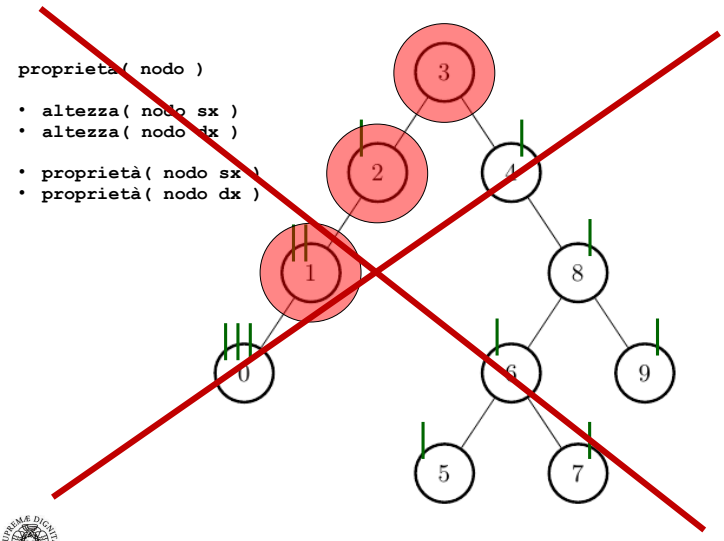


no



si


```
1  bool wrongSol( Node * tree )
2  {
3        int hl,hr;
4
5     // Controllo altezza dei figli sx e dx
6        hl = height(tree->left);
7        hr = height(tree->right);
8
9
10    // se la proprietà e' verificata da:
11    // nodo corrente, nodo sx, nodo dx
12       return true;
13    else
14       return false;
15
16 }
17
18
```




```
1  bool isOk( Node * tree, int & maxH )
2  {
3     // Controllo se ho raggiunto una foglia
4        return true;
5
6     // Controllo i figli sinistro e destro
7     bool propL = isOk(tree->left,hl);
8     bool propR = isOk(tree->right,hr);
9
10    // ottengo l'altezza del nodo corrente
11    // .. il massimo tra quella sx e dx
12
13    // se la proprietà e' verificata da:
14    // nodo corrente, nodo sx, nodo dx
15       return true;
16    else
17       return false;
18 }
```


Esercizi

- Esperimenti
  - Test dimensione table
  - Test tipi di hash
  - Confronto map vs hash
- Esercizi
  - Implementare open addressing
  - Classifica videogame online
  - Motore di ricerca tematico