

Inf Gen II

INFORMATICA GENERALE II
 Ingegneria delle Telecomunicazioni
 Università di Trento

Marco Roveri
roveri@irst.itsc.it

Algoritmi di Ordinamento

AA 2005/2006 MR

Inf Gen II

Ordinamento per selezione

```
void selectionsort( int A[], int N) {
  for (int i = 0; i < N - 1; i++) {
    int min = i;
    for(int j = i + 1; j < N; j++)
      if (A[j] < A[min]) min = j;
    swap(A[i], A[min]);
  }
}
```

Cerco elemento più piccolo nella parte di array ancora da ordinare.

Scambio elemento più piccolo trovato con il primo elemento dell'array ancora da ordinare.

AA 2005/2006 MR

Inf Gen II

Algoritmi di Ordinamento

- Problema: dato un array di elementi (e.g. interi) riordinare gli elementi dell'array in modo che per ogni $1 \leq i \leq N - 1$, $A[i - 1] \leq A[i]$
- Esistono diversi algoritmi con diverse caratteristiche computazionali.
- Nel seguito analizzeremo gli algoritmi più utilizzati e ne discuteremo le caratteristiche.

AA 2005/2006 MR

Inf Gen II

Caratteristiche algoritmo ordinamento per selezione

- L'algoritmo di ordinamento per selezione effettua circa $n^2/2$ confronti ed n scambi in media.
- Il limite asintotico superiore è $O(n^2)$.

AA 2005/2006 MR

Inf Gen II

Ordinamento per selezione

- Cerco elemento più piccolo dell'array e lo scambio con il primo elemento dell'array.
- Cerco il secondo elemento più piccolo e lo scambio con il secondo elemento dell'array.
- Proseguo in questo modo finché l'array non è ordinato.

AA 2005/2006 MR

Inf Gen II

Ordinamento per inserzione

- È il metodo usato dai giocatori di carte per ordinare in mano le carte.
- Considero un elemento per volta e lo inserisco al proprio posto tra quelli già considerati (mantenendo questi ultimi ordinati).
 - l'elemento considerato viene inserito nel posto rimasto vacante in seguito allo spostamento di un posto a destra degli elementi più grandi.

AA 2005/2006 MR

Inf Gen II

Ordinamento per inserzione

AA 2005/2006 MR 7

Inf Gen II

Algoritmo Bubble Sort

- Si basa su scambi di elementi adiacenti se necessari, fino a quando non è più richiesto alcuno scambio e l'array risulta ordinato.

AA 2005/2006 MR 10

Inf Gen II

Algoritmo per inserzione

```
void insertsort( int A[], int N) {
  for(int i = N - 1; i > 0; i--) // porto elemento più piccolo in A[0]
    if (A[i] < A[i-1]) swap(A[i], A[i-1]);
  for(int i = 2; i <= N - 1; i++) {
    int j = i; int v = A[i];
    while( v < A[j-1] ) {
      A[j] = A[j-1]; j--;
    }
    A[j] = v;
  }
}
```

AA 2005/2006 MR 8

Inf Gen II

Algoritmo bubblesort

AA 2005/2006 MR 11

Inf Gen II

Caratteristiche algoritmo ordinamento per inserzione

- L'algoritmo di ordinamento per inserzione effettua circa $n^2/4$ confronti ed $n^2/4$ scambi in media.
- Il limite asintotico superiore è $O(n^2)$.

AA 2005/2006 MR 9

Inf Gen II

Algoritmo bubblesort

AA 2005/2006 MR 12

Inf Gen II

Algoritmo bubblesort

```
void bubblesort(int A[], int N) {
    for( int i = 0; i < N - 1; i++)
        for( int j = N - 1; j > i; j--)
            if (A[j] < A[j-1])
                swap(A[j-1], A[j]);
}
```

AA 2005/2006 MR 13

Inf Gen II

Merge di due array ordinati

```
int * mergeArray(int A[], int N, int B[], M) {
    int * C = new int [M+N];
    for(int i = 0, j = 0, k = 0; k < M + N; k++) {
        if (i == N) {
            C[k] = B[j++]; continue;
        }
        if (j == M) {
            C[k] = A[i++]; continue;
        }
        if (A[i] < B[j])
            C[k] = A[i++];
        else
            C[k] = B[j++];
    }
    return C;
}
```

Allocazione array per il risultato.

A si è esaurito: riempio C con B

B si è esaurito: riempio C con A

Riempio C con elemento minore tra A[i] e B[j], incrementando opportuno indice.

$O(M + N)$

AA 2005/2006 MR 16

Inf Gen II

Caratteristiche algoritmo ordinamento bubblesort

- L'algoritmo di ordinamento bubblesort effettua circa $n^2/2$ confronti ed $n^2/2$ scambi in media.
- Il limite asintotico superiore è $O(n^2)$.

AA 2005/2006 MR 14

Inf Gen II

Merge di due array ordinati

```
void mergeArray(int A[], int N, int B[], M, int C[]) {
    for(int i = 0, j = 0, k = 0; k < M + N; k++) {
        if (i == N) {
            C[k] = B[j++]; continue;
        }
        if (j == M) {
            C[k] = A[i++]; continue;
        }
        if (A[i] < B[j])
            C[k] = A[i++];
        else
            C[k] = B[j++];
    }
}
```

Array per risultato passato come argomento. Deve avere dimensione per contenere tutti elementi di A e B.

$O(M + N)$

AA 2005/2006 MR 17

Inf Gen II

Merge di due array ordinati

- Problema: combinare due array ordinati A[N] e B[M] in un terzo array ordinato C[N+M].
- Usiamo un ciclo for che ad ogni iterazione *i* inserisce un elemento in C[i].
 - Se A si esaurisce prendiamo prossimi elementi da B;
 - Viceversa, se B si esaurisce prendiamo i prossimi elementi da A;
 - Se abbiamo elementi sia in A che in B, il prossimo elemento inserito *i* sarà il minore tra i due elementi A[i] e B[k], e incrementiamo il corrispondente indice.

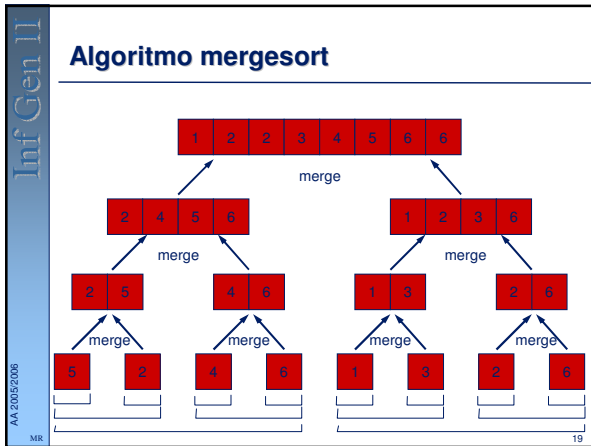
AA 2005/2006 MR 15

Inf Gen II

Ordinamento MergeSort

- L'algoritmo MergeSort è un esempio tipico di programma ricorsivo di tipo *divide et impera*.
 - L'array A[1] ... A[N] è ordinato spezzando l'array in due parti A[1]...A[m] e A[m+1]...A[N], ordinandoli indipendentemente con la stessa tecnica.
 - Merging dei due risultati intermedi.

AA 2005/2006 MR 18



Ordinamento MergeSort: merge

```
void merge(int A[], int l, int m, int r) {
    int i, j;
    int * aux = new int[r+1];
    for(i = m + 1; i > l; i--) aux[i-1] = A[i-1];
    for(j = m; j < r; j++) aux[m+(r-j)] = A[j+1];
    for(int k = l; k <= r; k++) {
        if (aux[j] < aux[i]) {
            A[k] = aux[j--];
        }
        else {
            A[k] = aux[i++];
        }
    }
    delete [] aux;
}
```

Assunzione: $l \leq m \leq r$

Copia degli elementi compresi tra l e m in aux , e elementi tra m e r in altra parte di aux .

Merge dei sotto-array di aux compresi tra l e m e tra m e r in A a partire da l .

Deallocazione di array ausiliario.

AA 2005/2006 MR 22

Ordinamento MergeSort

```
void MergeSort(int A[], int n) {
    MergeSortAux(A, 0, n-1);
}

void MergeSortAux(int A[], int l, int r) {
    if (r <= l) return;
    int m = (r+l)/2;
    MergeSortAux(A, l, m);
    MergeSortAux(A, m+1, r);
    merge(A, l, m, r);
}
```

Ricorsione su sotto-array compreso tra indici l e m .

Ricorsione su sotto-array compreso tra indici $m+1$ e r .

AA 2005/2006 MR 20

- Caratteristiche algoritmo ordinamento mergesort**
- L'algoritmo di ordinamento mergesort effettua circa $n \log(n)$ confronti per ordinare un qualunque array di dimensione n .
 - Lo spazio ausiliario necessario per ridurre il numero di confronti è proporzionale a n .
 - Il limite asintotico superiore è $O(n \log(n))$.
- AA 2005/2006 MR 23

Ordinamento MergeSort

```
void MergeSort(int A[], int n) {
    MergeSortAux(A, 0, n-1);
}

void MergeSortAux (int A[], int l, int r) {
    for(int m = 1; m <= r-l; m = m+m)
        for(int i = l; i <= r-m; i += m + m)
            merge(A, i, i+m-1, min(i+m+m-1, r));
}
```

AA 2005/2006 MR 21

- Algoritmo quicksort**
- È un algoritmo di ordinamento del tipo *divide et impera*.
 - Si basa su un processo di partizionamento dell'array in modo che le seguenti tre condizioni siano verificate:
 - Per qualche valore di i , l'elemento $A[i]$ si trova al posto giusto.
 - Tutti gli elementi $A[1], \dots, A[i-1]$ sono minori od uguali ad $A[i]$.
 - Tutti gli elementi $A[i+1], \dots, A[N]$ sono maggiori od uguali ad $A[i]$.
 - L'array è ordinato partizionando ed applicando ricorsivamente il metodo ai sotto array.
- AA 2005/2006 MR 24

Inf Gen II

Ordinamento quicksort

```

void quicksort(int A[], int N) {
    quicksort_aux(A, 0, N-1);
}

void quicksort_aux(int A[], int l, int r) {
    if (r <= l) return
    int i = partition(A, l, r);
    quicksort_aux(A, l, i-1);
    quicksort_aux(A, i+1, r);
}

```

AA 2005/2006 MR 25

Inf Gen II

Partizionamento dell'array

- Partizioniamo a partire da $A[l] = 5$.
- Gli elementi dell'array precedenti ad $A[j]$ sono minori od uguali a 5.
- Gli elementi dopo $A[j]$ sono maggiori od uguali a 5.

return j

AA 2005/2006 MR 28

Inf Gen II

Caratteristiche algoritmo ordinamento quicksort

- L'algoritmo di ordinamento quicksort effettua circa $n \log(n)$ confronti in media per ordinare un qualunque array di dimensione n .
- Il limite asintotico superiore è $O(n \log(n))$.

AA 2005/2006 MR 26

Inf Gen II

Partizionamento dell'array

```

int partition(int A[], int l, int r) {
    int i = l-1, j = r, v = A[r];
    while (true) { // ciclo infinito
        while (A[++i] < v);
        while (v < A[--j]) if (j == l) break;
        if (i >= j) break;
        swap(A[i], A[j]);
    }
    swap(A[i], A[r]);
    return(i);
}

```

AA 2005/2006 MR 29

Inf Gen II

Partizionamento dell'array

- Scegliamo arbitrariamente un elemento (e.g. $A[r]$), che chiameremo *pivot* (o *elemento di partizionamento*).
- Scandiamo l'array dall'estremità sinistra fino a quando non troviamo un elemento $A[i] \leq A[r]$.
- Scandiamo l'array dall'estremità destra fino a che non troviamo un elemento $A[j] \geq A[r]$.
- Scambiamo $A[i]$ e $A[j]$, e iteriamo.
- Procedendo in questo modo si arriva ad una situazione in cui gli elementi a sinistra di i sono minori di $A[r]$, mentre quelli a destra di j sono maggiori di $A[r]$.

AA 2005/2006 MR 27

Inf Gen II

Analisi delle prestazioni degli algoritmi di sorting

Algoritmo	Limite superiore asintotico
selezione	$O(N^2)$
inserimento	$O(N^2)$
bubblesort	$O(N^2)$
mergesort	$O(N \log(N))$
quicksort	$O(N \log(N))$

AA 2005/2006 MR 30

Algoritmo ShellSort

- La lentezza dell'algoritmo InsertSort risiede nel fatto che le operazioni di scambio avvengono solo tra elementi contigui.
 - Ad esempio se l'elemento più piccolo è in fondo all'array, occorrono N scambi per posizionarlo al posto giusto.
- Per migliorare questo algoritmo è stato pensato l'algoritmo ShellSort.
- L'idea è quella di organizzare l'array in modo che esso soddisfi la proprietà per cui gli elementi aventi tra loro distanza h costituiscono una sequenza ordinata, indipendentemente dall'elemento di partenza.
 - Se si applica l'algoritmo con una sequenza di h che termina con 1, si ottiene un file ordinato.

Esercizi

- Esistono altri algoritmi di ordinamento?
- Trovarli e provare a implementarli, confrontandoli con quelli visti a lezione.

Algoritmo ShellSort

```
void ShellSort(int A[], int l, int r) {
    int h;
    for(h = 1; h <= (r-1)/9; h = 3*h+1);
    for(; h > 0; h /= 3)
        for(int i = l+h; i <= r; i++) {
            int j = i; int v = A[i];
            while((j >= l + h) && (v < A[j-h])) {
                A[j] = A[j-h];
                j = j - h;
            }
            A[j] = v;
        }
}
```

Algoritmo ShellSort

- L'algoritmo ShellSort esegue meno di $O(n^{3/2})$ confronti per come lo abbiamo realizzato.
- Il limite superiore asintotico è $O(n^{3/2})$.
- Usando sequenze particolari di h si possono ottenere prestazioni diverse (e.g. $O(n \log n^2)$).