Corso di Laurea in INFORMATICA

Algoritmi e Strutture Dati

a.a. 2012-2013

MODULO 15

TECNICHE ALGORITMICHE 3

Paradigma generativo: tecnica golosa e tecnica Divide-et-impera

DAL PARADIGMA *GENERATIVO* SCATURISCONO TECNICHE TECNICHE DI PROGETTO DI ALGORITMI CHE GENERANO DIRETTAMENTE LA SOLUZIONE SENZA SELEZIONARLA TRA GLI ELEMENTI DELLO SPAZIO DI RICERCA.

IN QUESTO PARADIGMA LO SPAZIO DI RICERCA È CONSIDERATO ESCLUSIVAMENTE IN FASE DI PROGETTO DELL'ALGORITMO ALLO SCOPO DI CARATTERIZZARE LE SOLUZIONI DEL PROBLEMA E DEFINIRE UNA STRATEGIA RISOLUTIVA DIRETTA PER OGNI ISTANZA.

APPARTENGONO A QUESTO PARADIGMA LA TECNICA GOLOSA E LA DIVIDE- ET-IMPERA.

LA TECNICA GREEDY (GOLOSA, AGGRESSIVA O AVIDA)

SI APPLICA GENERALMENTE A PROBLEMI DI OTTIMIZZAZIONE.

UN ALGORITMO DI TIPO GREEDY E' UN ALGORITMO CHE, DATA UNA ISTANZA DI UN PROBLEMA DI OTTIMIZZAZIONE, CERCA DI OTTENERE UNA SOLUZIONE OTTIMA DA UN PUNTO DI VISTA *GLOBALE* ATTRAVERSO LA SCELTA DELLA SOLUZIONE CHE AD OGNI PASSO APPARE MIGLIORE A LIVELLO *LOCALE*.

QUESTA TECNICA CONSENTE, DOVE APPLICABILE (INFATTI NON SEMPRE SI ARRIVA AD UNA SOLUZIONE OTTIMA), DI TROVARE SOLUZIONI OTTIMALI PER PROBLEMI DI NATURA NON DETERMINISTICA.

LA TECNICA GOLOSA (GREEDY)

Un algoritmo greedy tenta di costruire una soluzione ottima partendo da una soluzione iniziale parziale (abbozzata) ed estendendola a poco a poco fino a quando questo non è più possibile.

Quando l'algoritmo tenta di estendere una soluzione parziale non lo fa considerando tutte le possibili estensioni (che potrebbero essere numerosissime) ma solamente quelle che chiamiamo estensioni locali.

Le estensioni locali di una soluzione parziale rappresentano in qualche modo le più piccole estensioni possibili e sono relativamente poche. Fra tutte le estensioni locali l'algoritmo sceglie la più conveniente, ovvero quella estensione che sembra, almeno localmente, la più promettente per raggiungere una soluzione ottima.

ESEMPIO:

Dare il minor numero di monete di resto utilizzando solo monete da 100, 10, 1 eurocent.

E' un problema risolvibile tramite un algoritmo di tipo greedy: ad ogni passo viene controllato il resto ancora da dare e si aggiunge la moneta con valore maggiore possibile. Quindi per dare un resto di 112 eurocent la macchina farà cadere in sequenza una moneta da 100, poi 10, poi 1 e poi 1 eurocent per un totale di 4 monete).

Si noti che il problema è risolvibile grazie ad un algoritmo greedy solo per quell'insieme di valori di monete: se infatti, per assurdo, avessimo anche monete da 105 eurocent, l'algoritmo greedy darebbe un totale di 8 monete (una da 105 e 7 da 1), quando posso trovare una soluzione ottima con 4 monete (100+10+1+1).

LA TECNICA GOLOSA (GREEDY)

RICHIEDE CHE L'ALGORITMO ESEGUA IL PROCESSO DI COSTRUZIONE DI UN ELEMENTO DELLO SPAZIO DI RICERCA IN *STADI* E SI BASA SUI SEGUENTI PRINCIPI:

- **AD OGNI *STADIO i*, PER LA COMPONENTE *i-ESIMA* VIENE SCELTO IL VALORE CHE, TRA QUELLI AMMISSIBILI, RISULTA IL MIGLIORE RISPETTO DA UN DETERMINATO CRITERIO**
- **UNA VOLTA FATTA LA SCELTA PER LA i-ESIMA COMPONENTE, SI PASSA A CONSIDERARE LE ALTRE, *SENZA PIÙ TORNARE SULLA DECISIONE PRESA* .**

LO SCHEMA DI UN ALGORITMO GOLOSO (GREEDY)

PRESUPPONE CHE L'ALGORITMO ACQUISISCA LA RAPPRESENTAZIONE DI UNA ISTANZA DEL PROBLEMA E DISPONGA DI UN METODO PER ORGANIZZARE IN STADI LA COSTRUZIONE DI UN ELEMENTO DELLO SPAZIO DI RICERCA

1. **PONI i A 1 E INIZIALIZZA z**
2. **DETERMINA L'INSIEME A DEI VALORI AMMISSIBILI PER LA COMPONENTE i-ESIMA DI z E SE A ≠ ∅ SCEGLI IL MIGLIORE IN A, RISPETTO AL CRITERIO DI PREFERENZA FISSATO**
3. **SE L'I-ESIMO STADIO È L'ULTIMO ALLORA TERMINA E RESTITUISCE O(z) COME RISULTATO**
4. **ALTRIMENTI INCREMENTA i DI 1 E TORNA AL PASSO 2**

ESEMPIO:

IL PROBLEMA DELLO ZAINO (KNAPSACK)

COME È NOTO, SI DISPONE DI UN BUDGET B SE SI DEVE CERCARE DI MASSIMIZZARE LA RENDITA SCEGLIENDO TRA n POSSIBILI INVESTIMENTI, CIASCUNO DEI QUALI CARATTERIZZATO DA UN PROFITTO P_i E DA UN COSTO C_i. IL PROBLEMA VIENE FORMULATO COSÌ:

DATI 2n+1 INTERI POSITIVI

$$P_1 , P_2 , \dots , P_n , C_1 , C_2 , \dots , C_n , B$$

SI VOGLIONO TROVARE n VALORI

$$X_1 , X_2 , \dots , X_n$$ **TALI CHE**

- $X_n \in \{ 0 , 1 \} \quad (1 \leq i \leq n)$
- $\sum_{i=1}^{n} P_i X_i$ **SIA MASSIMA**
- $\sum_{i=1}^{n} C_i X_i \leq B$

RICORDIAMO CHE NELLA FORMALIZZAZIONE GENERALE I VALORI:

$$P_1,\ldots,P_n$$

SONO DETTI *PROFITTI*. I VALORI:

$$C_1,\ldots,C_n$$

SONO CHIAMATI *COSTI*.

LA SPECIFICA

- **LO SPAZIO DI INPUT I È L'INSIEME DELLE COPPIE (n,Q) DOVE n È UN INTERO POSITIVO E Q È UNA SEQUENZA DI 2n+1 INTERI POSITIVI. L'ELEMENTO DI QUESTO SPAZIO È:**

$$(n,(P_1,\ldots,P_n,C_1,\ldots,C_n,B));$$

- **LO SPAZIO DELLE SOLUZIONI S È L'INSIEME DELLE SEQUENZE FINITE:**

$$<X_1,\ldots,X_n>$$

CON $n \in N^+$ E $X_i \in \{0,1\}$;

- **LA RELAZIONE CARATTERISTICA È DATA DALL'INSIEME DELLE COPPIE:**

$$(n,(P_1,\ldots,P_n,C_1,\ldots,C_n,B)),(X_1,\ldots,X_m)$$

TALI CHE:

$$\sum_{i=1}^{n} C_i X_i \leq B ;$$

- **IL QUESITO È $q_{ott}(N,m,\geq)$ DOVE:**

$$m = \sum_{i=1}^{n} P_i X_i .$$

CONSIDERIAMO L'ISTANZA B=5 E TRE ELEMENTI DISPONIBILI.

i	P_i	C_i	P_i / C_i
1	6	2	3
2	4	1	4
3	7	3	3.5

IL CRITERIO DI BASE PER ORDINARE LE VARIABILI È QUELLO DI VALUTARE SIA IL COSTO CHE IL PROFITTO P_i / C_i .

UNA VOLTA EFFETTUATO L'ORDINAMENTO, SI POSSONO SCEGLIERE LE VARIABILI PONENDOLE AL MASSIMO VALORE COMPATIBILE CON I VINCOLI.

IL METODO/1

- **DEFINIRE UNA FUNZIONE *ZAINOGREEDY* CHE HA COME PARAMETRI:**
  - ***IL NUMERO n DI OGGETTI DENOTATI DAI NUMERI INTERI DA ∅ A n – 1***
  - ***IL VETTORE P DEI PROFITTI***
  - ***IL VETTORE C DEI COSTI***
  - ***IL BUDGET B***
- **IL RISULTATO VIENE RESTITUITO NEL VETTORE X NEL QUALE LA i-ESIMA COMPONENTE VALE 1 SE L'OGGETTO i È INCLUSO NELLA SOLUZIONE, E 0 ALTRIMENTI;**

IL METODO/2

- LA FUNZIONE ORDINA IN UN VETTORE AUSILIARIO V GLI OGGETTI SECONDO L'ORDINE NON CRESCENTE DEL RAPPORTO P_i/C_i. OGNI ELEMENTO È IN CORRISPONDENZA DI UN OGGETTO E CONTIENE L'INDICE DELL'OGGETTO IN UN APPOSITO CAMPO, OLTRE AL RAPPORTO P_i/C_i AD ESSO ASSOCIATO;
- DOPO L'ORDINAMENTO, LA FUNZIONE SCANDISCE IL VETTORE V, ANALIZZANDO $\forall$ i L'OGGETTO IN V(i) E DECIDENDO, IN BASE ALLA COMPATIBILITÀ CON IL VINCOLO SUL BUDGET, SE INCLUDERLO O MENO NELLA SOLUZIONE.

TECNICA GREEDY: CONCETTI DI BASE

UN ALGORITMO GREEDY PERMETTE DI OTTENERE UNA SOLUZIONE OTTIMA MEDIANTE UNA SEQUENZA DI DECISIONI; IN OGNI PASSO VIENE PRESA LA DECISIONE CHE AL MOMENTO APPARE MIGLIORE.

QUESTA STRATEGIA EURISTICA NON GARANTISCE SEMPRE UNA SOLUZIONE OTTIMA

COME SI PUÒ DETERMINARE SE UN ALGORITMO GREEDY È IN GRADO DI TROVARE LA SOLUZIONE DI UN PROBLEMA DI OTTIMIZZAZIONE ?

I PROBLEMI CHE SI PRESTANO AD ESSERE RISOLTI CON UNA STRATEGIA GREEDY PRESENTANO ALCUNE CARATTERISTICHE :

- ***LA PROPRIETÀ DELLA SCELTA GREEDY***
- ***LA SOTTOSTRUTTURA OTTIMA***

LA PROPRIETÀ DELLA SCELTA GREEDY ASSICURA CHE SI PUÒ OTTENERE UNA SOLUZIONE OTTIMA GLOBALE PRENDENDO DECISIONI CHE SONO *OTTIMI LOCALI*.

UN PROBLEMA PRESENTA UNA *SOTTOSTRUTTURA OTTIMA* SE UNA SOLUZIONE OTTIMA DEL PROBLEMA CONTIENE AL SUO INTERNO UNA SOLUZIONE OTTIMA DEI SOTTOPROBLEMI.

LA TECNICA GREEDY È SPESSO UTILE PER PROBLEMI DI *"SCHEDULING"*, IN CUI SI HANNO PROGRAMMI DA ESEGUIRE SU UN "PROCESSORE" E SI VUOLE L'ORDINE DI ESECUZIONE "OTTIMO" IN BASE A UN CERTO CRITERIO.

ESEMPIO:

PROBLEMA DI SELEZIONE DI ATTIVITÀ

IL PROBLEMA È QUELLO DELL'ASSEGNAMENTO DI UNA RISORSA CONDIVISA DA UN CERTO NUMERO DI ATTIVITÀ IN COMPETIZIONE FRA LORO.

SIA $S=\{1,2,\ldots,n\}$ UN INSIEME DI n ATTIVITÀ CHE DEVONO UTILIZZARE UNA DETERMINATA RISORSA CHE NON PUÒ ESSERE UTILIZZATA CONTEMPORANEAMENTE.

UNA GENERICA ATTIVITÀ k È CARATTERIZZATA DA UN TEMPO DI INIZIO (ATTIVAZIONE) I_k E UN TEMPO DI FINE (CONCLUSIONE) F_k CON $I_k \leq F_k$.

DUE ATTIVITÀ k E j SONO DETTE *COMPATIBILI* SE GLI INTERVALLI $[I_k, F_k]$ E $[I_j, F_j]$ *NON SI SOVRAPPONGONO*.

IL PROBLEMA CHIEDE DI INDIVIDUARE UN INSIEME CHE CONTIENE IL MASSIMO NUMERO DI ATTIVITÀ MUTUAMENTE COMPATIBILI.

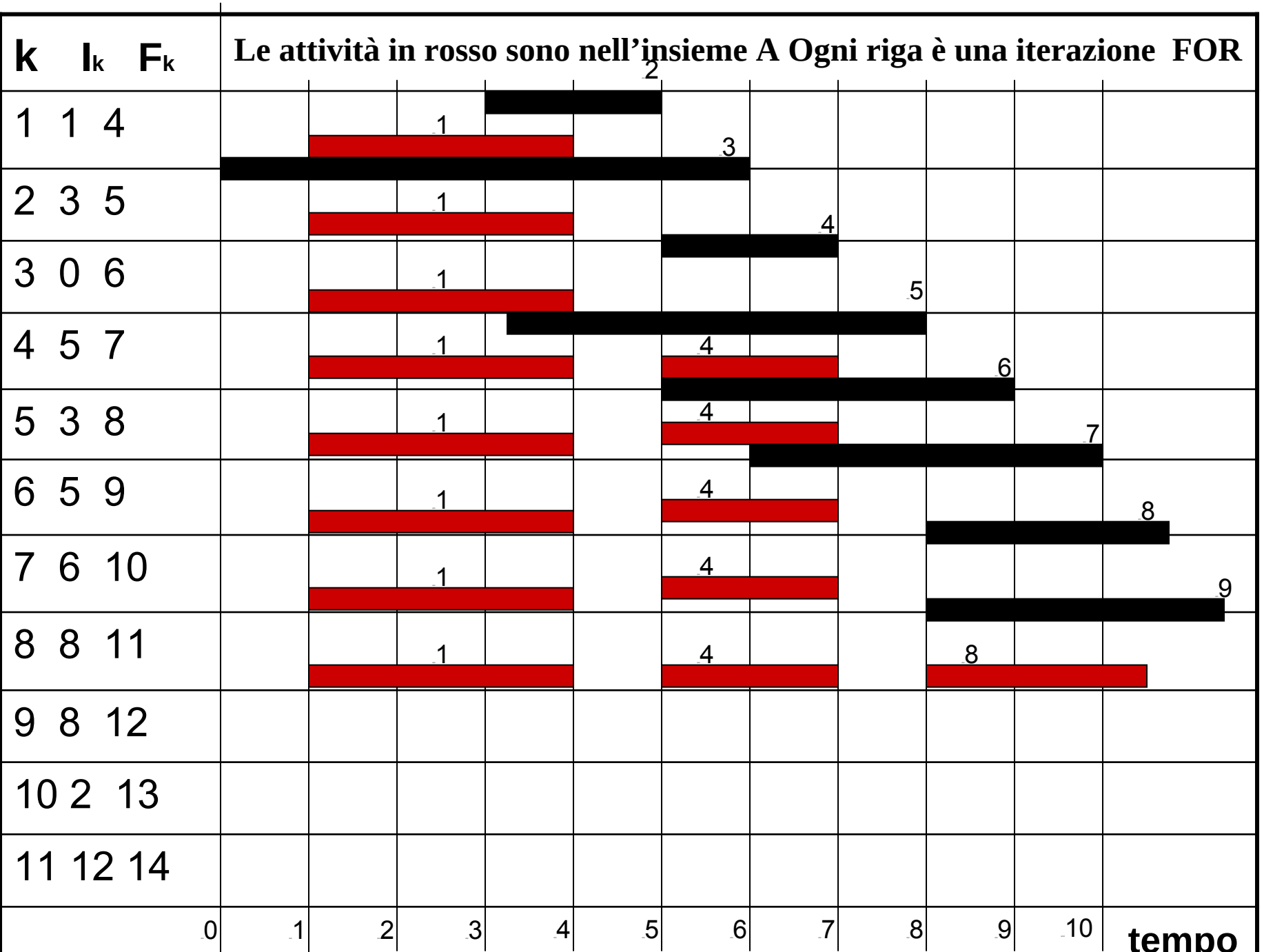
SI ASSUME CHE LE ATTIVITÀ IN INGRESSO SIANO ORDINATE IN MODO CRESCENTE RISPETTO AL LORO TEMPO DI FINE $F_1 \leq F_2 \leq F_3 \leq \ldots \leq F_n$

LO SPAZIO DI RICERCA PUÒ ESSERE COSÌ DEFINITO. SE 1,…,n SONO LE ATTIVITÀ RELATIVE A UNA ISTANZA i, ALLORA LO SPAZIO DI RICERCA È L'INSIEME DI TUTTI POSSIBILI SOTTOINSIEMI DI {1,…,n}.

AD OGNI STADIO L'ALGORITMO GOLOSO SCEGLIE L'ATTIVITÀ k CHE, TRA QUELLE ANCORA DISPONIBILI, RILASCIA PER PRIMA LA RISORSA CONDIVISA (MINOR TEMPO F_k).

UNA VOLTA SCELTA L'ATTIVITÀ k, SI AGGIORNA L'INSIEME DI ATTIVITÀ DISPONIBILI, ELIMINANDO DALL'INSIEME QUELLE INCOMPATIBILI CON k, CIOÈ QUELLE CHE RICHIEDONO L'USO DELLA RISORSA NEL TEMPO CHE INTERCORRE TRA I_k E F_k

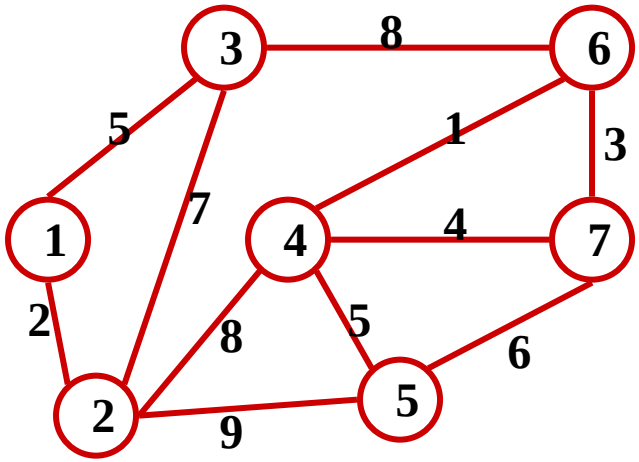
```
PROCEDURE      GREEDY (I,F : VETTORE ; n:INTEGER)
VAR A:INSIEME                    {CONTIENE LE ATTIVITÀ}
BEGIN
        CREAINSIEME (A)
        INSERISCI (1,A)
        J←1
        FOR k=2 TO n DO BEGIN
                IF Ik ≤ Fk THEN BEGIN
                        INSERISCI (k,A)
                        j ←k
                        END
                END
END
```


k Ik Fk
Le attività in rosso sono nell'insieme A Ogni riga è una iterazione FOR
1 1 4
2 3 5
3 0 6
4 5 7
5 3 8
6 5 9
7 6 10
8 8 11
9 8 12
10 2 13
11 12 14
0 1 2 3 4 5 6 7 8 9 10
tempo

UN PROBLEMA:

PROBLEMA DEL MINIMO ALBERO DI COPERTURA

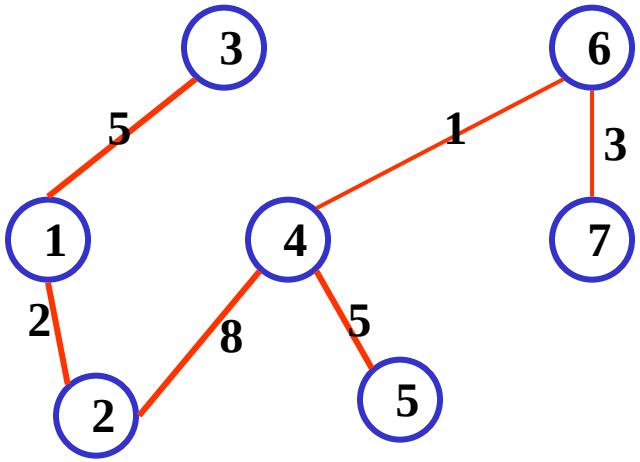
DATO UN GRAFO NON ORIENTATO E CONNESSO G=(N,A), CON PESI SUGLI ARCHI (NON NEGATIVI), TROVARE UN ALBERO DI COPERTURA PER G, CIOÈ UN ALBERO AVENTE TUTTI I NODI IN N, MA SOLO ALCUNI ARCHI IN A, IN MODO TALE CHE SIA MINIMA LA SOMMA DEI PESI ASSOCIATI AGLI ARCHI.



IL PROBLEMA PUÒ ESSERE RISOLTO CON MOLTI ALGORITMI, DEI QUALI I PIU' NOTI SI DEVONO A KRUSKAL (1956) E A PRIM (1957).

L'ALGORITMO DI KRUSKAL USA LA *TECNICA "GREEDY"*.

L'ALBERO DI COPERTURA MINIMO PER IL GRAFO PRECEDENTE E'



L'ALGORITMO DI KRUSKAL, DOPO AVER ORDINATO GLI ARCHI SECONDO I PESI CRESCENTI, LI ESAMINA IN TALE ORDINE, INSERENDOLI NELLA SOLUZIONE *SE NON FORMANO CICLI CON ALTRI ARCHI GIÀ SCELTI*.

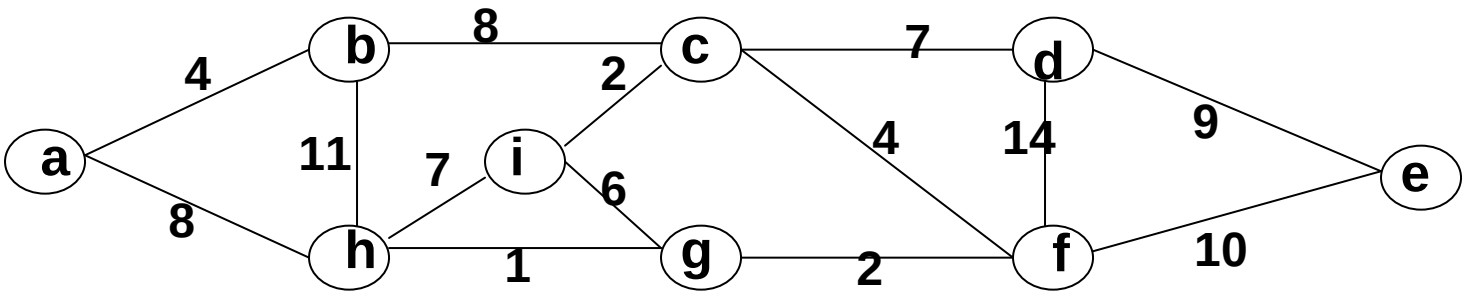
AD UN LIVELLO MOLTO GENERALE, L'ALGORITMO E' ESPRIMIBILE IN QUESTI TERMINI:

```
PROCEDURE      KRUSKAL(GRAFO)
 BEGIN
   T ← ∅
   {ORDINA GLI ARCHI DI G PER PESO CRESCENTE}
   FOR   a=1 TO m DO
        IF (L'ARCO a=(i,j) NON FORMA CICLO CON ALTRI
                    ARCHI DI T )
        THEN   T ← T ∪ (a)
END
```

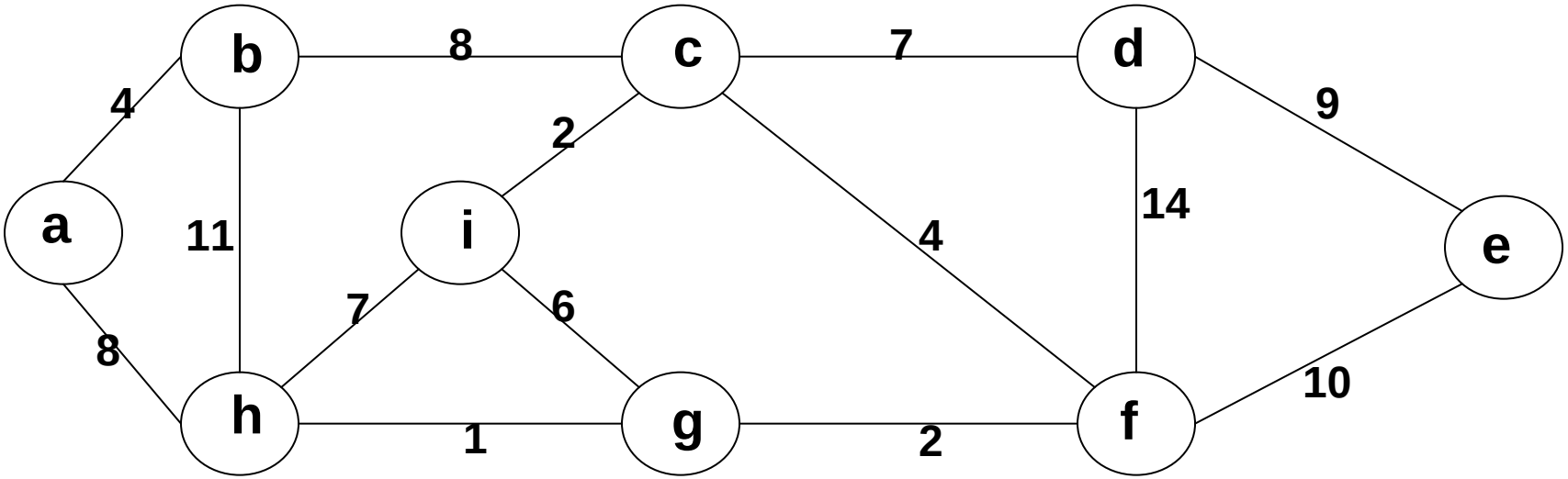
POSSIAMO RAPPRESENTARE IL GRAFO G COME PREFERIAMO. NELLA REALIZZAZIONE DATA DA KRUSKAL IL GRAFO E' REALIZZATO CON UN VETTORE DI ARCHI E L'ALBERO T CON UNA LISTA DI ARCHI (REALIZZATA CON PUNTATORI).

PER ESEMPIO, SI POTREBBE DEFINIRE UN TIPO

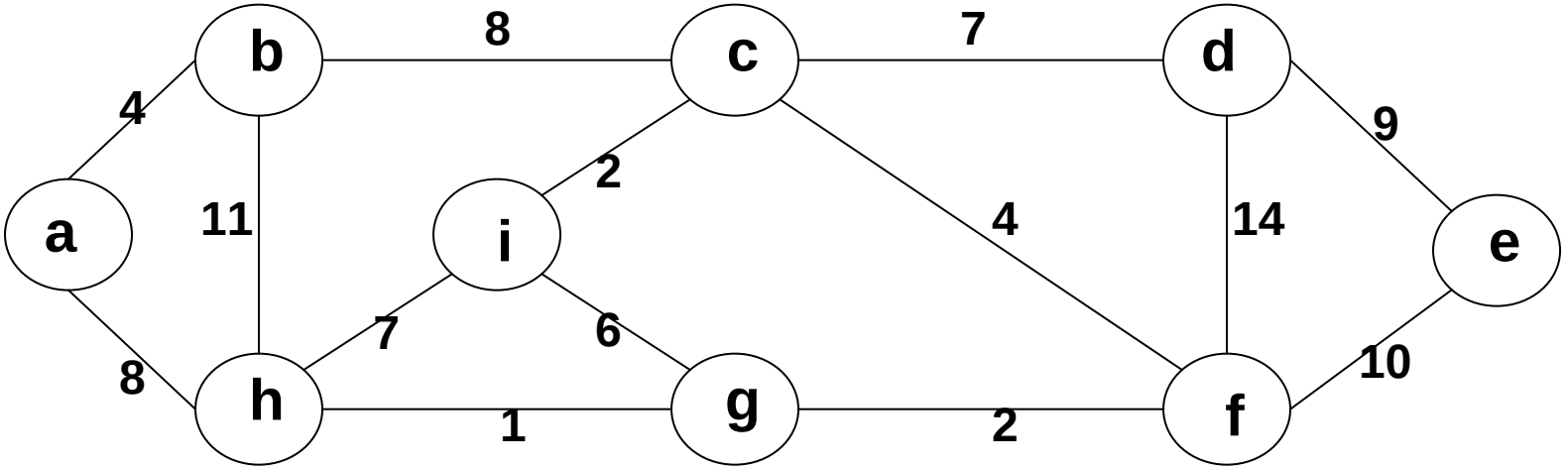
```
TYPE
   ARCO = RECORD
              i,j:INTEGER;
              PESO:INTEGER;
              END;
    GRAFO = RECORD
                A = ARRAY (1..MAXLUNG) OF ARCO;
                n,m:INTEGER;
                END;
```

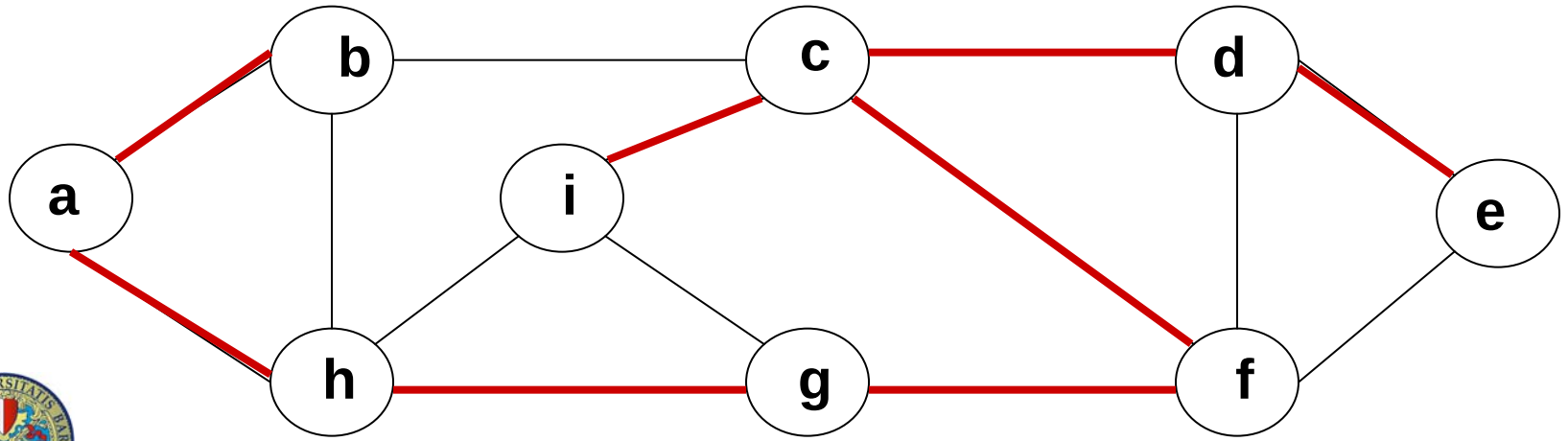


ECCO L'ORDINE IN CUI L'ALGORITMO CONSIDERA GLI ARCHI.



UNA VOLTA CHE SI CONSIDERA L'ARCO SE QUESTO UNISCE DUE ALBERI DISTINTI NELLA FORESTA, L'ARCO VIENE AGGIUNTO ALLA FORESTA E I DUE ALBERI FUSI IN UNO.





NELL'ALGORITMO DI KRUSKAL LA COSTRUZIONE DI T AVVIENE PER UNIONE DI COMPONENTI CONNESSE RAPPRESENTABILI COME INSIEMI DISGIUNTI (DUE COMPONENTI CONNESSE SI FONDONO IN UNA CON L'AGGIUNTA DI UN NUOVO ARCO)

COME E' NOTO, LA STRUTTURA MFSET (MERGE-FIND SET) È UNA PARTIZIONE DI UN INSIEME IN SOTTOINSIEMI DISGIUNTI DETTI COMPONENTI.

DUNQUE E' POSSIBILE UTILIZZARE UNA STRUTTURA DATI DI TIPO MFSET PER COSTRUIRE T.

DIAMO DI SEGUITO UNA DESCRIZIONE IN PSEUDOLINGUAGGIO LASCIANDO COME ESERCITAZIONE LA REALIZZAZIONE DEL CODICE IN UN AMBIENTE OO.

```
PROCEDURE    KRUSKAL    (G di tipo GRAFO)
        h di tipo INTEGER, T di tipo LISTA,
        S di tipo MFSET; {PARTIZIONE DI UN INSIEME IN
                SOTTOINSIEMI DISGIUNTI (COMPONENTI)}
            CREALISTA(T);
            (ORDINA G.A(1), …, G.A(G.m) PER ORDINE
            CRESCENTE DI G.A(h).PESO );
            CREAMFSET (G.n,S);
            FOR h:=1 TO G.m DO
                    IF NOT TROVA (A(h).i,A(h).j,S)
                            THEN  FONDI (A(h).i,A(h).j,S)
                            INSLISTA (PRIMOLISTA(T),A(h),T)

        END
```

DIVIDE-ET-IMPERA

Il *divide et impera* rappresenta un approccio molto potente per la risoluzione di vari problemi computazionali.

Gli algoritmi *divide et impera* dividono ricorsivamente un problema in due o più sotto-problemi sino a che questi ultimi diventino di semplice risoluzione, quindi, si combinano le soluzioni al fine di ottenere la soluzione del problema dato. Il procedimento è caratterizzato dai seguenti passi:

- **Divide: suddivisione del problema in sottoprolemi**
- **Impera: soluzione ricorsiva dei sottoproblemi. Problemi piccoli sono risolti direttamente.**
- **Combina: le soluzioni dei sottoproblemi sono ricoombinate per ottenere la soluzione del problema originale**

LA TECNICA DIVIDE-ET-IMPERA

DERIVA DALL'IDEA DI DETERMINARE LA STRATEGIA DI UN PROBLEMA FACENDO RICORSO AL *PRINCIPIO DI DECOMPOSIZIONE INDUTTIVA*. È NECESSARIO DISPORRE DI:

- **UNA RELAZIONE DI ORDINAMENTO SULLE ISTANZE DEL PROBLEMA, BASATA SULLA DIMENSIONE DELL'INPUT;**
- **UN METODO DI RISOLUZIONE DIRETTO PER TUTTE LE ISTANZE DEL PROBLEMA CHE NON SUPERANO UNA PREFISSATA DIMENSIONE LIMITE;**
- **UN MECCANISMO PER SUDDIVIDERE I DATI DI INGRESSO RELATIVI AD UNA ISTANZA IN DIVERSE PARTI, CIASCUNA DI DIMENSIONE MINORE DI QUELLA ORIGINARIA E RAPPRESENTANTE L'INPUT DI UNA NUOVA ISTANZA DELLO STESSO PROBLEMA;**
- **UN MECCANISMO PER COMPORRE LE SOLUZIONI PER LE ISTANZE OTTENUTE DALLA SUDDIVISIONE, PER OTTENERE LA SOLUZIONE PER L'ISTANZA ORIGINARIA.**

LA TECNICA DIVIDE-ET-IMPERA

È UNA TECNICA GENERATIVA CHE FA USO DELLA DECOMPOSIZIONE INDUTTIVA PER DETERMINARE IL METODO SOLUTIVO.

L'EFFICACIA DELLA TECNICA SI MANIFESTA ATTRAVERSO DUE ASPETTI:

- **CONSENTE DI PROGETTARE ALGORITMI SEMPLICI E INTUITIVI ATTRAVERSO L'INDUZIONE.**
- **SPESSO QUESTI ALGORITMI HANNO PRESTAZIONI MIGLIORI RISPETTO AD ALTRI (TIPICAMENTE A QUELLI DEL PARADIGMA SELETTIVO).**

L'EFFICIENZA DEGLI ALGORITMI DIPENDE DAL NUMERO a DI SOTTOPROBLEMI GENERATI DALLA DIMENSIONE b DEI DATI IN INGRESSO AI SOTTOPROBLEMI E DAL COSTO f(n) NECESSARIO PER LA SCOMPOSIZIONE DEL PROBLEMA E LA COMPOSIZIONE DEI RISULTATI.

ALGORITMO DIVIDE-ET-IMPERA

1. **SE L'INPUT HA DIMENSIONE INFERIORE A UN CERTO VALORE k ALLORA UTILIZZA UN METODO DIRETTO PER OTTENERE IL RISULTATO**
2. **ALTRIMENTI, DIVIDI L'INPUT IN PARTI, CIASCUNA DI DIMENSIONE INFERIORE ALL'INPUT ORIGINARIO (*DIVIDE*)**
3. **ESEGUI RICORSIVAMENTE L'ALGORITMO SU CIASCUNO DEGLI INPUT INDIVIDUATI AL PASSO PRECEDENTE**
4. **COMPONI I RISULTATI OTTENUTI AL PASSO PRECEDENTE OTTENENDO IL RISULTATO PER L'ISTANZA ORIGINARIA (*IMPERA*)**

ESEMPIO:

IL PROBLEMA DEL MINIMO E MASSIMO SIMULTANEI

IN ALCUNE APPLICAZIONI SERVE TROVARE IL MINIMO E IL MASSIMO IN UN INSIEME DI n ELEMENTI SIMULTANEAMENTE.

PER ESEMPIO, UN PROGRAMMA GRAFICO PUÒ AVER BISOGNO DI RAPPRESENTARE IN SCALA UN INSIEME DI DATI (x,y): IN QUESTO CASO VA DETERMINATO IL MINIMO E IL MASSIMO DI OGNI COORDINATA.

CERCANDO IL MINIMO E IL MASSIMO IN MODO INDIPENDENTE CI VORRÀ UN TOTALE DI 2(n–1) CONFRONTI.

MANTENENDO GLI ELEMENTI MINIMO E MASSIMO VIA VIA INCONTRATI E CONFRONTANDO I DUE ELEMENTI DELLA COPPIA IN INPUT SONO SUFFICIENTI 3(n/2) CONFRONTI.

L'ALGORITMO DEL MASSIMO E MINIMO SIMULTANEI:

1. SE LA DIMENSIONE DEL VETTORE NON SUPERA 2, ALLORA CALCOLA DIRETTAMENTE, MEDIANTE UN UNICO CONFRONTO, IL MINIMO E IL MASSIMO
2. ALTRIMENTI, DIVIDI IL VETTORE IN DUE SOTTOVETTORI DELLA STESSA DIMENSIONE, CALCOLA RICORSIVAMENTE IL MINIMO MIN1 E IL MASSIMO MAX1 DEL PRIMO SOTTOVETTORE E IL MINIMO MIN2 E MAX2 DEL SECONDO SOTTOVETTORE
3. DETERMINA IL MINIMO E IL MASSIMO DEL VETTORE COMPLESSIVO CONFRONTANDO MIN1 CON MIN2 E MAX1 CON MAX2

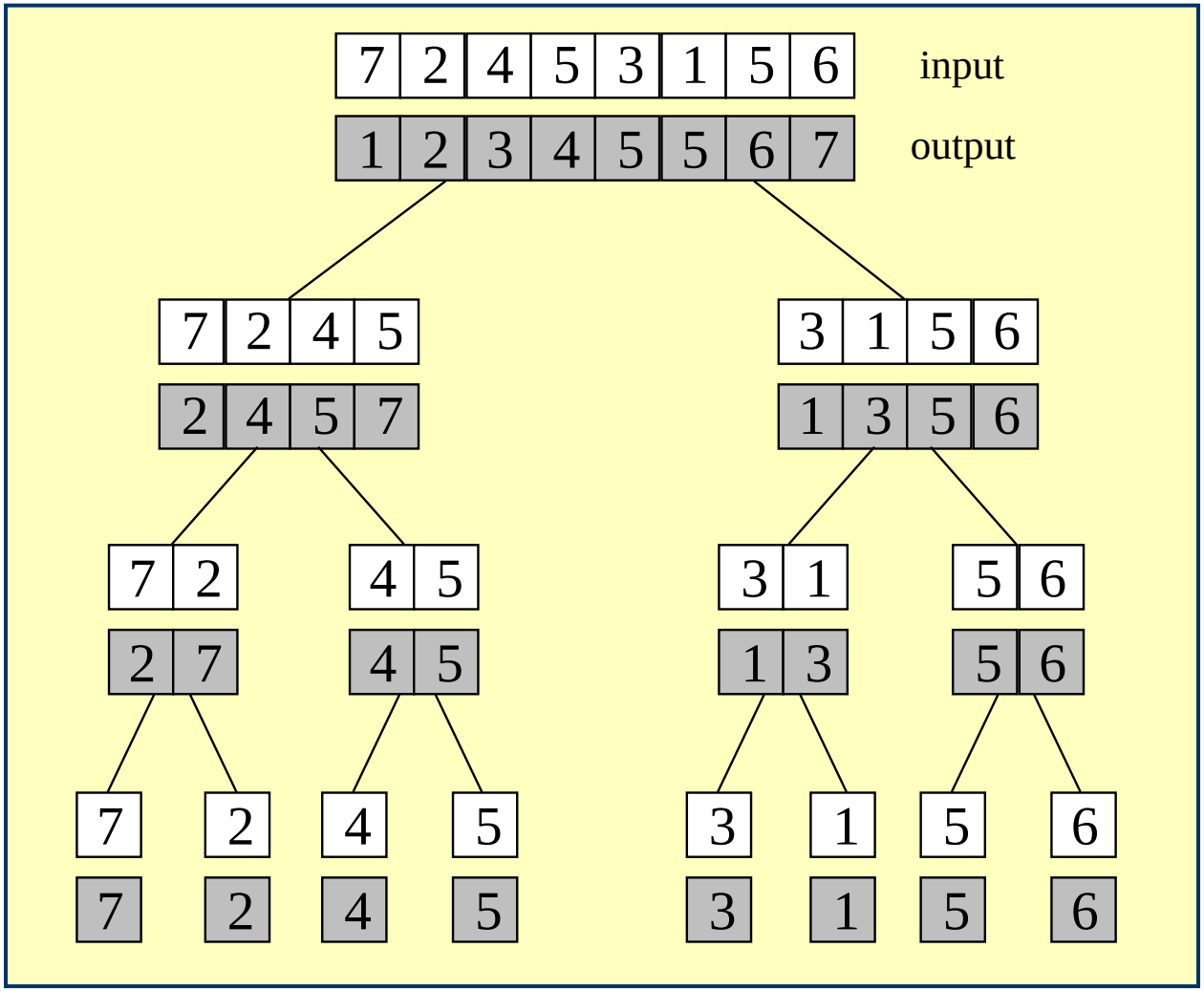
LA APPLICAZIONE PIU' NOTA DI QUESTA TECNICA SI HA NEGLI ALGORITMI DI ORDINAMENTO

NaturalMergeSort (su array)

Usa la tecnica del divide et impera:

1. Divide: dividi l'array a metà
2. Risolvi i due sottoproblemi ricorsivamente
3. Impera: fondi le due sottosequenze ordinate

Esempio di esecuzione



Input ed output delle chiamate ricorsive

Procedura Merge

- Due array ordinati A e B possono essere fusi rapidamente:
  - estrai ripetutamente il minimo di A e B e copialo nell'array di output, finché A oppure B non diventa vuoto
  - copia gli elementi dell'array non vuoto alla fine dell'array di output

Nota: dato un array A e due indici $x \leq y$, denotiamo con A[x;y] la porzione di A costituita da A[x], A[x+1],…,A[y]

Merge (A, i_1, f_1, f_2)

1. Sia X un array ausiliario di lunghezza f_2-i_1+1
2. i=1
3. i_2=f_1+1
4. **while** ($i_1 \leq f_1$ e $i_2 \leq f_2$) **do**
5. **if** ($A[i_1] \leq A[i_2]$)
6. **then** $X[i]=A[i_1]$
7. incrementa i e i_1
8. **else** $X[i]=A[i_2]$
9. incrementa i e i_2
10. **if** ($i_1<f_1$) **then** copia $A[i_1;f_1]$ alla fine di X
11. **else** copia $A[i_2;f_2]$ alla fine di X
12. copia X in $A[i_1;f_2]$

fonde $A[i_1;f_1]$ e $A[f_1+1;f_2]$
output in $A[i_1;f_2]$

Osservazione: sto usando un array ausiliario

QuickSort

Usa la tecnica del divide et impera:

1. Divide: scegli un elemento x della sequenza (perno o pivot) e partiziona la sequenza in elementi $\leq x$ ed elementi $>x$
2. Risolvi i due sottoproblemi ricorsivamente
3. Impera: restituisci la concatenazione delle due sottosequenze ordinate

Rispetto al MergeSort, divide complesso ed impera semplice

QuickSort (A)

1. scegli elemento x in A
2. partiziona A rispetto a x calcolando:
3. $A_1=\{y \in A : y \leq x\}$
4. $A_2=\{y \in A : y > x\}$
5. **if** ($|A_1| > 1$) **then** QuickSort(A_1)
6. **if** ($|A_2| > 1$) **then** QuickSort(A_2)
7. copia la concatenazione di A_1 e A_2 in A

Esempio di esecuzione