• Illustrare l’algoritmo di ordinamento Bubble Sort e discuterne il costo computazionale.

• Mostrare i passi eseguiti dall’algoritmo per ordinare l’array di input: 3 6 12 9 4 1.

• Illustrare mediante pseudo-codice il procedimento di fusione usato nell’algoritmo merge sort e indicare il costo computazionale della sola operazione di fusione.

• Mostrare con un esempio numerico (usando valori a vostra scelta) i passi dell’operazione di fusione del merge sort.

• Descrivere le strutture (possibilmente in codice C) per la rappresentazione collegata di alberi n-ari.

• Illustrare tramite pseudo-codice una funzione che verifica se un nodo di un albero n-ario `e una foglia, secondo la definizione data al punto precedente.

• Calcolare il costo della funzione ricorsiva countPosSCL() definita nel file scl.c

// conta i valori positivi

// Schema Contatore condizionato

int countPosSCL ( TipoSCL scl ) {

if (emptySCL(scl)){ // 1

return 0; // 1

}

if (scl -> info > 0){ // 1

return 1 + countPosSCL(scl -> next); // 1

}

return countPosSCL(scl -> next); // 1

}

Soluzione

costo della funzione ricorsiva (per input di dimensione n) = numero di chiamate ricorsive (per input di dim n)\*costo di ogni chiamata.

costo (su input dim n) = n \* (1+1+1) = 3n = O(n)

costo (su input dim n) = n \* 5 = 5n = O(n)

• Selezionare la funzione che si ritiene migliore dal punto di vista del costo computazionale tra le funzioni C f1() ed f2() definite nel file confronto-costi.c

// T(n) = O(log n)

int f1(int\* a, int n){

int c = 1;

int r = 0;

while (c < n && a[c] != 0){

r += a[c];//operazione dominante

c += c; // 1,2,4,8,16,32,...

}

return r;

}

// T(n) = O(n)

int f2(int\* a, int n){

int c = 1;

int r = 0;

while (c < n && a[c] != 0){

r += a[c];

for (int j=c; j>0; j--){

c++; // operazione dominante: n volte

}

}

return r;

}

• Illustrare (pseudo-codice) l'algoritmo di ordinamento per selezione (selection sort) e mostrarne l'esecuzione sul seguente array di input: 2 6 1 34 10 9 8 4