Algoritmi di ordinamento in linguaggio C

Ordinamento per inserimento

Insertion Sort con funzione ausiliaria

```
#include <stdio.h>
#define MAX_DIM 5

int inserisci_valore(int vett[], int dim, int valore) {
    int i = dim;

    while (i > 0 && vett[i-1] > valore) {
        vett[i] = vett[i-1];
        i--;
    }

    vett[i] = valore;
    return dim+1;
}

/*
 * Funzione insertion_sort(...)
 *
 * Ordina un vettore in modo crescente utilizzando la
 * funzione ausiliaria "inserisci_valore".
 *
 *       Algoritmo: Insertion Sort
 * Classificazione: algoritmo ingenuo, not in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n)
 *   AVERAGE CASE: O(n^2)
 *     WORST CASE: O(n^2)
 */

int insertion_sort(int dvett[], int svett[], int dim_s) {
    int dim_d = 0;

    for(int i = 0; i < dim_s; i++)
        dim_d = inserisci_valore(dvett, dim_d, svett[i]);

    return dim_d;
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {
    int uv[MAX_DIM] = {12, 4, 1, 1, 9};
    int ov[MAX_DIM];
    int dim_ov;

    dim_ov = insertion_sort(ov, uv, MAX_DIM);

    printf("Vettore ordinato:\n");
    stampa_vett(ov, dim_ov);
    return 0;
}
```

Insertion Sort senza funzione ausiliaria

```
#include <stdio.h>
#define MAX_DIM 5

/*
 * Funzione insertion_sort(...)
 *
 * Ordina un vettore in modo crescente.
 *
 *       Algoritmo: Insertion Sort
 * Classificazione: algoritmo ingenuo, not in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n)
 *   AVERAGE CASE: O(n^2)
 *     WORST CASE: O(n^2)
 */

int insertion_sort(int dvett[], int svett[], int dim_s) {
    int dim_d = 0;

    for(int i = 0; i < dim_s; i++) {
        int j = dim_d;
        while (j > 0 && dvett[j-1] > svett[i]) {
            dvett[j] = dvett[j-1];
            j--;
        }
        dvett[j] = svett[i];
        dim_d++;
    }
    return dim_d;
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {

    int uv[MAX_DIM] = {12, 4, 1, 1, 9};
    int ov[MAX_DIM];
    int dim_ov;


    dim_ov = insertion_sort(ov, uv, MAX_DIM);

    printf("Vettore ordinato:\n");
    stampa_vett(ov, dim_ov);

    return 0;
}
```

Ordinamento per selezione

Selection Sort

```
#include <stdio.h>
#define MAX_DIM 5

/*
 * Funzione selection_sort(...)
 *
 * Ordina un vettore in modo crescente.
 *
 *       Algoritmo: Selection Sort
 * Classificazione: algoritmo ingenuo, in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n^2)
 *   AVERAGE CASE: O(n^2)
 *     WORST CASE: O(n^2)
 */

void selection_sort(int vett[], int dim) {
    for(int i = 0; i < dim; i++) {
        int i_min = i;
        for(int j = i+1; j < dim; j++) {
            if (vett[j] < vett[i_min])
                i_min = j;
        }

        if (i != i_min) {
            int temp = vett[i];
            vett[i] = vett[i_min];
            vett[i_min] = temp;
        }
    }
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {

    int v[MAX_DIM] = {-30, 25, 30, 1, -10};

    selection_sort(v, MAX_DIM);

    stampa_vett(v, MAX_DIM);

    return 0;

}
```

Ordinamento per scambio

Bubble Sort

```
#include <stdio.h>
#define MAX_DIM 5

/*
 * Funzione bubble_sort(...)
 *
 * Ordina un vettore in modo crescente.
 *
 *       Algoritmo: Bubble Sort non ottimizzato
 * Classificazione: algoritmo ingenuo, in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n^2)
 *   AVERAGE CASE: O(n^2)
 *     WORST CASE: O(n^2)
 */

void bubble_sort(int vett[], int dim) {
    for(int i = 0; i < dim-1; i++) {
        for(int j = 0; j < dim-1; j++)
            if (vett[j+1] < vett[j]) {
                int temp = vett[j];
                vett[j] = vett[j+1];
                vett[j+1] = temp;
            }
    }
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {

    int v[MAX_DIM] = {-30, 25, 30, 1, -10};

    bubble_sort(v, MAX_DIM);

    stampa_vett(v, MAX_DIM);

    return 0;

}
```

Bubble Sort con sentinella

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_DIM 5

/*
 * Funzione bubble_sort(...)
 *
 * Ordina un vettore in modo crescente (variante con sentinella).
 *
 *       Algoritmo: Bubble Sort
 * Classificazione: algoritmo ingenuo, in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n)
 *   AVERAGE CASE: O(n^2)
 *     WORST CASE: O(n^2)
 */

void bubble_sort(int vett[], int dim) {
    bool scambi;
    do {
        scambi = false;
        for(int i = 0; i < dim-1; i++)
            if (vett[i+1] < vett[i]) {
                int temp = vett[i];
                vett[i] = vett[i+1];
                vett[i+1] = temp;
                scambi = true;
            }
    }
    while (scambi == true);
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {

    int v[MAX_DIM] = {-30, 25, 30, 1, -10};

    bubble_sort(v, MAX_DIM);

    stampa_vett(v, MAX_DIM);

    return 0;

}
```

Ordinamento avanzato

Quick Sort

```
#include <stdio.h>
#define MAX_DIM 10

/*
 * Funzione quick_sort(...)
 *
 * Ordina un vettore in modo crescente.
 *
 *       Algoritmo: Quick Sort
 * Classificazione: algoritmo avanzato, in-place
 *
 * Complessita' computazionale:
 *      BEST CASE: O(n*log(n))
 *   AVERAGE CASE: O(n*log(n))
 *     WORST CASE: O(n^2)
 */

void quick_sort(int vett[], int inf, int sup) {
    int sx = inf;
    int dx = sup;
    int pivot;

    pivot = vett[(inf + sup) / 2];

    while (sx <= dx) {
        while (vett[sx] < pivot) sx++;
        while (vett[dx] > pivot) dx--;

        if (sx < dx) {
            int temp = vett[sx];
            vett[sx] = vett[dx];
            vett[dx] = temp;
        }
        if (sx <= dx) {
            sx++;
            dx--;
        }
    }

    if (inf < dx)
        quick_sort(vett, inf, dx);

    if (sx < sup)
        quick_sort(vett, sx, sup);
}

void stampa_vett(int vett[], int dim) {
    for (int i=0; i < dim; i++) printf("%4d", vett[i]);
    printf("\n");
}

int main( void ) {
    int v[MAX_DIM] = {-30, 25, 30, 1, -10, 8, -4, 21, 14, -1};

    quick_sort(v, 0, MAX_DIM-1);
    stampa_vett(v, MAX_DIM);
    return 0;
}
```