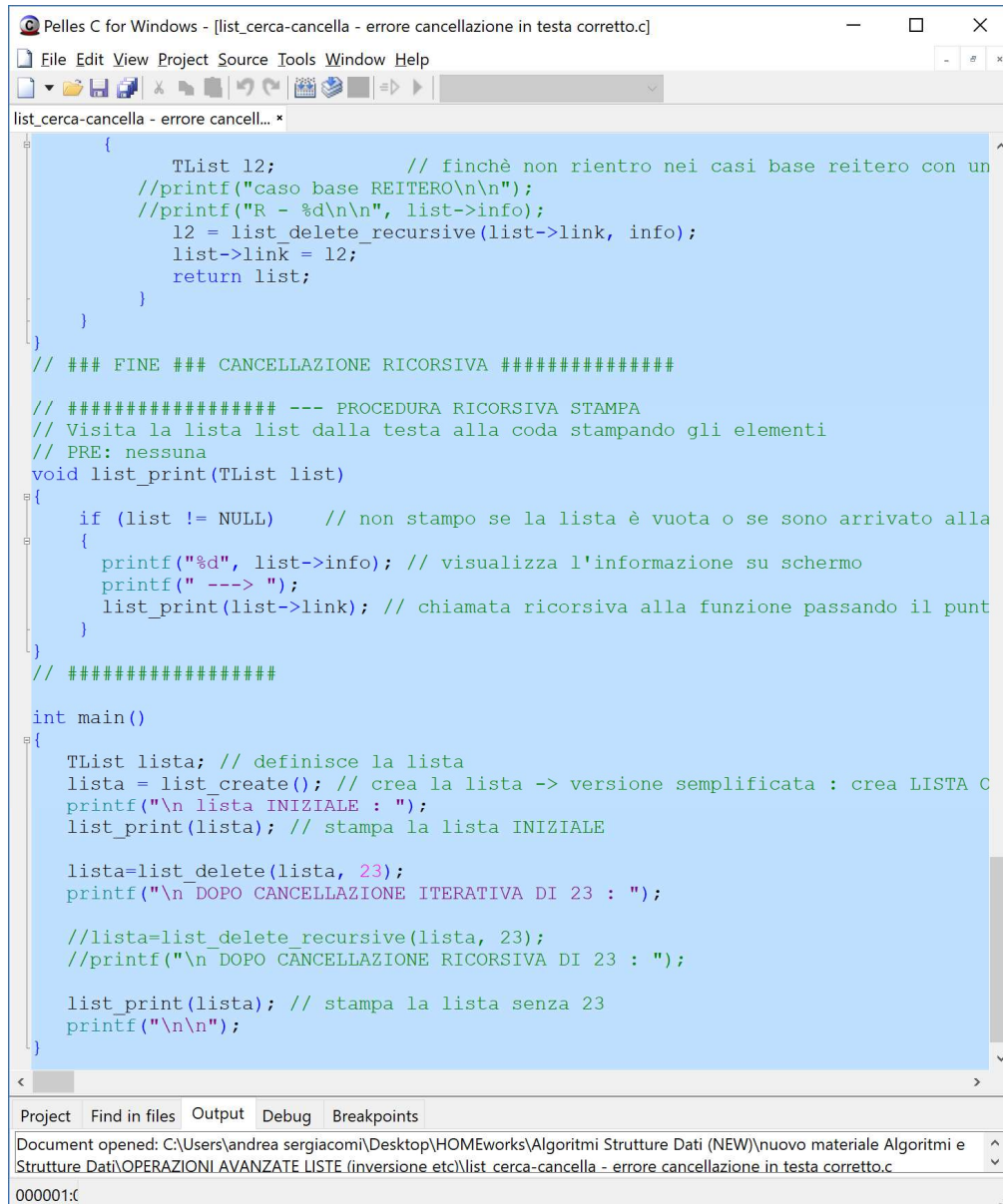


Liste dinamiche



```

Pelles C for Windows - [list_cerca-cancella - errore cancellazione in testa corretto.c]
File Edit View Project Source Tools Window Help
list_cerca-cancella - errore cancell... x

{
    TList l2;          // finchè non rientro nei casi base reitero con un
    //printf("caso base REITERO\n\n");
    //printf("R - %d\n\n", list->info);
    l2 = list_delete_recursive(list->link, info);
    list->link = l2;
    return list;
}

// ### FINE ### CANCELLAZIONE RICORSIVA #####

// ##### --- PROCEDURA RICORSIVA STAMPA
// Visita la lista list dalla testa alla coda stampando gli elementi
// PRE: nessuna
void list_print(TList list)
{
    if (list != NULL)    // non stampo se la lista è vuota o se sono arrivato alla
    {
        printf("%d", list->info); // visualizza l'informazione su schermo
        printf(" ---> ");
        list_print(list->link); // chiamata ricorsiva alla funzione passando il punt
    }
}

// #####

int main()
{
    TList lista; // definisce la lista
    lista = list_create(); // crea la lista -> versione semplificata : crea LISTA C
    printf("\n lista INIZIALE : ");
    list_print(lista); // stampa la lista INIZIALE

    lista=list_delete(lista, 23);
    printf("\n DOPO CANCELLAZIONE ITERATIVA DI 23 : ");

    //lista=list_delete_recursive(lista, 23);
    //printf("\n DOPO CANCELLAZIONE RICORSIVA DI 23 : ");

    list_print(lista); // stampa la lista senza 23
    printf("\n\n");
}

```

Project Find in files Output Debug Breakpoints

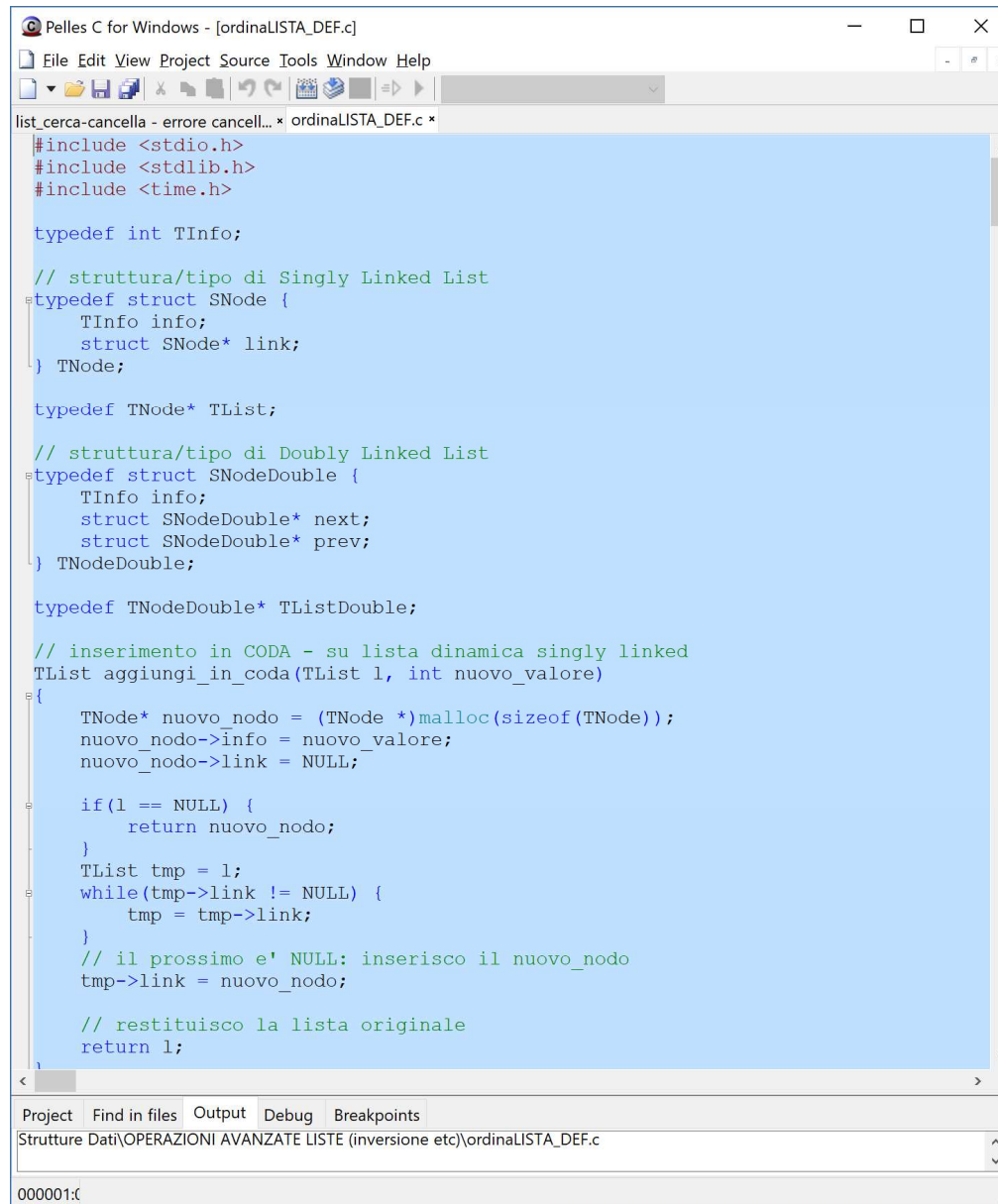
Document opened: C:\Users\andrea.sergiacomi\Desktop\HOMEworks\Algoritmi Strutture Dati (NEW)\nuovo materiale Algoritmi e Strutture Dati\OPERAZIONI AVANZATE LISTE (inversione etc)\list_cerca-cancella - errore cancellazione in testa corretto.c

000001:c

LIST_CERCA-CANCELLA -
errore cancellazione in testa
corretto.c

Correzione errore nella
funzione MAIN - evidenziato
dal vostro collega Alessio
Paolucci - nel richiamo delle
procedure iterative e
ricorsive di cancellazione di
un nodo nella lista (quando
effettuata in testa),
presentate nella precedente
esercitazione.

Liste dinamiche



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef int TInfo;

// struttura/tipo di Singly Linked List
typedef struct SNode {
    TInfo info;
    struct SNode* link;
} TNode;

typedef TNode* TList;

// struttura/tipo di Doubly Linked List
typedef struct SNodeDouble {
    TInfo info;
    struct SNodeDouble* next;
    struct SNodeDouble* prev;
} TNodeDouble;

typedef TNodeDouble* TListDouble;

// inserimento in CODA - su lista dinamica singly linked
TList aggiungi_in_coda(TList l, int nuovo_valore)
{
    TNode* nuovo_nodo = (TNode *)malloc(sizeof(TNode));
    nuovo_nodo->info = nuovo_valore;
    nuovo_nodo->link = NULL;

    if(l == NULL) {
        return nuovo_nodo;
    }
    TList tmp = l;
    while(tmp->link != NULL) {
        tmp = tmp->link;
    }
    // il prossimo e' NULL: inserisco il nuovo_nodo
    tmp->link = nuovo_nodo;

    // restituisco la lista originale
    return l;
}
```

ordinalISTA_DEF.c

- Complessità computazionale delle operazioni di inserimento in testa e in coda;
- Lavorare con liste dinamiche concatenate (singly linked) e con liste doppie (doubly linked);
- Ordinare liste disordinate (con algoritmi bubble e shaker sort);
- Misurare il tempo di esecuzione di un programma.

Liste dinamiche

ordinaLISTA_DEF.exe -> risultati possibili #1

```

C:\Users\andrea sergiacomì>"C:\Users\andrea sergiacomì\Desktop\HOMEworks\Algoritmi Strutture Dati (NEW)\nuovo materiale Algoritmi e Strutture Dati\OPERAZIONI AVANZATE LISTE
(inversione etc)\ordinaLISTA_DEF.exe"

7321 10 3 267 54 3421 24 2394 782 1 8843 23 5250 59 123 6304 345 3962 567 8 99 6727 987 1234 234 2947 345 67 6 15 4444 37 9376 44 9726 66
5169 88 11573 73 896 69 627 15839 482 7825 555 1000 843 2 7311 0 -7 257 44 3411 14 2384 772 -9 8833 13 5240 49 113 6294 335 3952 557 -2
89 6717 977 1224 224 2937 335 57 -4 5 4434 27 9366 34 9716 56 5159 78 11563 63 886 59 617 15829 472 7815 545 990 833 -8

-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a singolo verso -> INSERIMENTO IN CODA + ORDINAMENTO CON BUBBLE SORT): 0.038000 seconds

-8 833 990 545 7815 472 15829 617 59 886 63 11563 78 5159 56 9716 34 9366 27 4434 5 -4 57 335 2937 224 1224 977 6717 89 -2 557 3952 335 62
94 113 49 5240 13 8833 -9 772 2384 14 3411 44 257 -7 0 7311 2 843 1000 555 7825 482 15839 627 69 896 73 11573 88 5169 66 9726 44 9376 37
4444 15 6 67 345 2947 234 1234 987 6727 99 8 567 3962 345 6304 123 59 5250 23 8843 1 782 2394 24 3421 54 267 3 10 7321

-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a singolo verso -> INSERIMENTO IN TESTA + ORDINAMENTO CON BUBBLE SORT): 0.034000 seconds

-8 833 990 545 7815 472 15829 617 59 886 63 11563 78 5159 56 9716 34 9366 27 4434 5 -4 57 335 2937 224 1224 977 6717 89 -2 557 3952 335 62
94 113 49 5240 13 8833 -9 772 2384 14 3411 44 257 -7 0 7311 2 843 1000 555 7825 482 15839 627 69 896 73 11573 88 5169 66 9726 44 9376 37
4444 15 6 67 345 2947 234 1234 987 6727 99 8 567 3962 345 6304 123 59 5250 23 8843 1 782 2394 24 3421 54 267 3 10 7321

-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a doppio verso -> INSERIMENTO IN TESTA + ORDINAMENTO CON SHAKER SORT): 0.033000 seconds

C:\Users\andrea sergiacomì>
```


Liste dinamiche

ordinaLISTA_DEF.exe -> risultati possibili #2

```

C:\Users\andrea sergiacomì>"C:\Users\andrea sergiacomì\Desktop\HOMEworks\Algoritmi Strutture Dati (NEW)\nuovo materiale Algoritmi e Strutture Dati\OPERAZIONI AVANZATE LISTE
(inversione etc)\ordinaLISTA_DEF.exe"

7321 10 3 267 54 3421 24 2394 782 1 8843 23 5250 59 123 6304 345 3962 567 8 99 6727 987 1234 234 2947 345 67 6 15 4444 37 9376 44 9726 66
5169 88 11573 73 896 69 627 15839 482 7825 555 1000 843 2 7311 0 -7 257 44 3411 14 2384 772 -9 8833 13 5240 49 113 6294 335 3952 557 -2
89 6717 977 1224 224 2937 335 57 -4 5 4434 27 9366 34 9716 56 5159 78 11563 63 886 59 617 15829 472 7815 545 990 833 -8

-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a singolo verso -> INSERIMENTO IN CODA + ORDINAMENTO CON BUBBLE SORT): 0.039000 seconds

-8 833 990 545 7815 472 15829 617 59 886 63 11563 78 5159 56 9716 34 9366 27 4434 5 -4 57 335 2937 224 1224 977 6717 89 -2 557 3952 335 62
94 113 49 5240 13 8833 -9 772 2384 14 3411 44 257 -7 0 7311 2 843 1000 555 7825 482 15839 627 69 896 73 11573 88 5169 66 9726 44 9376 37
4444 15 6 67 345 2947 234 1234 987 6727 99 8 567 3962 345 6304 123 59 5250 23 8843 1 782 2394 24 3421 54 267 3 10 7321

-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a singolo verso -> INSERIMENTO IN TESTA + ORDINAMENTO CON BUBBLE SORT): 0.034000 seconds

-8 833 990 545 7815 472 15829 617 59 886 63 11563 78 5159 56 9716 34 9366 27 4434 5 -4 57 335 2937 224 1224 977 6717 89 -2 557 3952 335 62
94 113 49 5240 13 8833 -9 772 2384 14 3411 44 257 -7 0 7311 2 843 1000 555 7825 482 15839 627 69 896 73 11573 88 5169 66 9726 44 9376 37
4444 15 6 67 345 2947 234 1234 987 6727 99 8 567 3962 345 6304 123 59 5250 23 8843 1 782 2394 24 3421 54 267 3 10 7321

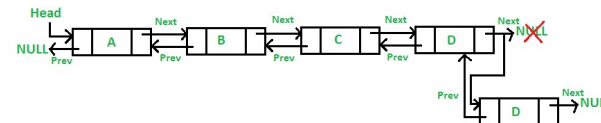
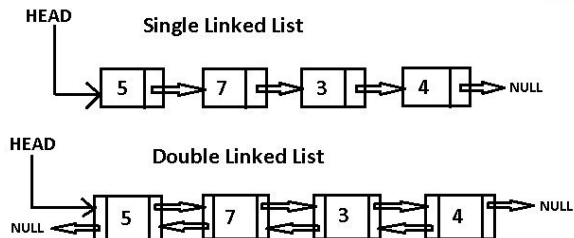
-9 -8 -7 -4 -2 0 1 2 3 5 6 8 10 13 14 15 23 24 27 34 37 44 44 49 54 56 57 59 59 63 66 67 69 73 78 88 89 99 113 123 224 234 257 26
7 335 335 345 345 472 482 545 555 557 567 617 627 772 782 833 843 886 896 977 987 990 1000 1224 1234 2384 2394 2937 2947 3411 3421 3952 3962
4434 4444 5159 5169 5240 5250 6294 6304 6717 6727 7311 7321 7815 7825 8833 8843 9366 9376 9716 9726 11563 11573 15829 15839

tempo di esecuzione Clock() (operazioni lista a doppio verso -> INSERIMENTO IN TESTA + ORDINAMENTO CON SHAKER SORT): 0.038000 seconds

C:\Users\andrea sergiacomì>"C:\Users\andrea sergiacomì\Desktop\HOMEworks\Algoritmi Strutture Dati (NEW)\nuovo materiale Algoritmi e Strutture Dati\OPERAZIONI AVANZATE LISTE
```

Liste dinamiche

| | Inserimento in testa | Inserimento in coda | Bubble Sort | Shaker Sort |
|---|----------------------------|----------------------------|-------------------------|-------------------------|
| Liste dinamiche concatenate (singly linked) | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)-\Theta(n^2)$ | $\Theta(n)-\Theta(n^2)$ |
| | T=0,034'' (BS) | T=0,038''- 0,039'' (BS) | BS | |
| liste doppie (doubly linked) | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)-\Theta(n^2)$ | $\Theta(n)-\Theta(n^2)$ |
| | T=0,033''- 0,038'' (SS) | | | SS |



```

for (i = 0; i < size; i++) BS
{
    for (j = 0; j < size - i; j++)
    {
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
    }
}

```

```

for (p = 1; p <= n / 2; p++) SS
{
    for (i = p - 1; i < n - p; i++)
        if (a[i] > a[i+1])
            swap(&a[i], &a[i+1]);
    for (i = n - p - 1; i >= p; i--)
        if (a[i] < a[i-1])
            swap(&a[i], &a[i-1]);
}

```

Liste dinamiche

ordinaLISTA_DEF.c -> CONCLUSIONI

- Complessità computazionale e tempo di esecuzione non sono la stessa cosa: un conto è l'andamento asintotico teorico di un'operazione, un conto è la reale misurazione empirica di variabili spaziali-temporali in termini di consumo di risorse condivise (storage, CPU, RAM, multi-threading, ...)
- Il tempo misurato con Clock() non è preciso su piccole scale (considera secondi ma non i microsecondi o i nanosecondi) quindi poco affidabile se non prendiamo in considerazione grandi moli di dati e/o strutture dati da ordinare caratterizzate da precise distribuzioni dei dati (es. già ordinate). Implementazioni diverse non sono agevoli (non funzionano automaticamente su tutti i sistemi): GetTimeOfDay -> deprecated, Clock_GetTime() e timespec -> valido su sistemi POSIX compliant (ok Linux, ma non Windows nativamente e non MacOS), Std::Chrono -> solo su C++

Liste dinamiche



```
LISTE_operazioni_avanzate.c
listaT=aggiungi_in_testaSINGLY(listaT, 142);
listaT=aggiungi_in_testaSINGLY(listaT, 67);

printf("\n");
printf("lista TS\n");
stampa_lista(listaTS);

printf("lista T\n");
stampa_lista(listaT);
printf("\n\n");

printf("minimo TS (proc. iterativa): %d\n",list_min(listaTS)->info);
printf("minimo T (proc. ricorsiva): %d\n",list_min_recu(listaT)->info);
printf("\n");
printf("n. nodi TS (proc. iterativa): %d\n",list_count_nodes(listaTS));
printf("n. nodi T (proc. ricorsiva): %d\n",list_count_nodes_recu(listaT));
printf("\n");
printf("somma nodi TS (proc. iterativa): %d\n",list_sum_nodes(listaTS));
printf("somma nodi T (proc. ricorsiva): %d\n",list_sum_nodes_recu(listaT));
printf("\n");
listaTS=list_reverse(listaTS);
printf("lista TS invertita iterativamente\n");
stampa_lista(listaTS);

listaT=list_reverse_recu(listaT);
printf("lista T invertita ricorsivamente\n");
stampa_lista(listaT);
printf("\n");
copyTS=list_copy(listaTS);
printf("lista copyTS copiata iterativamente da TS\n");
stampa_lista(copyTS);

copyT=list_copy_recu(listaT);
printf("lista copyT copiata ricorsivamente da T\n");
stampa_lista(copyT);
printf("\n");

printf("fondo copyTS e copyT in un'unica lista ... il risultato del merge è ");
copy=list_merge_recursive(copyTS , copyT);
stampa_lista(copy);
```

LISTE_operazioni_avanzate.
C

Implementazioni iterative e
ricorsive di operazioni
avanzate con le liste: ricerca
del minimo, conteggio nodi,
somma dei valori nei nodi,
inversione, copia, fusione di
liste.

Liste dinamiche

LISTE_operazioni_avanzate.exe -> risultato

```

C:\Users\andrea sergiacomì> "C:\Users\andrea sergiacomì\Desktop\HOMEworks\Algoritmi Strutture Dati (NEW)\nuovo materiale
Algoritmi e Strutture Dati\OPERAZIONI AVANZATE LISTE (inversione etc)\LISTE_operazioni_avanzate.exe"

lista TS
1 782 2394 24 3421 54 267 3 10 7321
lista T
67 142 529 30 2

minimo TS (proc. iterativa): 1
minimo T (proc. ricorsiva): 2

n. nodi TS (proc. iterativa): 10
n. nodi T (proc. ricorsiva): 5

somma nodi TS (proc. iterativa): 14277
somma nodi T (proc. ricorsiva): 770

lista TS invertita iterativamente
7321 10 3 267 54 3421 24 2394 782 1
lista T invertita ricorsivamente
2 30 529 142 67

lista copyTS copiata iterativamente da TS
7321 10 3 267 54 3421 24 2394 782 1
lista copyT copiata ricorsivamente da T
2 30 529 142 67

fondo copyTS e copyT in un'unica lista ... il risultato del merge è la lista copy:
2 30 529 142 67 7321 10 3 267 54 3421 24 2394 782 1

C:\Users\andrea sergiacomì>_

```


Liste dinamiche



```
// INIZIO ORDINAMENTO BUBBLE SORT SU LISTA SINGLY LINKED
begin = clock();
listaRIS1=bubble_sort_lista(listaTS_COP);
for (cont=1;cont<2000;cont++)
{
    printf("%d ", cont-cont);
}
printf("\n\n");
end = clock();
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("tempo di esecuzione Clock() (lista disordina");
printf("\n");
// FINE ORDINAMENTO BUBBLE SORT SU LISTA SINGLY LINKED D
printf("dopo ordinamento: ");
stampa_lista(listaRIS1);
printf("\n\n");

// INIZIO ORDINAMENTO BUBBLE SORT SU LISTA SINGLY LINKED
begin = clock();
listaRIS2=bubble_sort_lista(listaTS_ORD_COP);
for (cont=1;cont<2000;cont++)
{
    printf("%d ", cont-cont);
}
printf("\n\n");
end = clock();
time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("tempo di esecuzione Clock() (lista già ordin");
printf("\n");
// FINE ORDINAMENTO BUBBLE SORT SU LISTA SINGLY LINKED G
printf("dopo ordinamento: ");
stampa_lista(listaRIS2);
printf("\n\n");

// INIZIO ORDINAMENTO BUBBLE SORT SU LISTA SINGLY LINKED
begin = clock();
listaRIS3=bubble_sort_lista(listaTS_INV_COP);
for (cont=1;cont<2000;cont++)
{
    printf("%d ", cont-cont);
}
printf("\n\n");
end = clock();
```

tempi esecuzione BUBBLE.c

Tempo di esecuzione
(corretto) di operazioni di
ordinamento crescente -
tramite bubble sort - di liste:

- CASUALMENTE
DISORDINATE;
- GIA' ORDINATE;
- INVERTITE (in ordine
decrescente);

Liste dinamiche

tempi esecuzione BUBBLE.c -> possibile risultato

```
tempo di esecuzione Clock() (lista disordinata -> ORDINAMENTO CON BUBBLE SORT): 0.402000 seconds
```

```
tempo di esecuzione Clock() (lista già ordinata -> ORDINAMENTO CON BUBBLE SORT): 0.401000 seconds
```

```
tempo di esecuzione Clock() (lista invertita -> ORDINAMENTO CON BUBBLE SORT): 0.435000 seconds
```