Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

Soluzioni ricorsive per l'ordinamento di array

Ordinamento per selezione

dato un vettore $\mathbf{v}$ di reali, ordinare il vettore implementando una soluzione ricorsiva che sfrutti l'ordinamento per selezione.

(Una) versione iterativa:

```
float idx_min_array(float v[], int n, int start)
{
    int idx_min = start;
    float min   = v[start];
    for (int i=start+1; i<n; i++)
        if (v[i] < min)
        {
            min     = v[i];
            idx_min = i;
        }
    return idx_min;
}

void selection_sort(float v[], int n)
{
    int idx_min;
    for(int i=0; i<n-1; i++) // n-1, in quanto ultimo elemento,
                             // sarà già nella posizione corretta
                             // grazie agli spostamenti precedenti
    {
        idx_min    = idx_min_array(v, n, i);
        float tmp  = v[i];
        v[i]       = v[idx_min];
        v[idx_min] = tmp;
    }

}
```

Provo a trovare una soluzione ricorsiva.

1. esistono dei casi in cui il mio problema è facilmente risolvibile?

- Se $\mathbf{v}$ ha un solo elemento, la versione ordinata del vettore è banalmente $\mathbf{v}$ stesso. Quindi,
  - se $\#\mathbf{v} = 1$, $selection\_sort(\mathbf{v}) = \mathbf{v}$
- Se $\mathbf{v}$ ha due elementi, i.e.$\mathbf{v} = (v_1, v_2)$, la versione ordinata di $\mathbf{v}$ sarà:
  - se $\min(v_1, v_2) = v_1$, $\mathbf{v}^{sorted} = (v_1, v_2)$
  - se $\min(v_1, v_2) = v_2$, $\mathbf{v}^{sorted} = (v_2, v_1)$.

  Posso quindi definire la funzione selection_sort per un vettore di due elementi come:
  - $selection\_sort(\mathbf{v}) = (v_{idx\_min}, v_j)$ con $v_{idx\_min} \leq v_j$.

2. posso sfruttare le soluzioni ai casi banali individuati per costruire soluzioni ad un caso un po' più complesso?

- analizziamo il caso in cui $\mathbf{v} = (v_1, v_2, v_3)$.

In questo caso, se si vuole sfruttare il principio di base del Selection Sort (ossia la ricerca del minimo), la versione ordinata del vettore la posso esprimere come il vettore composto dalla concatenazione di:

- v_{idx_min}, con idx_min indice del valore minimo in $\mathbf{v}$
- l'ordinamento del sottoarray formato dalle 2 componenti rimanenti, che so calcolarlo.

Indicando con $\mathbf{v}^{i \neq t}$ il vettore composto da tutte le componenti di $\mathbf{v}$ tranne la t-esima (ossia $\mathbf{v}^{i \neq t} = (v_1, v_2, \ldots, v_{t-1}, v_{t+1}, \ldots, v_{\#\mathbf{v}})$) posso riscrivere la funzione $selection\_sort(\mathbf{v})$ su array di dimensione 3 come:

$$selection\_sort(\mathbf{v}) = concat\big(v_{idx\_min}, selection\_sort(\ \mathbf{v}^{i \neq idx\_min}\ )\big)$$

con $concat(\ldots)$ funzione che restituisce la concatenazione dei due vettori dati in input.

3. posso sfruttare la mia nuova soluzione per costruire una soluzione ad un caso ancora più complesso?

Posso ripetere il ragionamento di sopra per array di dimensione 4.

4. posso generalizzare?

Sì, posso definire quindi la funzione Selection Sort per un array di lunghezza arbitraria:

$$selection\_sort(\mathbf{v}):$$

$$\mathbf{v}_{sorted} = \begin{cases} v_1, & \text{se } \#\mathbf{v} = 1 \\ concat\big(v_{idx\_min}, selection\_sort(\mathbf{v}_{1,\ldots,idx\_min-1,idx\_min+1,\ldots,\#\mathbf{v}})\big), & \text{se } \#\mathbf{v} > 1 \end{cases}$$

// soluzione NON ricorsiva con funzioni ricorsive

include <stdio.h>

define MAX_LEN 15

void selection_sort(int v[], int n, int i, int k, int tmin) { if(k == n) return; if(i == n) { int t =v[k]; v[k] = v[tmin]; v[tmin] = t;

```
k++;
i    = k;
tmin = k;
```

```
    }
    if (v[i] < v[tmin])
    {
        tmin = i;
    }
    selection_sort(v, n, ++i, k, tmin);


}
```

void stampa_vett(int v[], int n) { printf("("); for(int i = 0; i < n; i++) { printf("%2d, ",v[i]); } printf("\b\b)\n");
}

int main() { int v[MAX_LEN] = {5,4,1,2,3,9,7,8,10,4,6,0,9,11,15}; selection_sort(v,MAX_LEN,0,0,0);
stampa_vett(v,MAX_LEN); return 0; }

In [4]:
```
// soluzione ricorsiva
#include <stdio.h>
#define MAX_LEN 15

int idx_min(int v[], int n)
{
    int idx = 0;
    for(int i=1; i < n; i++)
        if (v[i] < v[idx])
            idx = i;
    return idx;
}

void swap(int* a, int* b)
{
    int t = *a;
    *a    = *b;
    *b    = t;
}

void selection_sort(int v[], int n)
{
   if(n==0 || n==1)
       return;

    int idx = idx_min(v,n);

    swap(&v[0], &v[idx]);
    selection_sort(v+1, n-1);
}

void stampa_vett(int v[], int n)
{
    printf("(  ");
    for(int i = 0; i < n; i++)
    {
        printf("%2d, ",v[i]);
    }
    printf("\b\b)\n");
}

int main()
{
    int v[MAX_LEN] = {5,4,1,2,3,9,7,8,10,4,6,0,9,11,15};
    selection_sort(v,MAX_LEN);
    stampa_vett(v,MAX_LEN);
    return 0;
}
```

```
(   0,  1,  2,  3,  4,  4,  5,  6,  7,  8,  9,  9, 10, 11, 15)
```

Merge sort

Ricordiamo la funzione di fusione tra due vettori ordinati:

```
void fondi(int v1[], int n1,
           int v2[], int n2,
           int v_out[],int* n_out)
{
    int idx_v1    = 0;
    int idx_v2    = 0;
    int idx_v_out = 0;
    while(idx_v1 < n1 && idx_v2 < n2)
    {
        if (v1[idx_v1] < v2[idx_v2])
        {
            v_out[idx_v_out] = v1[idx_v1];
            idx_v1++;
        }
        else
        {
            v_out[idx_v_out] = v2[idx_v2];
            idx_v2++;
        }
        idx_v_out++;
    }

    if(idx_v1 < n1)
    {
        for(int i=idx_v1; i < n1; i++)
        {
            v_out[idx_v_out] = v1[i];
            idx_v_out++;
        }
    }

    if(idx_v2 < n2)
    {
        for(int i = idx_v2; i < n2; i++)
        {
            v_out[idx_v_out] = v2[i];
            idx_v_out++;
        }
    }
    *n_out = idx_v_out;

}
```

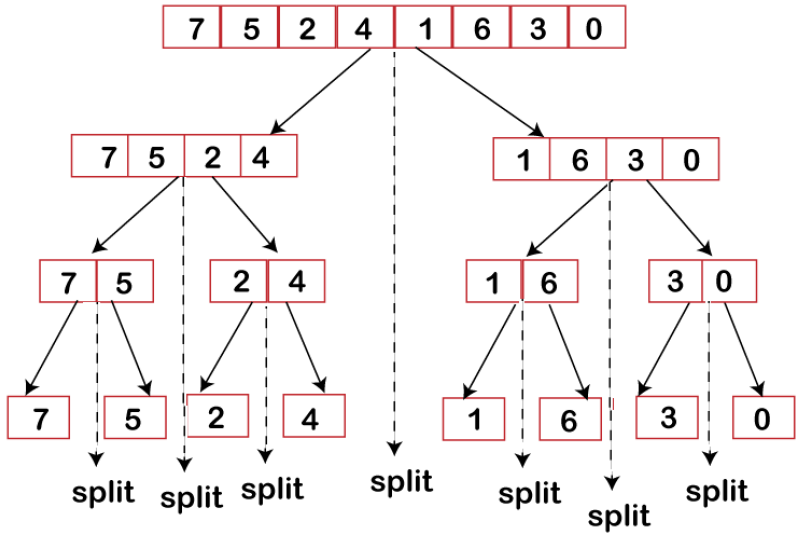
- Un vettore di un solo elemento è ordinato $\implies$ la funzione di merge di vettori ordinati è definita anche se gli input sono due array di un solo elemento ciascuno

Principio:

1. divido il vettore in due parti in maniera ricorsiva
2. ordino le due parti
3. fondo le due parti ordinate

$$mergeSort(\mathbf{v}):$$

$$\mathbf{v}_{sorted} = \begin{cases} \mathbf{v} & \text{se } \#\mathbf{v} = 1 \\ fondi(mergeSort(\mathbf{v}_{1\ldots\lfloor v/2 \rfloor}), mergeSort(mergeSort(\mathbf{v}_{\lfloor v/2 \rfloor+1,\ldots\#\mathbf{v}}))) & \text{altriment} \end{cases}$$



In [1]:
```
#include <stdio.h>
#define MAX_LEN 100


void stampa_vett(int v[],int n)
{
    printf("(");
    for(int i = 0; i < n; i++)
    {
        printf(" %d, ", v[i]);

    }
    printf("\b\b)\n");
}

void fondi(int v1[], int n1,
           int v2[],int n2,
           int v_out[],int* n_out)
{
    int idx_v1    = 0;
    int idx_v2    = 0;
    int idx_v_out = 0;
    while(idx_v1 < n1 && idx_v2 < n2)
    {
        if (v1[idx_v1] < v2[idx_v2])
        {
            v_out[idx_v_out] = v1[idx_v1];
            idx_v1++;
        }
        else
        {
            v_out[idx_v_out] = v2[idx_v2];
            idx_v2++;
        }
        idx_v_out++;
    }

    if(idx_v1 < n1)
    {
        for(int i=idx_v1; i < n1; i++)
        {
            v_out[idx_v_out] = v1[i];
            idx_v_out++;
        }
    }

    if(idx_v2 < n2)
    {
        for(int i = idx_v2; i < n2; i++)
        {
            v_out[idx_v_out] = v2[i];
            idx_v_out++;
        }
    }
    *n_out = idx_v_out;

}

void array_copy(int v[], int n,
                int out[])
{
    for(int i=0; i < n; i++)
        out[i] = v[i];
}

void merge_sort(int v[], int n)
{
    if (n == 1)
        return;
    merge_sort(v, n/2);
    merge_sort(v+n/2, n/2+n%2);

    int out[MAX_LEN];
    int n_out;
    fondi(v, n/2, v+n/2, n/2+n%2,
          out, &n_out);
    array_copy(out, n_out, v);

}


int main()
{
    int v[] = {10,9,8,7,6,5,4,3,2,1,0};//{5,10,4,2,1,6,8,7,9,0,3};
    int n   = 11;
    printf("prima: ");
    stampa_vett(v, n);

    merge_sort(v, n);

    printf("dopo:  ");
    stampa_vett(v, n);

}
```

```
prima: ( 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0)
dopo:  ( 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  10)
```