

Lezione 7

QuickSort su interi e stringhe

Rossano Venturini

rossano@di.unipi.it

Esercizio 3

Insertion Sort su stringhe

Scrivere una funzione che, dato un array di stringhe e la sua lunghezza, lo ordini utilizzando l'algoritmo **Insertion Sort**.

Scrivere un programma che utilizzi la funzione per ordinare un array di N stringhe lette da input e stampi in output gli elementi dell'array ordinato. Assumere che la lunghezza massima di una stringa sia 100 caratteri.

Si può utilizzare la funzione `strcmp` in `string.h` per confrontare lessicograficamente due stringhe. Utilizzare il comando `man strcmp` per maggiori informazioni.

La prima riga dell'input contiene la dimensione N dell'array. Le righe successive contengono gli elementi dell'array, una stringa per riga. Ogni stringa ha lunghezza massima di 100 caratteri.

L'output contiene gli elementi dell'array ordinato, una stringa per riga.

Esercizio 3

```
void insertionSort(char **A, int len) {  
    int i, j;  
    char *key;  
  
    for(i = 1; i < len; i++) {  
        key = A[i];  
        j = i - 1;  
        while (( j >= 0 ) && (strcmp(A[j], key) > 0)) {  
            A[j+1] = A[j]; // scambia i puntatori  
            j--;  
        }  
        A[j+1] = key;  
    }  
}
```

Esercizio 4

Ricerca binaria su stringhe

Scrivere una funzione che, data una stringa, un array di stringhe distinte e ordinate lessicograficamente e la sua lunghezza, cerchi la stringa nell'array utilizzando la ricerca binaria. La funzione restituisce la posizione della stringa se essa è presente, il valore -1 altrimenti.

Scrivere un programma che implementi il seguente comportamento. L'input è formato da una prima riga contenente la lunghezza N dell'array. Le successive N righe contengono le stringhe dell'array ordinate lessicograficamente.

Segue una sequenza di dimensione non nota di richieste espresse con coppie. La prima riga di ogni coppia è un valore che può essere "0" o "1". Se il valore è 0, il programma termina (non ci sono più richieste). Se il valore è "1", sulla riga successiva si trova una stringa da cercare.

Per ciascuna richiesta ci si aspetta in output l'esito della ricerca: la posizione della stringa nell'array se essa è presente, -1 altrimenti.

Esercizio 4

```
int binsearchr(char **dict, int left, int right, char *str) {  
  
    int pos = (left+right)/2; // floor  
  
    if (left > right) {  
        return -1;  
    }  
    int cmp = strcmp(str, dict[pos]);  
  
    if (cmp == 0) return pos;  
    if (cmp < 0) {  
        return binsearchr(dict, left, pos-1, str);  
    } else {  
        return binsearchr(dict, pos+1, right, str);  
    }  
}
```

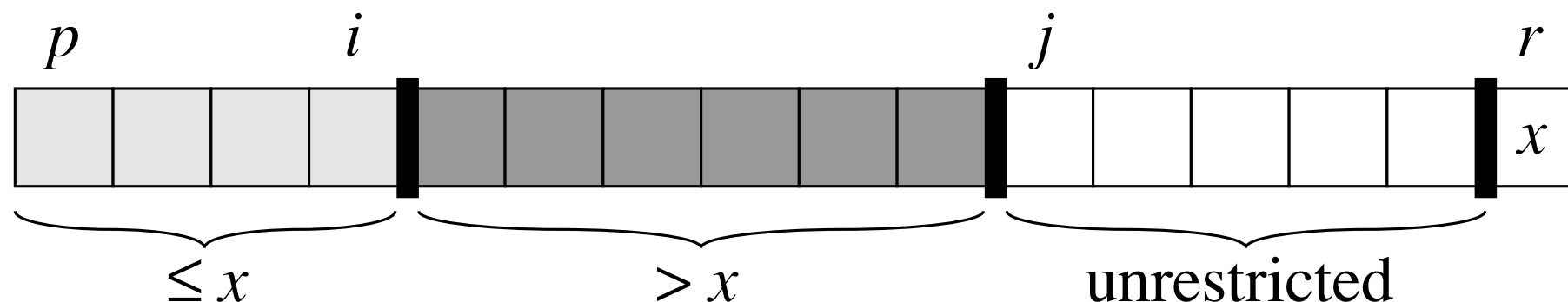
QuickSort parziale

```
void quicksort( int a[], int sx, int dx ) {  
  
    int perno, pivot;  
    if( sx < dx ) {  
        // Da implementare! scelta del pivot.  
        // Scegliere una posizione a caso tra sx e dx inclusi.  
        pivot = ???;  
        perno = distribuzione(a, sx, pivot, dx);  
        // separa gli elementi minori di a[pivot]  
        // da quelli maggiori o uguali  
        /* Ordina ricorsivamente le due metà */  
        quicksort(a, sx, perno-1);  
        quicksort(a, perno+1, dx);  
    }  
}
```

Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```
1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
```

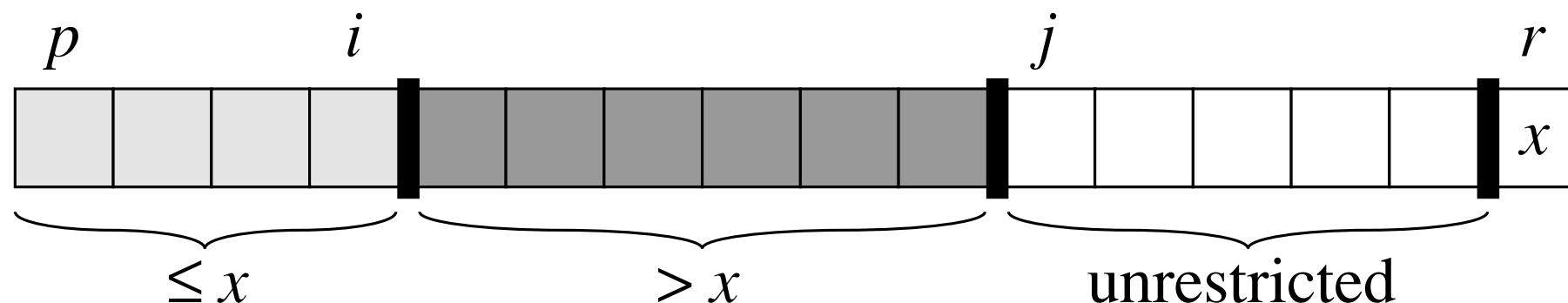


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```
1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
```

i	p	j						r
	2	8	7	1	3	5	6	4

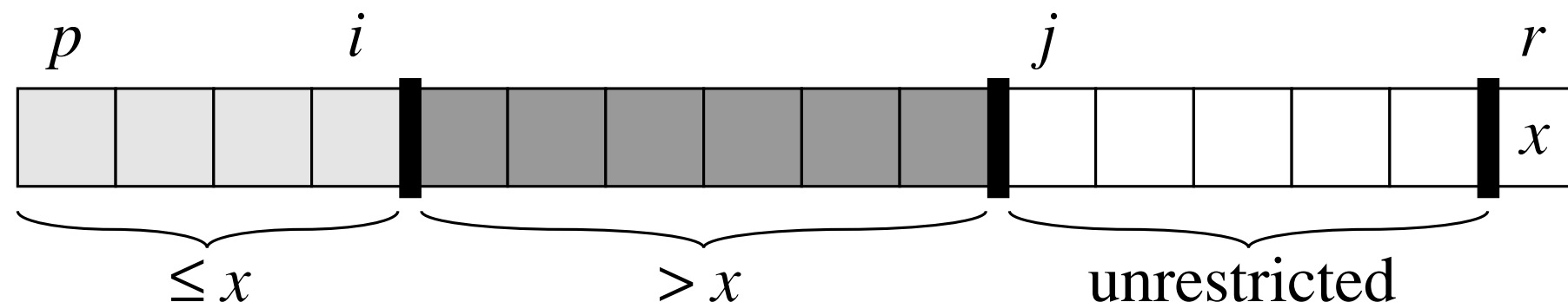
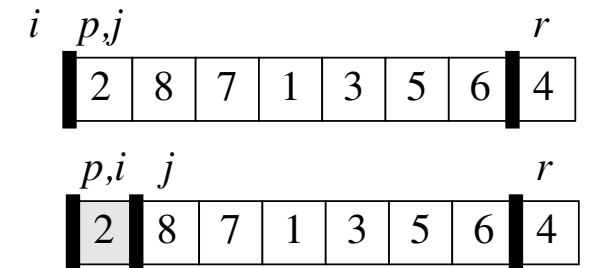


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

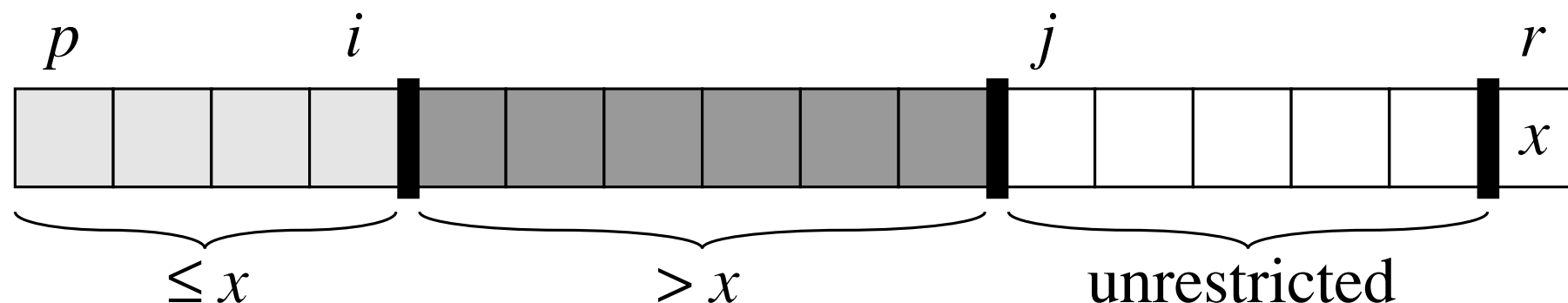
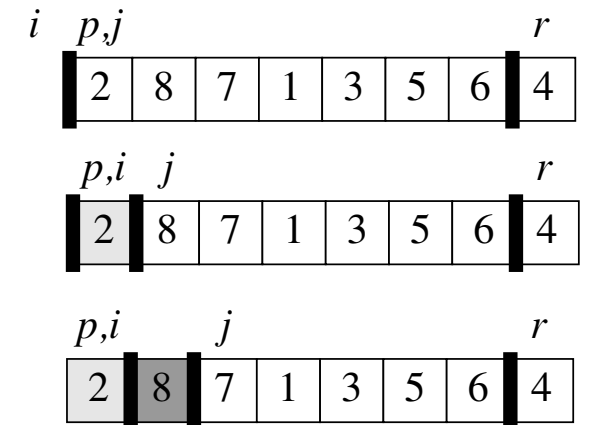


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

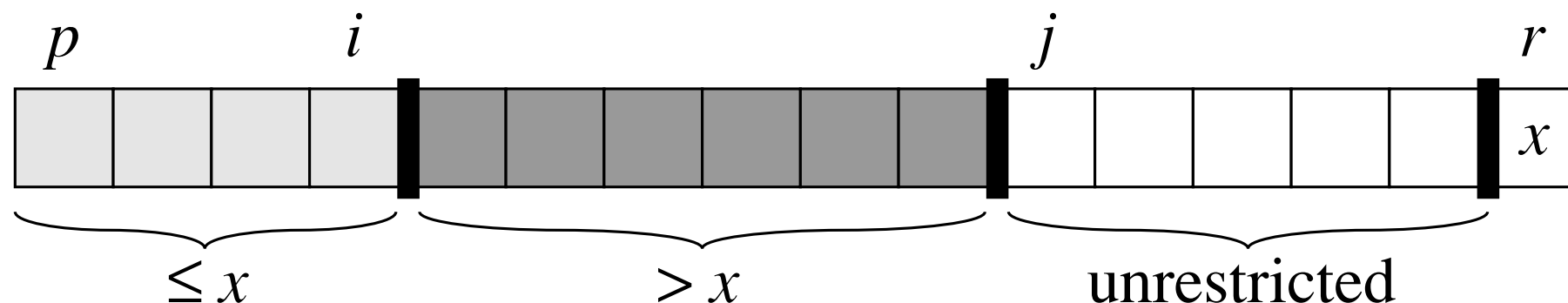
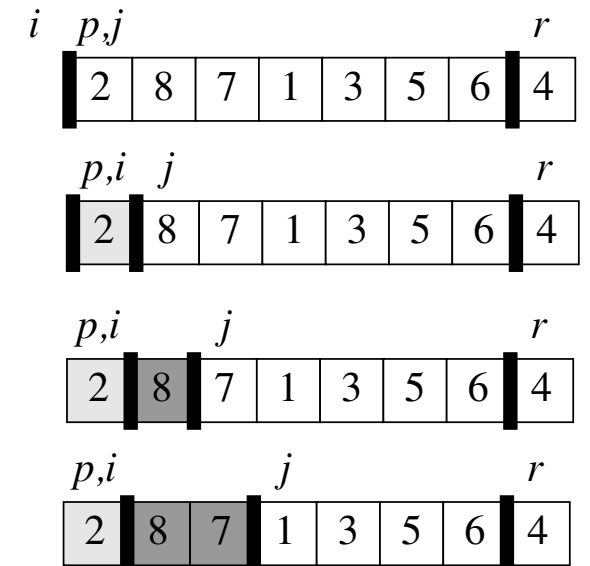


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

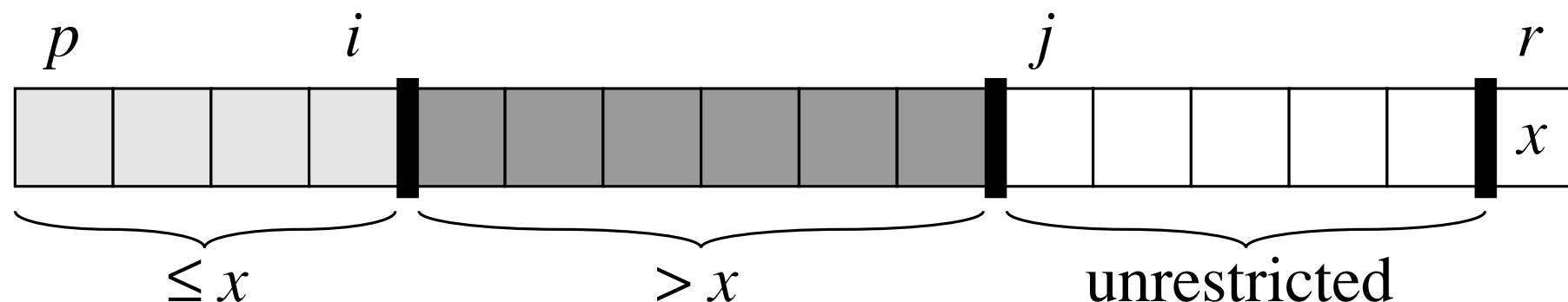
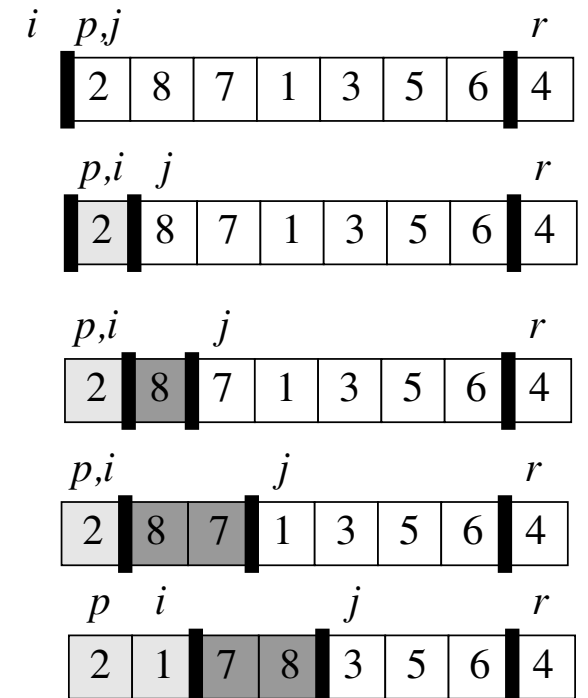


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

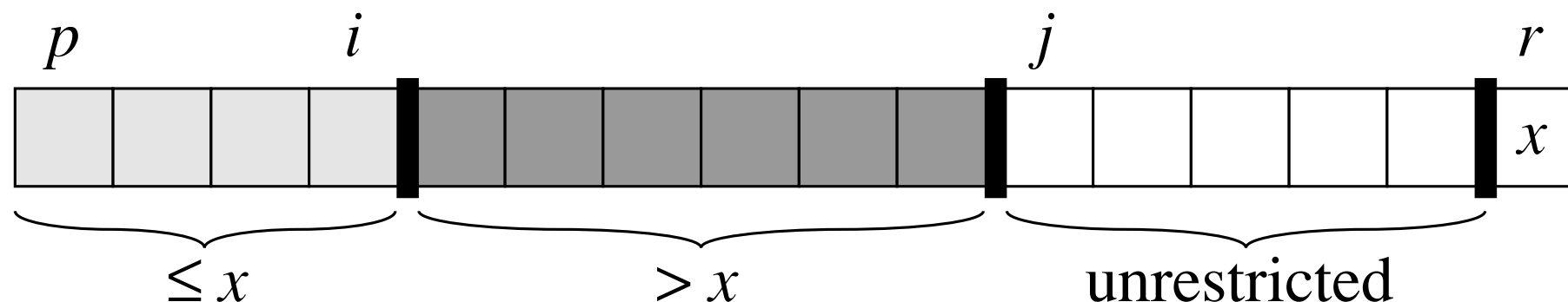
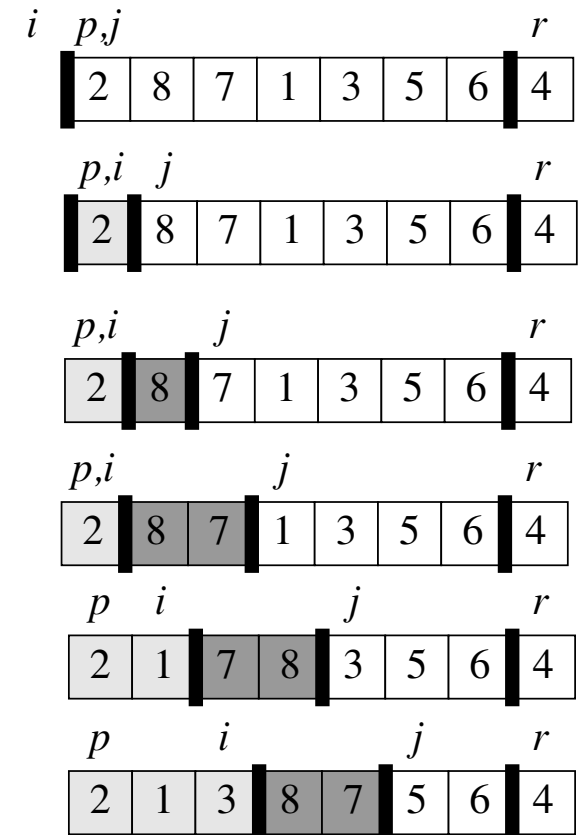


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

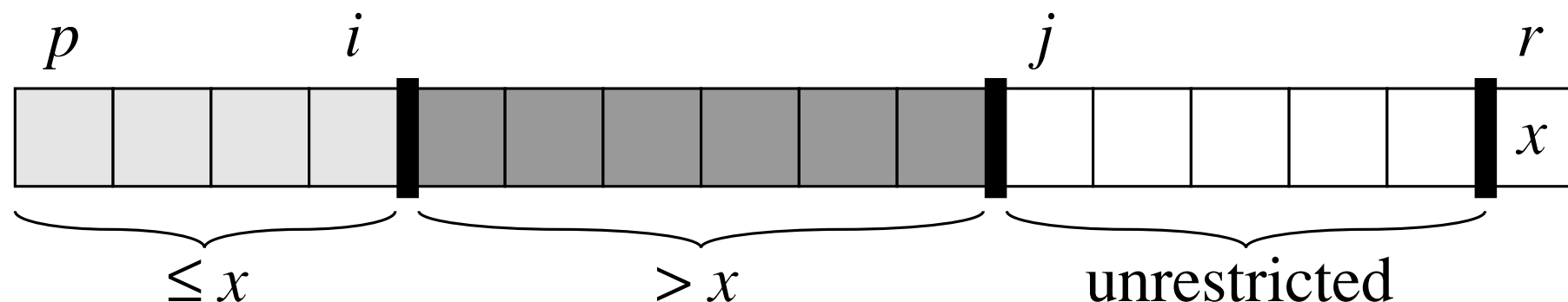
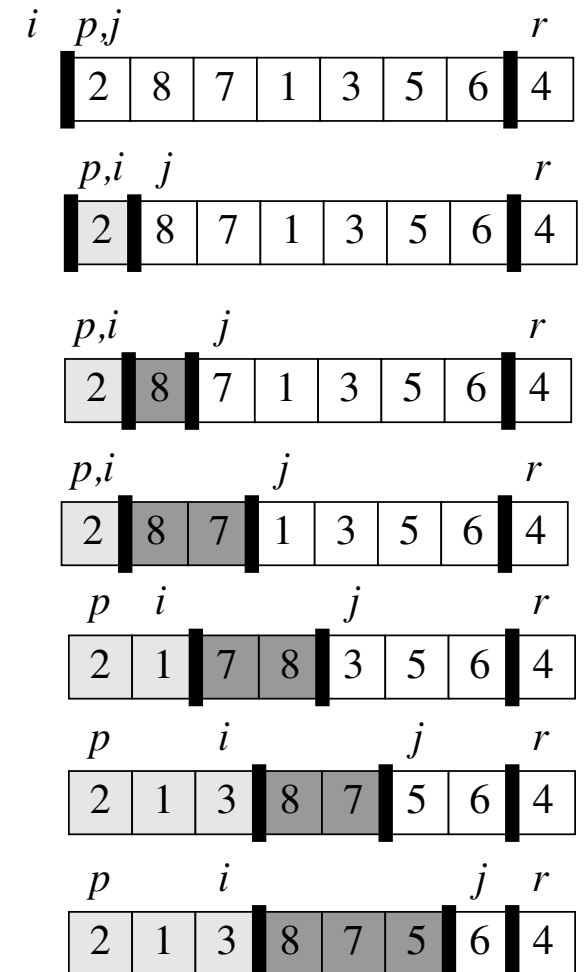


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

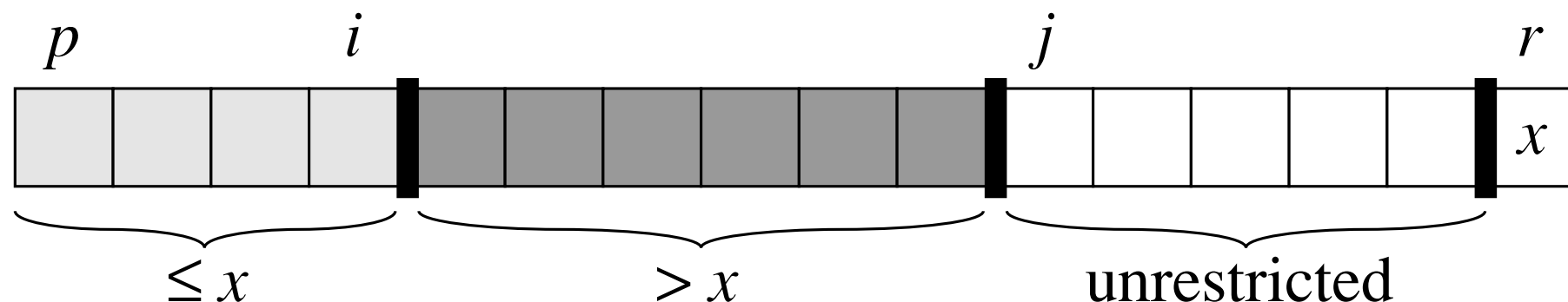
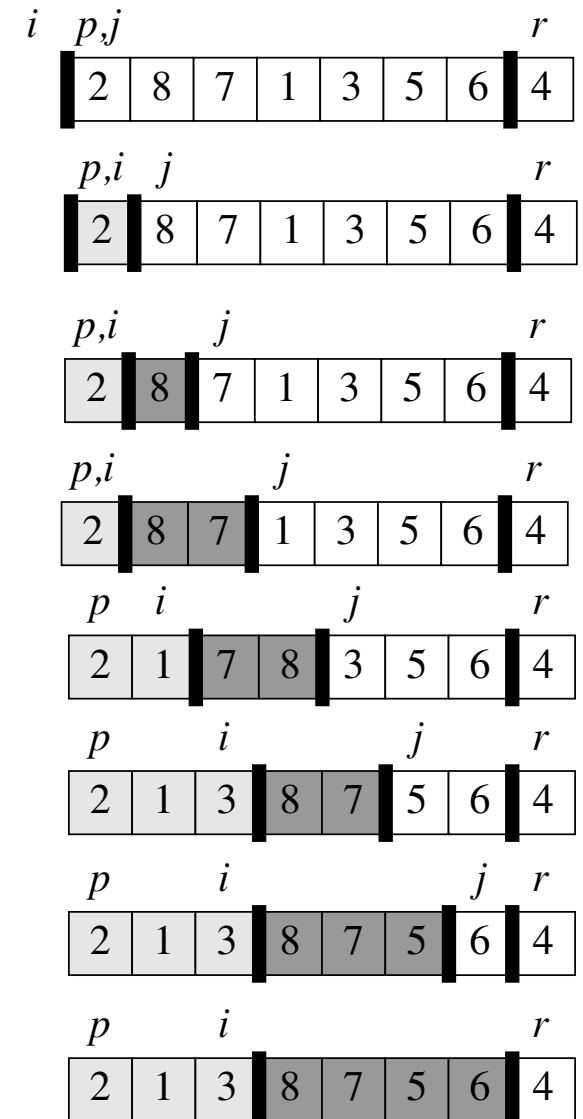


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```

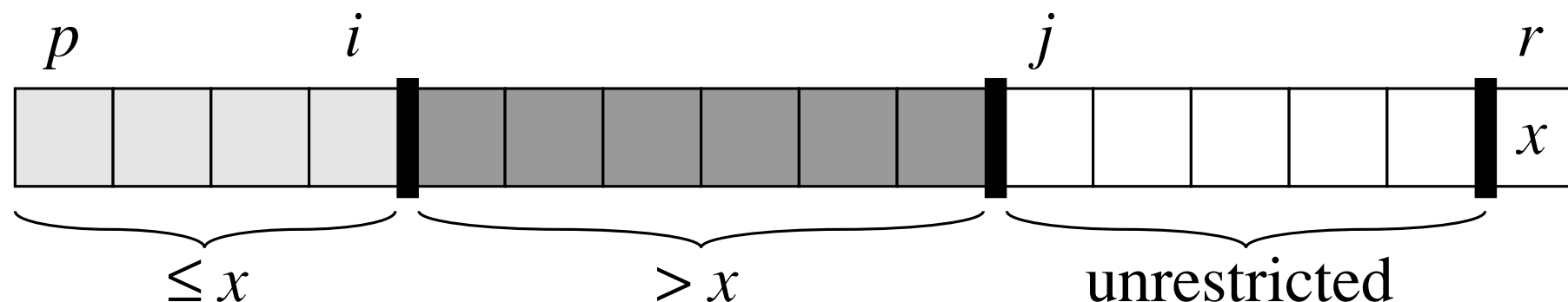
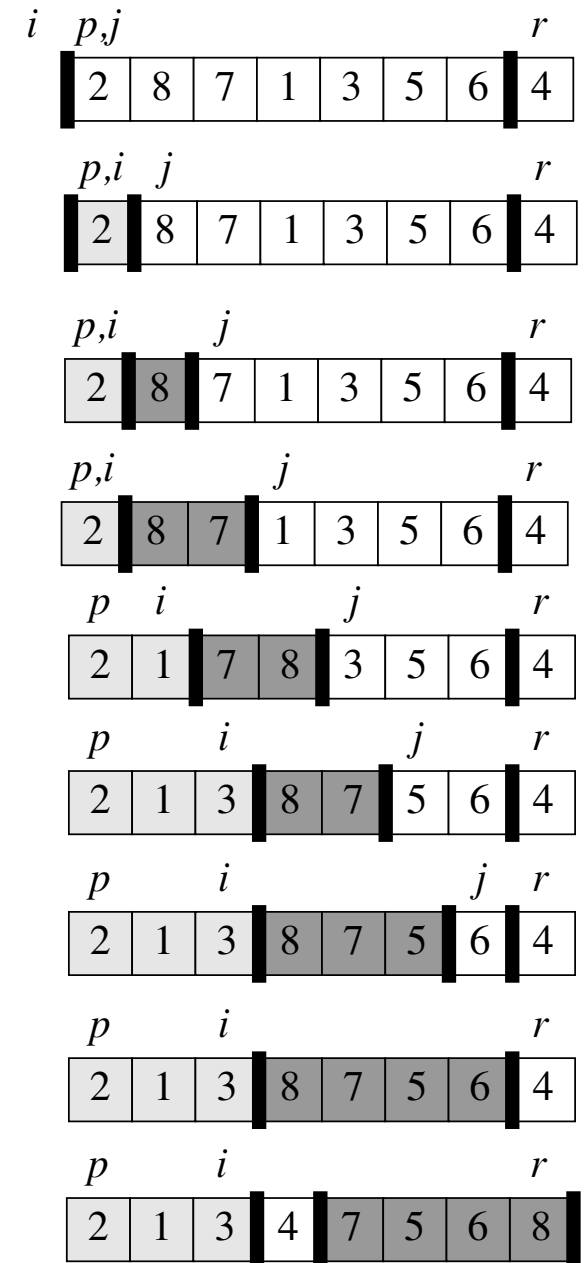


Distribuzione del QuickSort

PARTITION(A, p, r) // assume che il pivot sia in posizione r

```

1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  for  $j \leftarrow p$  to  $r - 1$ 
4      do if  $A[j] \leq x$ 
5          then  $i \leftarrow i + 1$ 
6              exchange  $A[i] \leftrightarrow A[j]$ 
7  exchange  $A[i + 1] \leftrightarrow A[r]$ 
8  return  $i + 1$ 
    
```



Esercizio 1

QuickSort su interi

Scaricare il sorgente `quicksort_parziale.c` che si trova sulla pagina del corso:

`http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/informatica/all-b/quicksort_parziale.c.zip`

Questo sorgente contiene un'implementazione di Quicksort che deve essere completata scrivendo il corpo della funzione

```
int distribuzione(int a[], int sx, int px, int dx)
```

Tale funzione deve partizionare gli elementi dell'array $a[sx \dots dx]$ utilizzando l'elemento $a[px]$ come pivot e restituire la posizione di tale elemento dopo il partizionamento.

La prima riga dell'input contiene la dimensione N (non limitata) dell'array.

Le righe successive contengono gli elementi dell'array, uno per riga.

L'output contiene gli elementi dell'array ordinato, **su una sola riga**.

Esercizio 2

QuickSort su stringhe

A partire dal codice dell'esercizio precedente, scrivere una funzione che, dato un array di stringhe e la sua lunghezza, lo ordini (lessicograficamente) utilizzando l'algoritmo **Quicksort**.

Scrivere un programma che utilizzi la funzione per ordinare un array di N stringhe lette da input e stampi in output gli elementi dell'array ordinato. Assumere che la lunghezza massima di una stringa sia 100 caratteri.

Si può utilizzare la funzione `strcmp` in `string.h` per confrontare lessicograficamente due stringhe. Utilizzare il comando `man strcmp` per maggiori informazioni.

La prima riga dell'input contiene la dimensione N dell'array. Le righe successive contengono gli elementi dell'array, una stringa per riga. Ogni stringa ha lunghezza massima di 100 caratteri.

L'output contiene gli elementi dell'array ordinato, **una stringa per riga**.

Esercizio 3

QuickSort strambo

Modificare il Quicksort del primo esercizio in maniera tale che ordini gli elementi pari nella parte inferiore dell'array e quelli dispari in quella superiore.

Scrivere un programma che utilizzi la funzione per ordinare come indicato un array di N interi letti da input.

La prima riga dell'input contiene la dimensione N (non limitata) dell'array. Le righe successive contengono gli elementi dell'array, uno per riga. L'output contiene la sequenza ordinata, **su una sola riga**.

Esercizio 4

3-way QuickSort

Partendo dall'implementazione del primo esercizio, implementare il Quick-sort su interi con *three-way partition*. L'algoritmo si differenzia dal Quicksort per la fase di partizionamento. In questo caso la funzione `distribuzione` divide l'array in tre intervalli (invece di due):

1. gli elementi minori del pivot;
2. gli elementi uguali al pivot;
3. gli elementi maggiori del pivot.

Scrivere un programma che utilizzi la funzione per ordinare un array di N interi letti da input.

La prima riga dell'input contiene la dimensione N (non limitata) dell'array. Le righe successive contengono gli elementi dell'array, uno per riga. L'output contiene gli elementi dell'array ordinato, **su una sola riga**.

Puzzle

La Scala

(da *Olimpiadi Italiane di Informatica, 2003*)

Un gradino è un rettangolo che giace sul piano cartesiano, i cui lati sono paralleli ai due assi. Una scala è una sequenza di gradini con le seguenti proprietà:

- i lati inferiori di tutti i gradini giacciono sull'asse X;
- il lato sinistro del primo gradino giace sull'asse Y;
- il lato sinistro di ogni gradino successivo al primo giace sul lato destro del gradino precedente;
- le altezze dei gradini sono strettamente decrescenti.

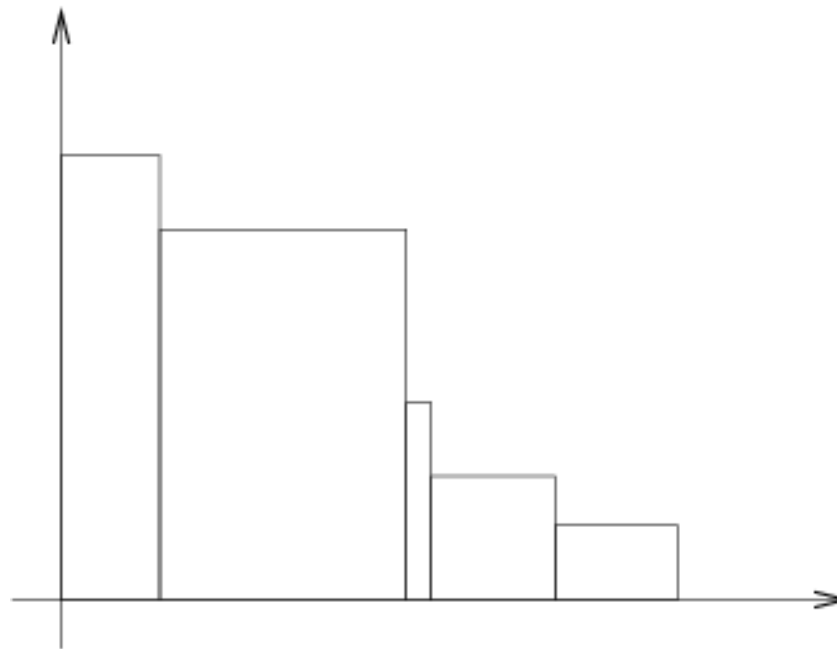


Figure 1: Un esempio di scala

Supponete di avere un insieme di punti sul piano cartesiano le cui coordinate sono numeri interi positivi. Il vostro obiettivo è di trovare una scala tale che tutti i punti dell'insieme giacciono nell'area sottesa alla scala (oppure, sul bordo della scala stessa). Fra tutte le scale possibili, volete

Puzzle

sceglierne una che minimizzi l'area sottesa.

Il programma legge i dati punti da input. Sulla prima riga è indicato un singolo numero intero N che è il numero di punti. Su ciascuna delle successive N righe è indicato un punto, espresso attraverso le sue coordinate x e y (due numeri interi separati da uno spazio). L'output sarà formato da una sola riga contenente l'area della scala di area minima.

La figura 2 mostra i punti dell'esempio e una scala di area minima che li contiene tutti. Notare che l'area sottesa misura $7 \times 13 + 3 \times 11 + 2 \times 7 + 8 \times 5 + 2 \times 1 = 91 + 33 + 14 + 40 + 2 = 180$.

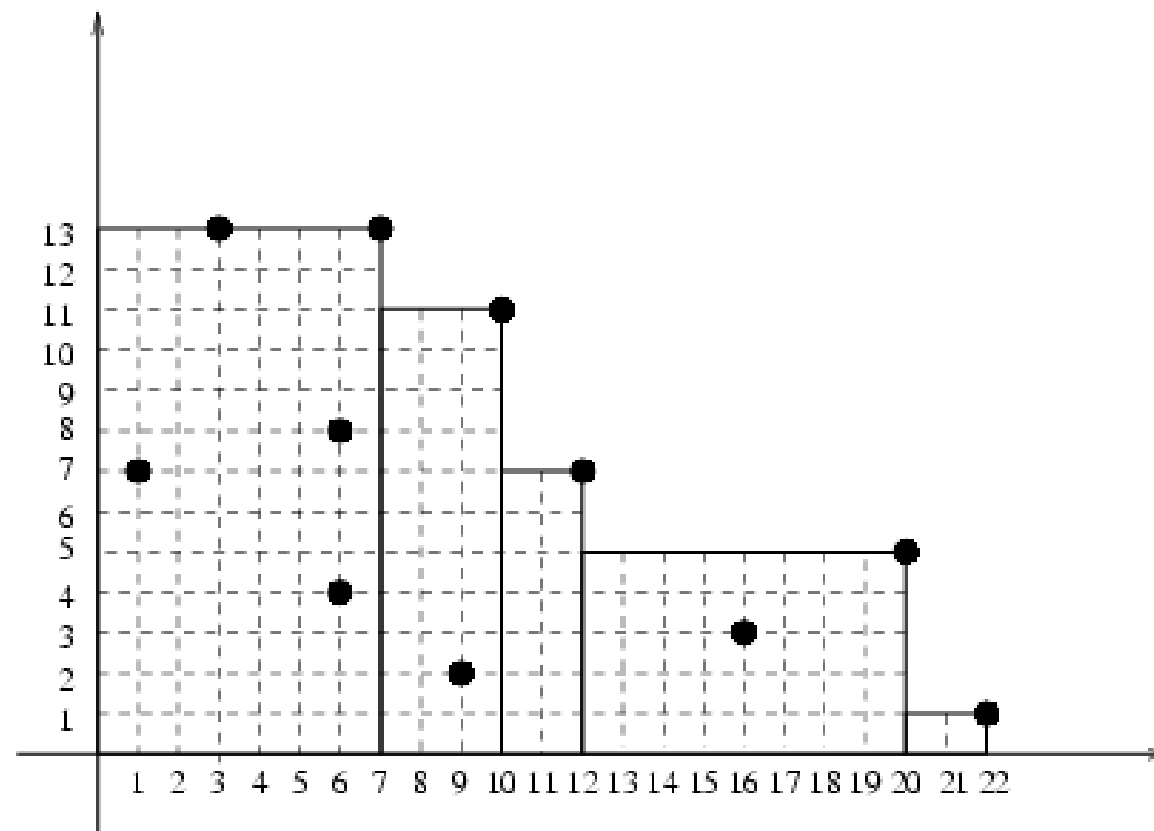


Figure 2: La scala di area minima per i punti dell'esempio.

Puzzle

Cavalli

(Folklore puzzle and Facebook's interview process.)

Si vuole stabilire quali sono i 3 cavalli più veloci in un insieme di 25. Possono essere organizzate gare tra 5 cavalli scelti in qualunque modo dall'insieme. Alla fine di ogni gara si conosce l'ordine di arrivo ma non il tempo impiegato dai cavalli. Qual è il numero minimo di gare necessario per identificare i 3 cavalli più veloci?