Lezione 3

Sottoarray di somma massima

Rossano Venturini

rossano.venturini@unipi.it

Pagina web del corso

http://didawiki.cli.di.unipi.it/doku.php/informatica/all-b/start

Esercizio 1

FindVal

Esercizio

Scrivere una funzione

```
int* FindVal(int a[], int len, int val)
```

che, dato un array *a* e la sua lunghezza *len*, cerchi il valore *val* all'interno di *a* e restituisca un puntatore alla cella che lo contiene, o la costante predefinita `NULL` se *val* non è contenuto in *a*.

Scrivere poi un programma che legga da input un array di 10 interi e un intero val e stampi `trovato` se l'intero val si trova nell'array, `non trovato` altrimenti.

L'input è formato da dieci righe contenenti gli elementi dell'array, seguite dall'intero val da cercare.
L'unica riga dell'output contiene la stringa

```
trovato
```

se l'intero val si trova nell'array,

```
non trovato
```

altrimenti.

Esercizio 1

```
int* FindVal(int a[], int len, int val) {

  int i = 0, trovato = 0;
  int *p = NULL;
  while ((i < len) && (!trovato)) {
    if (a[i] == val) {
      trovato = 1;
      p = &a[i];
    }
    i++;
  }
  return p;
}
```

Esercizio 2

Contatore

Esercizio

Scrivere una programma che data una sequenza di interi tenga traccia delle frequenze degli interi compresi tra 0 e 9 (estremi inclusi). La sequenza termina quando viene letto il valore -1. Il programma deve stampare in output le frequenze dei valori compresi tra 0 e 9.

Le frequenze saranno mantenute in un array di contatori di lunghezza 10 che sarà inizializzato a 0.

Implementare queste due funzioni:

- `void reset(int array[], int len)`: inizializza l'array dei contatori a 0;
- `void add(int array[], int len, int val)`: incrementa il contatore `array[val]` se `val` è tra 0 e `len`-1.

L'input è formato da una sequenza di interi terminata dall'intero -1.
L'output è costituito dalle frequenze (una per riga) degli interi tra 0 e 10 nella sequenza letta in input.

Esercizio 2

```
void reset(int array[], int len) {
  int i;
  for (i = 0; i < len; i++) {
    array[i] = 0;
  }
}
```

```
void add(int array[], int len, int val) {
  if ( (val < 0) || (val >= len) ) return;
  array[val] += 1;
}
```

Esercizio 4 Anagramma

Esercizio

Scrivere la funzione

`int anagramma(unsigned char *s1, unsigned char *s2)`

che restituisca 1 se le stringhe puntate da $s1$ e $s2$ sono una l'anagramma dell'altro e 0 altrimenti.
Esempio: `anagramma(''pizza'', ''pazzi'') == 1`

Strivere quindi un programma che legga da input due stringhe $s1$ e $s2$ e utilizzi questa funzione per stabilire se una è l'anagramma dell'altra. Nota: utilizzare il tipo `unsigned char *` per le stringhe.

Hint. Data una stringa S, costruire un array $aS[256]$ tale che $aS[i]$ memorizzi il numero di occorrenze del carattere i in S. Come sono gli array aS e aZ di due stringhe S e Z che sono una l'anagramma dell'altra?

L'input è formato da due stringhe $s1$ e $s2$.
L'output è 1 se $s1$ è l'anagramma di $s2$, 0 altrimenti.

Esempi

Input	Output
`aeiou` `uoaei`	`1`

Esercizio 4

```
int anagramma(unsigned char *x, unsigned char *y) {
  int i;
  int xc[256], yc[256];
  for (i = 0; i < 256; i++) { xc[i] = yc[i] = 0; }
  int lenx = strlen(x);
  int leny = strlen(y);
  if(lenx != leny) return 0;

  for (i = 0; i < lenx ; i++) {
    xc[x[i]] += 1;
    yc[y[i]] += 1;
  }

  for (i = 0; i < 256; i++) {
    if (xc[i] != yc[i]) return 0;
  }
  return 1;
}
```

Esercizio 5

My strcat 1

Esercizio

Implementare la funzione

```
char* my_strcat1(char *s1, char *s2)
```

che restituisce un puntatore alla *nuova* stringa ottenuta concatenando le stringhe puntate da s1 e s2.

Scrivere un programma che legga due stringhe da tastiera e stampi la stringa ottenuta concatenandole. Si può assumere che le stringhe in input contengano non più di 1000 caratteri.

Notare che il comportamento di `my_strcat1()` è diverso da quello della funzione `strcat()` presente nella libreria **string**.

L'input è formato da due stringhe di lunghezza non maggiore di 1000 caratteri.
L'unica riga dell'output contiene la stringa ottenuta concatenando nell'ordine le due stringhe inserite.

Esercizio 5

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

char* my_strcat(char *x, char *y) {
  int xlen = strlen(x);
  int ylen = strlen(y);

  char* str = (char*) malloc(sizeof(char)*(xlen + ylen + 1));
  if (str == NULL) exit(1);

  int pos = 0, i= 0;
  for (i = 0; i < xlen; i++) {
    str[pos++] = x[i];
  }
  for (i = 0; i < ylen; i++) {
    str[pos++] = y[i];
  }
  str[pos] = '\0';
  return str;
}
```

Esercizio 5

```
int main(void) {
  char *cat;
  char x[MAXLEN];
  char y[MAXLEN];
  scanf("%s", x);
  scanf("%s", y);
  cat = my_strcat(x, y);
  printf("%s\n",cat);
  return 0;
}
```

Esercizio 9

My strcpy

Esercizio

Scrivere una funzione

`char* my_strcpy(char* dest, char* src)`

che copi *src* in *dest* (incluso il terminatore `'\0'`) e restituisca un puntatore a *dest*. La funzione assume che in *dest* vi sia spazio sufficiente per contenere *src* (è compito del chiamante assicurarsi che ciò sia vero).
Si noti che il comportamento di `my_strcpy()` è uguale a quello della funzione `strcpy()` presente nella libreria **`string`**.

Scrivere poi un programma che: legga una stringa da tastiera (di lunghezza non maggiore di 1000 caratteri); allochi spazio sufficiente per una seconda stringa destinata a contenere la prima; copi la prima stringa nella seconda; stampi la seconda stringa.

L'input è formato da una sola riga contenente una stringa di lunghezza non maggiore di 1000 caratteri.
L'unica riga dell'output contiene la stampa della seconda stringa.

Esercizio 9

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

#define MAXLEN (1001)

char* my_strcpy(char* dest, char* src) {
  char *s = src, *d = dest;

  while ((*d = *s) != '\0') {
    d++;
    s++;
  }

  return dest;
}
```

Esercizio 9

```
int main(void) {
  char y[MAXLEN], *x;
  int len;

  scanf("%s", y);
  len = strlen(y);
  x = (char *) malloc(sizeof(char) * (len+1));
  x = my_strcpy(x, y);
  printf("%s\n", x);

  return 0;
}
```

Esercizio 10 Moltiplicazione di stringhe

Esercizio

Si scriva una funzione

`char* product(char *str, int k)`

che data una stringa *str* e un intero *k* restituisca una stringa ottenuta concatenando *k* volte la stringa *str*.

Si scriva un programma che legga in input:

- una stringa (assumendo che la stringa sia non più lunga di 1000 caratteri);
- un intero, che indica quante volte ripetere la stringa.

e infine stampi l'output di `product()`.

L'input è costituito, nell'ordine, da: una stringa di lunghezza non superiore a 1000 caratteri; un intero k che indica quante volte ripetere la stringa inserita.
L'unica riga dell'output è formata da una stringa contenente k concatenazioni della stringa data in input.

Esercizio 10

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

#define MAXSIZE (1001)

char* product(char *str, int k) {
  int i;
  int len = strlen(str);
  int plen = len*k+1;

  char *prod = malloc(plen*sizeof(char));

  for(i = 0; i < plen-1; i++) {
    prod[i]= str[ i%len ];
  }
  prod[plen-1] = '\0';

  return prod;
}
```

Esercizio 10

```
int main(void) {
  char str[MAXSIZE], *prod;
  int k;
  scanf("%s", str);
  scanf("%d", &k);
  prod = product(str, k);
  printf("%s\n", prod);
  return 0;
}
```

Sottoarray di Somma Massima

Sottoarray di Somma Massima

Problema: Dato un array a di n interi (positivi e negativi) individuare il sottoarray di somma massima.

Input: un array a di interi

Output: la somma del sottoarray di somma massima

Sottoarray di Somma Massima

Problema: Dato un array a di n interi (positivi e negativi) individuare il sottoarray di somma massima.

Input: un array a di interi

Output: la somma del sottoarray di somma massima

Esempio

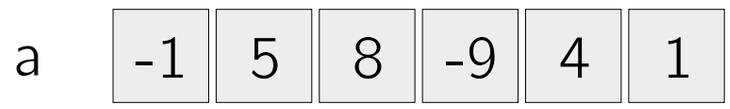
Sottoarray di Somma Massima

Problema: Dato un array a di n interi (positivi e negativi) individuare il sottoarray di somma massima.

Input: un array a di interi

Output: la somma del sottoarray di somma massima

Esempio



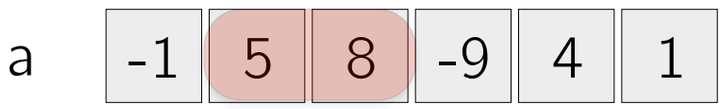
Sottoarray di Somma Massima

Problema: Dato un array a di n interi (positivi e negativi) individuare il sottoarray di somma massima.

Input: un array a di interi

Output: la somma del sottoarray di somma massima

Esempio



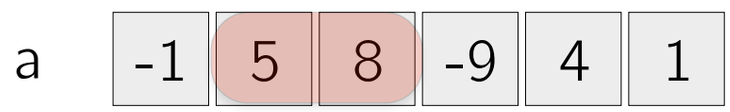
Sottoarray di Somma Massima

Problema: Dato un array a di n interi (positivi e negativi) individuare il sottoarray di somma massima.

Input: un array a di interi

Output: la somma del sottoarray di somma massima

Esempio



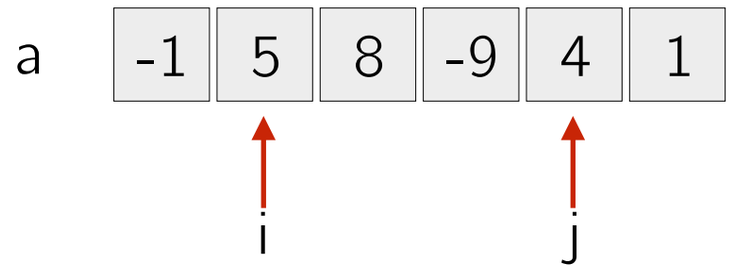
output: 13

Soluzione 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```

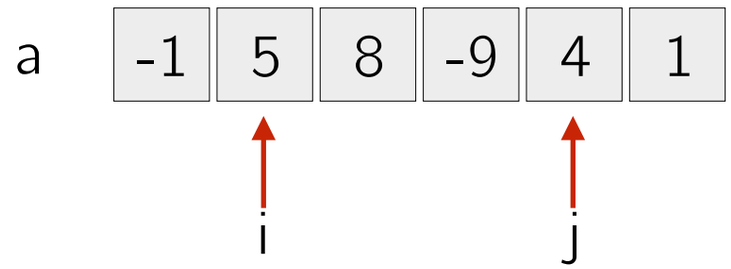
Soluzione 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```



Soluzione 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];        O(1)
        }
        if(somma > max) max=somma;
    }
}
```

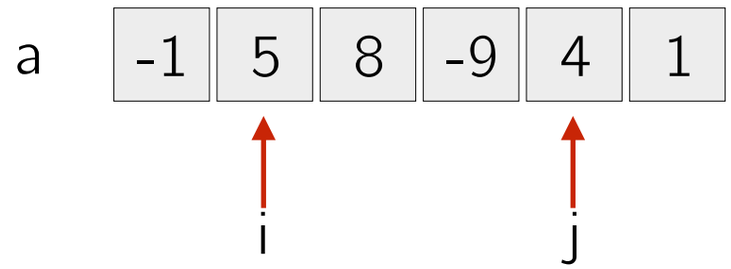


Soluzione 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```

O(1)

O(n)



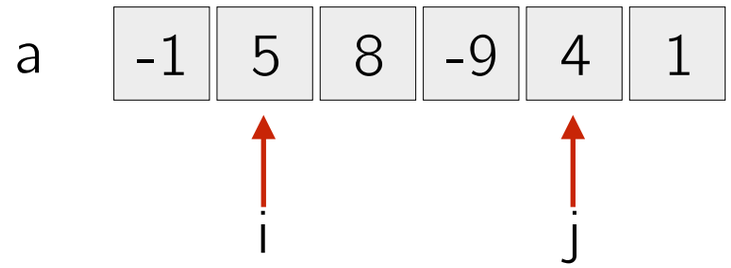
Soluzione 1

```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```

O(1)

O(n)

$O(n^2)$



Soluzione 1

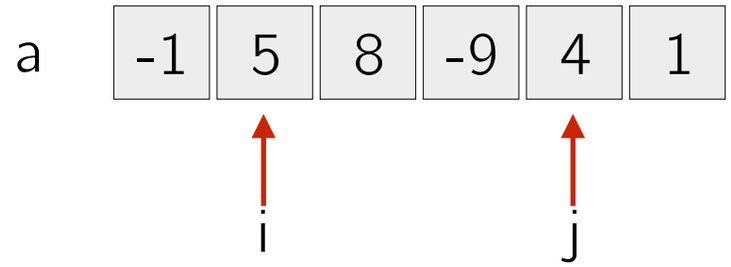
```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```

O(1)

O(n)

O(n²)

O(n³)



Soluzione 1

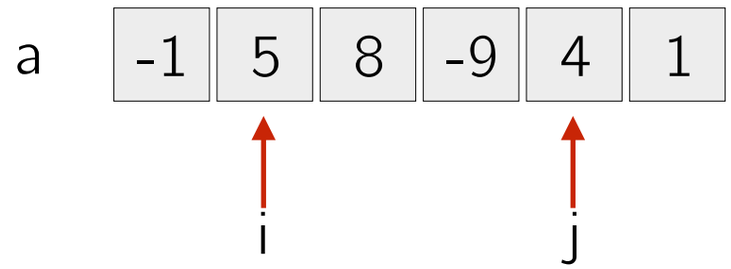
```
max = 0;

for(i=0; i<n; i++)
{
    for(j=i; j<n; j++)
    {
        somma=0;
        for(k=i; k<=j; k++)
        {
            somma+=a[k];
        }
        if(somma > max) max=somma;
    }
}
```

O(1)

O(n)

O(n²)

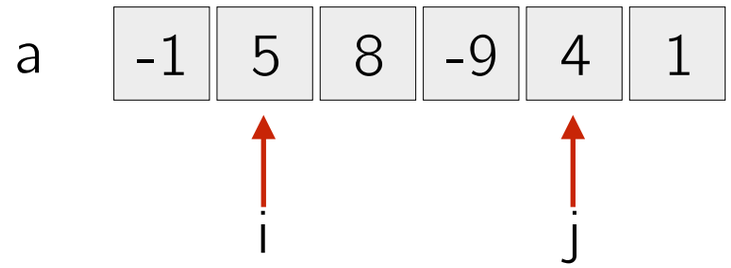
O(n³)



Tempo: $O(n^3)$:-(

Soluzione 2

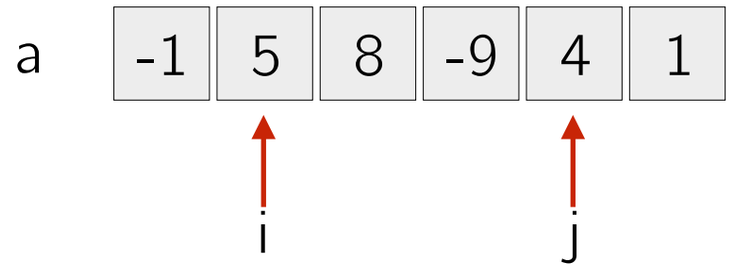
```
max = 0;

for(i=0; i<n; i++)
{
    somma=0;
    for(j=i; j<n; j++)
    {
        somma+=a[j];
        if(somma > max)
                max=somma;
    }
}
```



Soluzione 2

```
max = 0;

for(i=0; i<n; i++)
{
    somma=0;
    for(j=i; j<n; j++)
    {
        somma+=a[j];
        if(somma > max)
                max=somma;
    }
}
```

O(1)

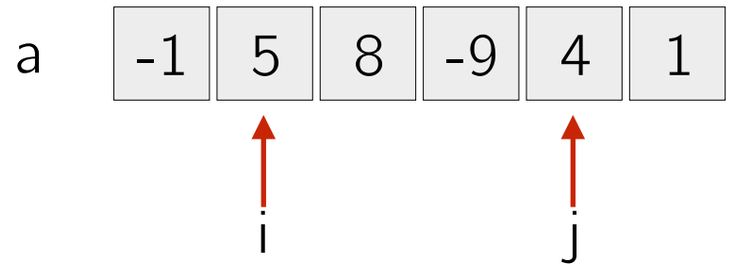


Soluzione 2

```
max = 0;

for(i=0; i<n; i++)
{
    somma=0;
    for(j=i; j<n; j++)
    {
        somma+=a[j];
        if(somma > max)
                max=somma;
    }
}
```

O(1)

O(n)



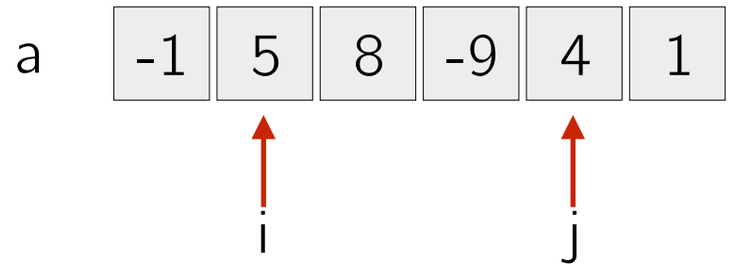
Soluzione 2

```
max = 0;

for(i=0; i<n; i++)
{
    somma=0;
    for(j=i; j<n; j++)
    {
        somma+=a[j];
        if(somma > max)
                max=somma;
    }
}
```

O(1)

O(n)

O(n^2)



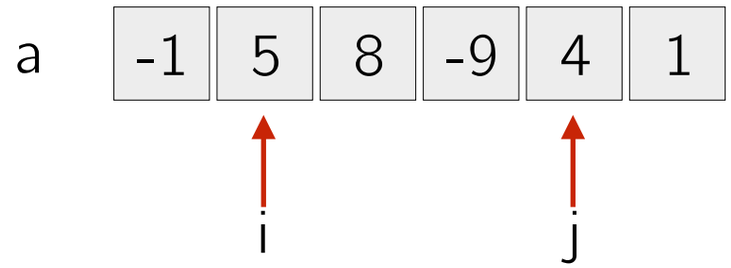
Soluzione 2

```
max = 0;

for(i=0; i<n; i++)
{
    somma=0;
    for(j=i; j<n; j++)
    {
        somma+=a[j];
        if(somma > max)
                max=somma;
    }
}
```

O(1)

O(n)

O(n^2)



Tempo: O(n^2) :-|

Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).

Come fare meglio?

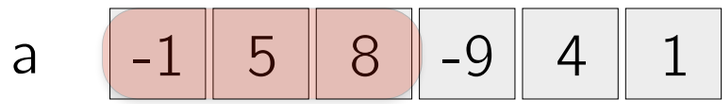
Possiamo sfruttare due proprietà del sottoarray di somma massima

1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).

Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

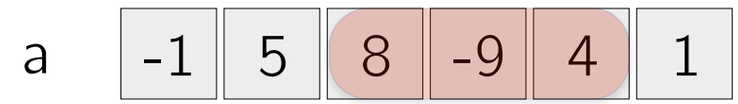
1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).



Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

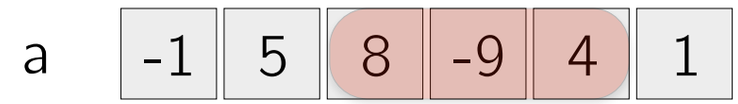
1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).



Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).

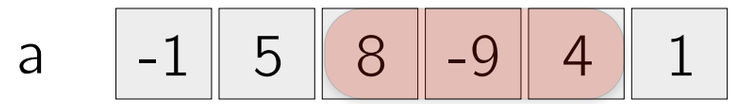


2) Il valore immediatamente precedente al primo valore del sottoarray ottimo è negativo, se così non fosse potremmo aggiungere tale valore ottenendo un sottoarray di somma maggiore (assurdo)

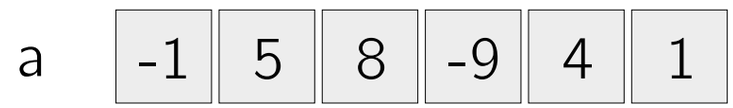
Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).



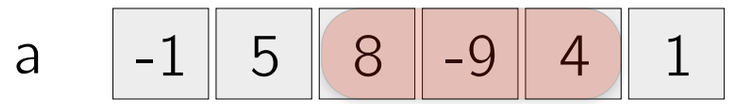
2) Il valore immediatamente precedente al primo valore del sottoarray ottimo è negativo, se così non fosse potremmo aggiungere tale valore ottenendo un sottoarray di somma maggiore (assurdo)



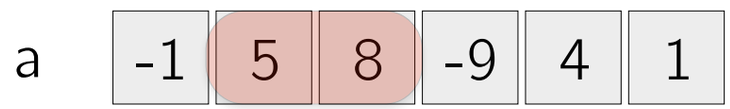
Come fare meglio?

Possiamo sfruttare due proprietà del sottoarray di somma massima

1) La somma dei valori in ogni prefisso del sottoarray ottimo è positiva, se così non fosse potremmo eliminare tale prefisso ottenendo un sottoarray di somma maggiore (assurdo).



2) Il valore immediatamente precedente al primo valore del sottoarray ottimo è negativo, se così non fosse potremmo aggiungere tale valore ottenendo un sottoarray di somma maggiore (assurdo)



Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

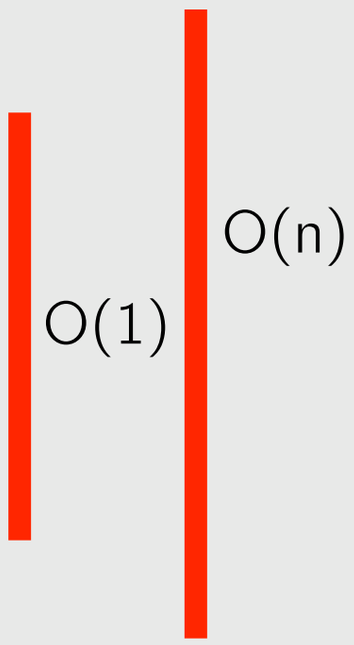
Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

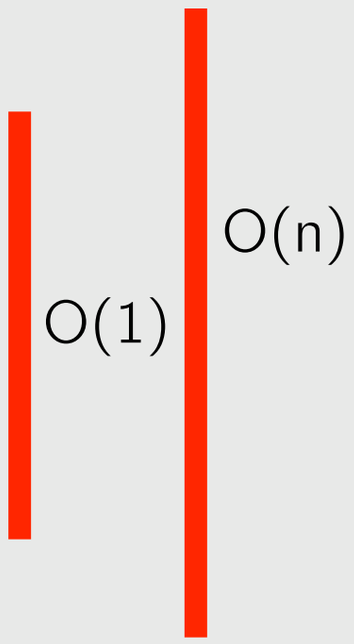
Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```



Soluzione 3

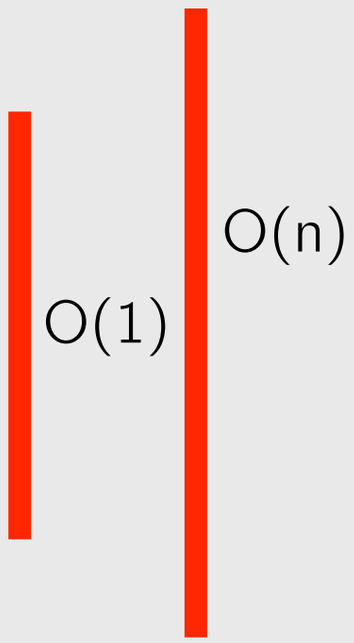
```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```



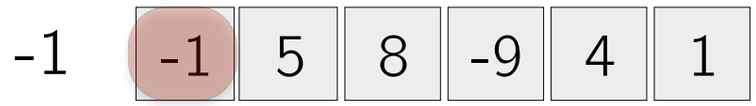
Tempo: O(n) :-)

Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```



somma



Tempo: O(n) :-)

Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

O(n)

somma

-1	-1	5	8	-9	4	1
5	-1	5	8	-9	4	1

Tempo: O(n) :-)

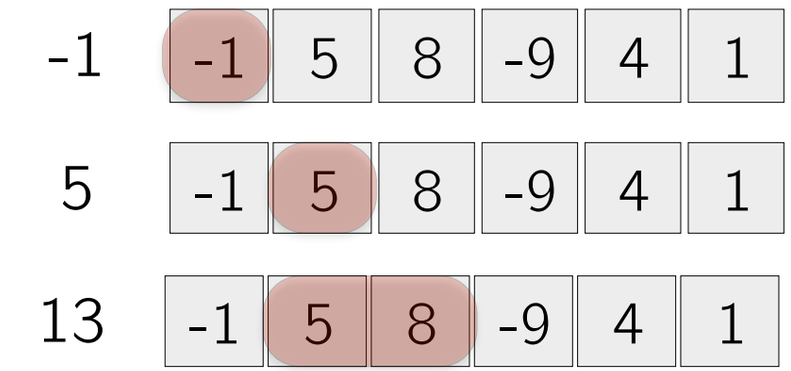
Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

O(n)

somma



Tempo: O(n) :-)

Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

O(n)

somma

-1	-1	5	8	-9	4	1
5	-1	5	8	-9	4	1
13	-1	5	8	-9	4	1
4	-1	5	8	-9	4	1

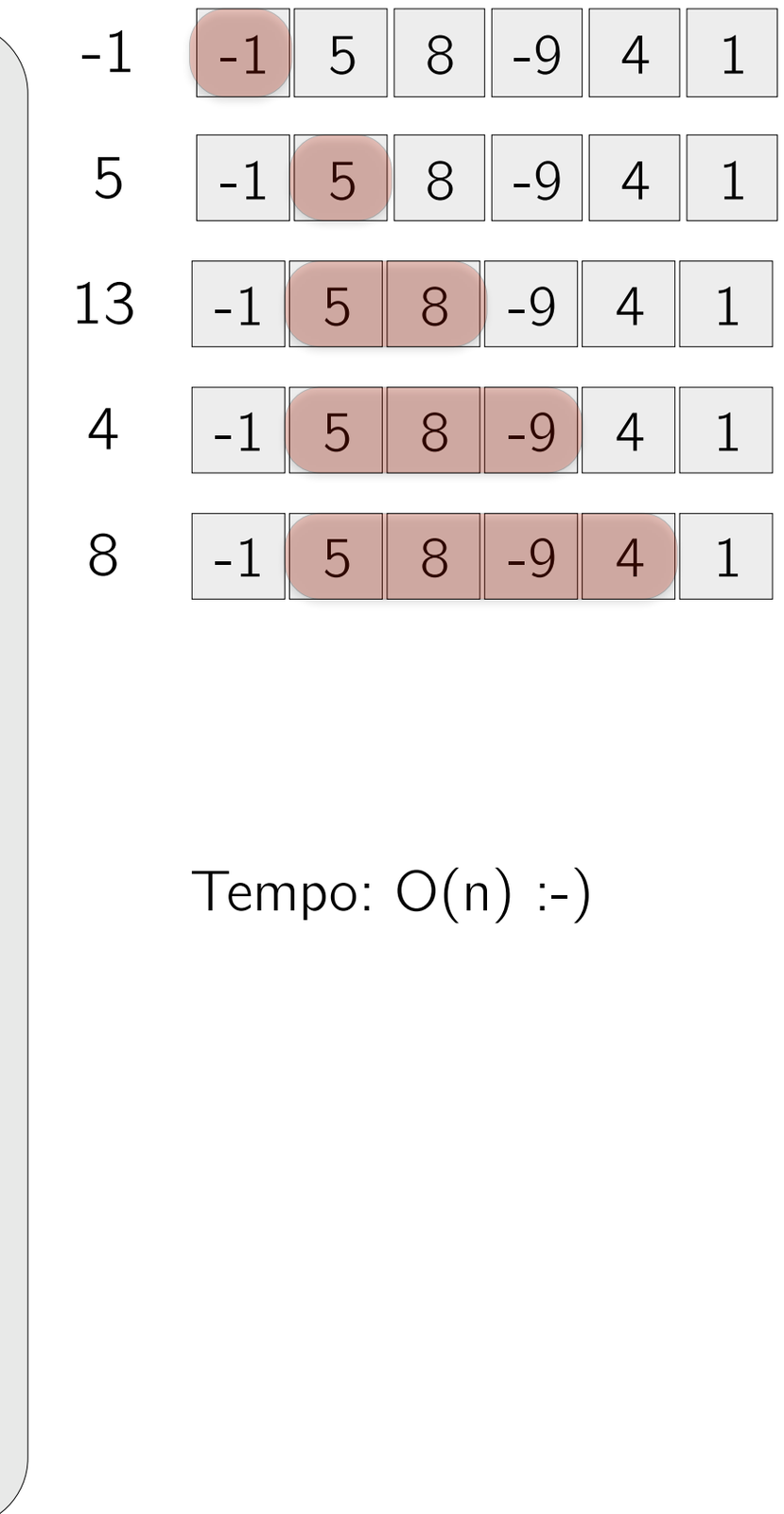
Tempo: O(n) :-)

Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

O(n)



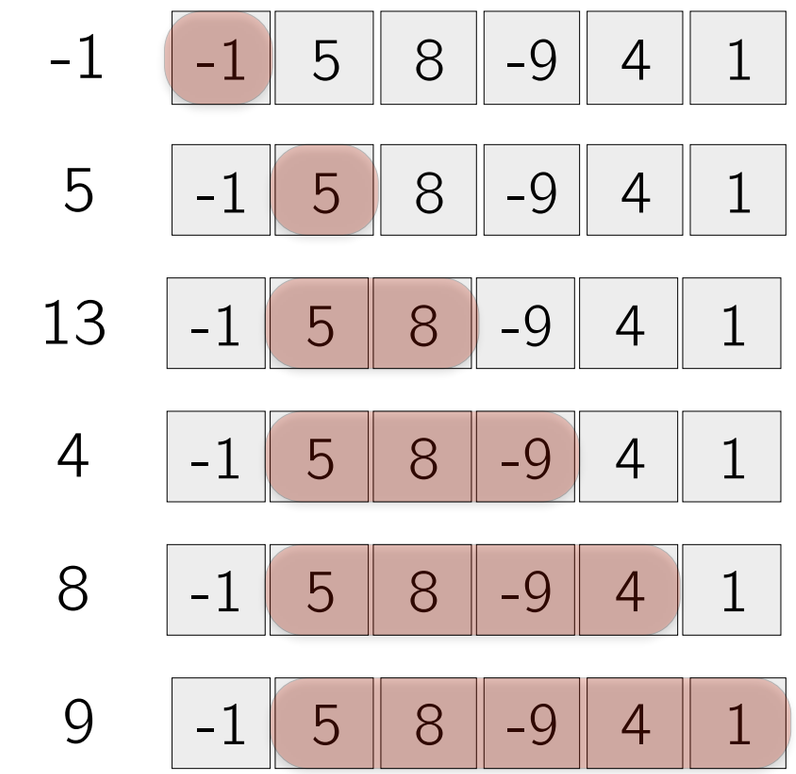
Tempo: O(n) :-)

Soluzione 3

```
max = 0; somma = 0;

for(i=0; i<n; i++)
{
   if(somma > 0) somma+=a[i];
   else somma=a[i];

   if(somma > max) max=somma;
}
```

O(1)

O(n)



Tempo: O(n) :-)

Esercizio 1

Intersezione tra array

Scrivere un programma che accetti in input due array di interi distinti e restituisca in output il numero di elementi che occorrono in entrambi gli array. Si assuma che la lunghezza di ogni array sia fornita prima dell'immissione degli elementi.

Una volta scritto il codice e superata la verifica sul sito, scaricare e decomprimere il file a questo indirizzo:

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

La directory contiene input diversi (file `.in`) insieme agli output attesi (file `.out`). Ad esempio: `100.in` contiene 2 array di lunghezza cento, ed è possibile usarlo come input al programma utilizzando la redirezione dell'input vista a lezione:

```
./esercizio.o < 100.in
```

Si provi a lanciare il programma su input diversi e per ogni input si controlli che l'output sia giusto confrontandolo col valore nel rispettivo file `.out`, che è possibile stampare sul terminale col comando `cat`, ad esempio:

```
cat 100.out
45
```

Infine, si provi a misurare quanto tempo impiega il programma su input diversi utilizzando il comando `time`, che restituisce in output il tempo impiegato dal programma, ad esempio:

```
time ./esercizio.o < 100.in
45
real 0.066 user 0.005 sys 0.003 pcpu 11.07
```

Come varia il tempo impiegato a seconda della dimensione dell'array?

Esercizio 2

Intersezione tra array ordinati

Scrivere un programma che accetti in input due array di interi distinti e restituisca in output il numero di elementi che occorrono in entrambi gli array. Si assuma che la lunghezza di ogni array sia fornita prima dell'immissione degli elementi.

Si assuma che questa volta gli array vengano inseriti *ordinati in maniera strettamente crescente*. Si può calcolare l'intersezione in maniera più efficiente?

Dopo aver superato i test sul sito, si ripetano gli esperimenti sul dataset utilizzato nel precedente esercizio (gli array vengono forniti in ordine crescente) e si confrontino i risultati ottenuti.

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

Esercizio 3

Sottoarray di somma massima

Dato un array A di n interi (positivi e negativi), scrivere un programma che identifichi il sottoarray di A i cui elementi hanno somma massima tra tutti gli altri sottoarray di A e ne stampi la somma.

Una volta scritto il codice e superata la verifica sul sito, scaricare i files di test diponibili all'indirizzo

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

e valutare l'efficienza della propria soluzione.

La soluzione implementata dovrebbe avere complessità lineare, in base all'algoritmo visto a lezione. Come utile esercizio facoltativo, implementare una soluzione di complessità non ottimale (ad esempio quadratica), misurare il tempo impiegato e notare la differenza tra le diverse soluzioni.

Esercizio 4

Fusione di array ordinati

Scrivere un programma che accetti in input due array *ordinati* di interi e restituisca in output l'unione ordinata dei due array. Si assuma che la lunghezza di ogni array sia fornita prima dell'immissione degli elementi.

Per semplicità si assuma che l'intersezione tra i due array sia vuota.

Google™

Y!™

twitter

LinkedIn

facebook®

YouTube

ebaY®

flickr™

amazon.com®

Microsoft®

How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.

How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.

How we hire

We're looking for our next Noogler - someone who's good for the role, good for Google and good at lots of things.

Things move quickly around here. At Internet speed. That means we have to be nimble, both in how we work and how we hire. We look for people who are great at lots of things, love big challenges and welcome big changes. We can't have too many specialists in just one particular area. We're looking for people who are good for Google—and not just for right now, but for the long term.

This is the core of how we hire. Our process is pretty basic; the path to getting hired usually involves a first conversation with a recruiter, a phone interview and an onsite interview at one of our offices. But there are a few things we've baked in along the way that make getting hired at Google a little different.

How we interview

We're looking for smart, team-oriented people who can get things done. When you interview at Google, you'll likely interview with four or five Googlers. They're looking for four things:

Leadership

We'll want to know how you've flexed different muscles in different situations in order to mobilize a team. This might be by asserting a leadership role at work or with an organization, or by helping a team succeed when you weren't officially appointed as the leader.

Role-Related Knowledge

We're looking for people who have a variety of strengths and passions, not just isolated skill sets. We also want to make sure that you have the experience and the background that will set you up for success in your role. For engineering candidates in particular, we'll be looking to check out your coding skills and technical areas of expertise.

How You Think

We're less concerned about grades and transcripts and more interested in how you think. We're likely to ask you some role-related questions that provide insight into how you solve problems. Show us how you would tackle the problem presented--don't get hung up on nailing the "right" answer.

Googleyness

We want to get a feel for what makes you, well, you. We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature.

How we decide

There are also a few other things we do to make sure we're always hiring the right candidate for the right role and for Google.

Microsoft interview

From Wikipedia, the free encyclopedia

The **Microsoft interview** is a job interview technique used by Microsoft to assess possible future Microsoft employees. It is significant because Microsoft's model was pioneering, and later picked up and developed by other companies including Amazon, Facebook, and Google[*citation needed*] [1]

Contents [hide]

Innovation [edit]

The Microsoft Interview was a pioneer in that it was about technical knowledge, problem solving and creativity as opposed to the goal and weaknesses interviews most companies used at the time. Initially based on Bill Gates' obsession with puzzles, many of the puzzles presented during interviews started off being Fermi problems, or sometimes logic problems, and have eventually transitioned over the years into questions relevant to programming[2]: *[P]uzzles test competitive edge as well as intelligence. Like business or football, a logic puzzle divides the world into winners and losers. You either get the answer, or you don't... Winning has to matter.* [3] Joel Spolsky phrased the problem as identifying people who are *smart and get things done* while separating them from *people who are smart but don't get things done* and *people who get things done but are not smart*[4][5]

This model is now used widely in the IT Industry, though it has been found that relationships that start under intense circumstances never last.

Microsoft interview

From Wikipedia, the free encyclopedia

The **Microsoft interview** is a job interview technique used by Microsoft to assess possible future Microsoft employees. It is significant because Microsoft's model was pioneering, and later picked up and developed by other companies including Amazon, Facebook, and Google[*citation needed*] [1]

Contents [hide]

Innovation [edit]

The Microsoft Interview was a pioneer in that it was about technical knowledge, problem solving and creativity as opposed to the goal and weaknesses interviews most companies used at the time. Initially based on Bill Gates' obsession with puzzles, many of the puzzles presented during interviews started off being Fermi problems, or sometimes logic problems, and have eventually transitioned over the years into questions relevant to programming[2]: *[P]uzzles test competitive edge as well as intelligence. Like business or football, a logic puzzle divides the world into winners and losers. You either get the answer, or you don't... Winning has to matter.* [3] Joel Spolsky phrased the problem as identifying people who are *smart and get things done* while separating them from *people who are smart but don't get things done* and *people who get things done but are not smart*[4][5]

This model is now used widely in the IT Industry, though it has been found that relationships that start under intense circumstances never last.


Google™

Please review the following topics:
Big-O notations also known as "the run time characteristic of an algorithm". You may want to refresh hash tables, heaps, binary trees, linked lists, depth-first search, recursion. For more information on Algorithms you can visit:
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index

Coding: You should know at least one programming language really well, and it should preferably be C++ or Java. C# is OK too, since it's pretty similar to Java. You will be expected to write some code in at least some of your interviews. You will be expected to know a fair amount of detail about your favorite programming language.

Sorting: Know how to sort. Don't do bubble-sort. You should know the details of at least one n*log(n) sorting algorithm, preferably two (say, quick sort and merge sort). Merge sort can be highly useful in situations where quick sort is impractical, so take a look at it.

Hashtables: Arguably the single most important data structure known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

Trees: Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarize yourself with binary trees, n-ary trees, and trie-trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal

Algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

Graphs: Graphs are really important at Google. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms: breadth-first search and depth-first search. Know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A*.

Other Data Structures: You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise. Find out whatNP-complete means.

Mathematics: Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems, and other Discrete Math 101 situations surrounds us. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n-choose-k problems and their ilk – the more the better.

Operating Systems: Know about processes, threads and concurrency issues. Know about locks and mutexes and semaphores and monitors and how they work. Knowabout deadlock and livelock and how to avoid them. Know what resources a processes needs, and a thread needs, and how context switching works, and how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs. For information on System

Puzzle

L'intero mancante

Viene dato un array di dimensione N contenente (non ordinati) tutti gli interi compresi tra 1 e $N + 1$ ad eccezione di uno di essi e si vuole stabilire l'elemento mancante.

Sono possibili almeno 4 soluzioni aventi le seguenti complessità:

Tempo	Spazio aggiuntivo	
$O(n^2)$	$O(1)$	`:-((`
$O(n \log n)$	$O(n)$	`:-(`
$O(n)$	$O(n)$	`:-)`
$O(n)$	$O(1)$	`:-D`

Input

```
8
3
4
9
2
7
1
8
6
```

Output

```
5
```