Algoritmi

Lezione 1

Organizzazione del Corso
Alcuni esempi

Organizzazione del Corso

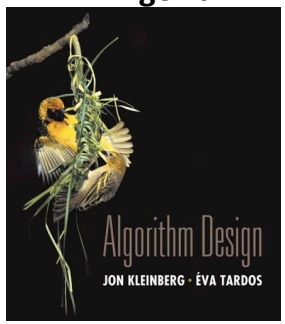
- Docente
  - Prof. Ferdinando Cicalese
    - Stanza 1.92, ferdinando.cicalese@univr.it
  - Tutor : Vincenzo Bonnici
    - vincenzo.bonnici@univr.it
- Lezioni
  - Giovedì, 2 ore, 11:30 - 13:30 aula D
  - Venerdì, 3 ore, 14:30 – 17:30 aula D
- Lezioni supplementari (fuori orario)
  - Lunedì 24 Novembre, 14:30 - 16:30 aula H
  - Lunedì 1 Dicembre, 14:30 - 16:30 aula H
  - Lunedì 15 Dicembre, 14:30 - 16:30 aula H
  - Lunedì 26 Gennaio, 14:30 - 16:30 aula H

Come contattare il docente

- alla fine di ogni lezione
- nell'orario di ricevimento *(durante il corso)*
  - Lunedì 14:30 -15:30
  - Martedì 14:30 - 15:30
  - A corso terminato, inviate mail e/o chiedete un appuntamento
- via e-mail
  - Indicate nel messaggio nome e matricola
  - Evitate domande che richiedano spiegazioni dettagliate (meglio un appuntamento o venire nell'orario di ricevimento)
  - Per domande relative alla soluzione degli esercizi, rivolgersi prima al tutor, (Vincenzo Bonnici, vincenzo.bonnici@univr.it)

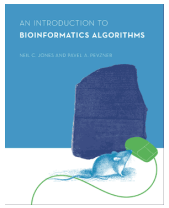
Libri

Algorithm Design



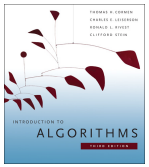
Autori: Jon Kleinberg – Éva Tardos
Casa editrice: Addison-Wesley
ISBN: 978-0321295354
2005

An Introduction to Bioinformatics Algorithms



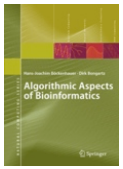
N. Jones, P. Pevzner
Casa editrice: MIT Press
ISBN: 978-0262101066
2004

Introduction to Algorithms



T.Cormen, C.Leiserson, R.Rivest, C.Stein
Casa editrice: MIT Press
ISBN: 978-0262033848
3rd Edition, 2009

Algorithmic Aspects of Bioinformatics

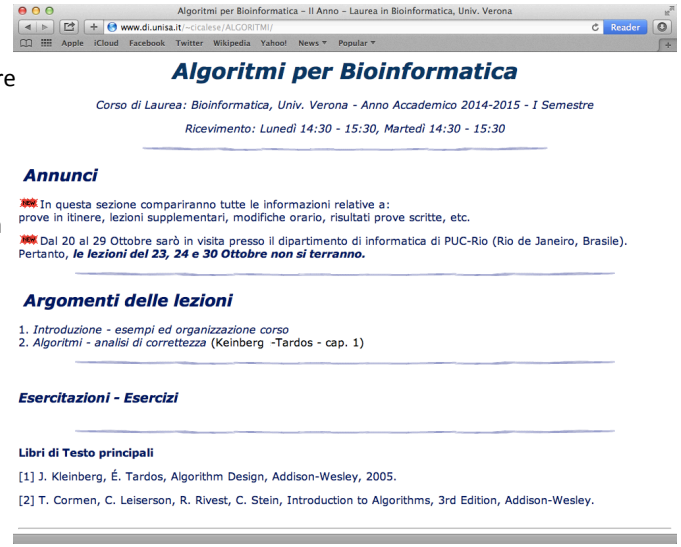


H.-J.Bockenhauer, D. Bongartz
Casa editrice: Springer
ISBN: 978-3540719137
2007

Sito web (in trasferimento)

- Materiale didattico
  - Slide
  - Note/Letture
  - Esercizi
- Informazioni
  - Programma
  - Libri
  - Calendario
  - Annunci
- Esami
  - Date
  - Risultati

http://profs.scienze.univr.it/~cicalese/ALGORITMI/



Esame

- Voto = 50% AlgoBio + 50% LabPrg2
- Algoritmi per Bioinformatica
  - 2 Prove in Itinere (+ Orale, opzionale)
  - Scritto (+ Orale, opzionale)
    - 😉 *opzionale* = a discrezione del docente 😉
- Voto AlgoBio (se fatto con le prove in itinere)
  - α P1 + (1-α)P2
    - P1: voto prova in itinere 1, P2: voto prova in itinere 2
    - α: dipende da quando faremo la prova in itinere 1

Domande?

Algoritmi per Bioinformatica

- Oggi parleremo di:
  - Cosa studieremo
  - Perché studiamo queste cose
  - Come le studieremo
  - Un esempio illustrativo

Cosa: Algoritmi per Bioinformatica

- Obiettivi formativi:
  - Fornire metodologie per il ***progetto*** e l'***analisi*** di algoritmi, con enfasi su problemi di interesse bioinformatico
    - Acquisire strumenti per la ***risoluzione di problemi***
  - Studieremo
    - ***Tecniche generali*** per lo sviluppo di algoritmi ***efficienti*** per risolvere problemi computazionali di interesse pratico
    - Strumenti per la ***valutazione*** degli algoritmi

Perché …studiare algoritmi?

"For me, great algorithms are the poetry of computation […] once unlocked, they cast a brilliant new light on some aspect of computing."

F. Sullivan, The Joy of Algorithms
IEEE Comp. Sc. Engineering, 2(1), 2000

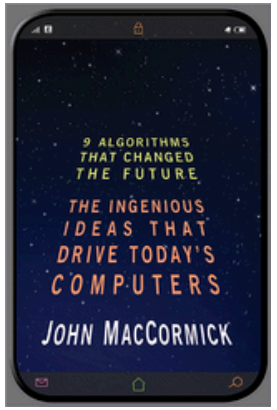
"Algorithmics is the core of computer science, and, in all fairness, can be said to be relevant to most of science, business, and technology. The very nature of algorithmics renders it particularly applicable to those disciplines that benefit from the use of computers, and these are fast becoming an overwhelming majority"

D. Harel,
Algorithmics: The Spirit of Computing

Perché studiare algoritmi?

- Tecniche algoritmiche permettono di risolvere problemi importantissimi in informatica e oltre
  - Trasmissione dati in Internet. Routing, file sharing
  - Ricerca sul WEB. Google, Bing, Yahoo
  - Business/Finance. Compravendita di azioni on-line, aste in rete (es. e-bay), e-commerce
  - Organizzazione di risorse. Scheduling di voli, assegnazione di frequenze in reti cellulari
  - Bioinformatica. Progetto Genoma, folding di proteine
  - Reti sociali. Feeds, condivisione e collegamenti, annunci personalizzati, viral campaigns

Perché studiare algoritmi?




TREND DE VIE

Cinema, app, fashion: il futuro è vintage

Orizzonti

Nuovi linguaggi, scienze, religioni, filosofie

Una Gaga-parrucchiera

Gli algoritmi del successo

Guida alle formule segrete che condizionano la nostra vita

di JOHN MACCORMICK

Algoritmo definizioni (n^{th} and $n+1^{st}$)

A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation

m-w.com

"An algorithm is a finite, definite, effective procedure, with some input and some output"

Donald Knuth – The Art of Computer Programming

Algoritmo - Etimologia

- **Algorithm** - This term, which means "rules for computing" in English, comes from al-Khowarizmi (Try saying it fast), an Arab mathematician living around A.D. 825 who completed the earliest known work in arithmetic using Arabic numerals. He was the first to establish rules for adding, subtracting, multiplying and dividing with the new Arabic numerals.

  Etymologically Speaking
  http://www.westegg.com/etymology/

- **Algorism =** process of doing arithmetic using Arabic numerals

Altri ***tentativi*** etimologici ***un pò più cattivelli***

Dal Greco: Algos [dolore] + arithmos [numero]
"farsi del male con i numeri" ?

Dal Latino: Algor [freddo]
"calcoli da far venire i brividi" ?

Cos'è un algoritmo… in soldoni

- Algoritmo: strumento/ricetta per risolvere un problema computazionale
- Un problema computazionale è definito da una relazione input/output
- Un algoritmo è quindi una procedura computazionale per ottenere la relazione input/output desiderata

Problemi computazionali

- Problema dell'Ordinamento
  - **Input**: sequenza di numeri $\langle a_1, \ldots, a_n \rangle$
  - **Output**: una permutazione $\langle a'_1, \ldots, a'_n \rangle$ dell'input tale che $a'_1 < \ldots < a'_n$
- Algoritmo di Ordinamento: procedura che prende in input $\langle a_1, \ldots, a_n \rangle$ e produce in output $\langle a'_1, \ldots, a'_n \rangle$

Problemi computazionali

- Problema della ricerca in un array
  - **Input**: un array A[1,…,n] e un elemento x
  - **Output**: un intero i, se A[i] = x per qualche indice i. Altrimenti il messaggio "non trovato"
- Algoritmo di Ricerca: procedura che prende in input A[1, …, n] ed x, e restituisce in output
  - "i" se A[i] = x per qualche i
  - "non trovato" se non esiste un i, tale che A[i]= x

Problemi reali ed astrazioni

- Mi serve un programma che trova le proteine in un database che presentano una determinata sequenza peptidica all'inizio
- Aha! questo è come un problema di ricerca in un array! La ricerca binaria cade a fagiolo!

Caveat! *Questa è una supersemplificazione di un problema molto più complesso*

Siamo Interessati ad Algoritmi efficienti

- Efficienza = uso di "poche" risorse
- Risorse = Tempo e Spazio richiesti dall'algoritmo per produrre l'output
- Q1: Come ***misuriamo*** le risorse utilizzate da un algoritmo? (analisi degli algoritmi)
- Q2: Come ***costruiamo*** algoritmi che usano "poche" risorse (tecniche di progetto)
- Un po' di pazienza… Lo vedremo…

Perché algoritmi efficienti

- Algoritmi efficienti portano a programmi efficienti
- Programmi efficienti si vendono meglio
  - Programmi efficienti fanno un uso migliore dell'hardware
- Programmatori che scrivono programmi efficienti sono più richiesti…
  - **e meglio pagati**

Cosa otterrete dal corso

- Metodi e conoscenze per formulare problemi astratti da problemi pratici
  - problemi concreti → problemi computazionali
- Metodi e conoscenze per la progettazione di algoritmi efficienti per la risoluzione di problemi computazionali
- Metodi e conoscenze per analizzare l'efficienza di un algoritmo
- Un "catalogo" di algoritmi efficienti, pronti per l'uso, per la risoluzione dei più comuni problemi computazionali
  - ***Ed in particolare problemi bioinformatici***

Domande?

Massimo utile in anni consecutivi

- All'ACME inc. vogliono conoscere il massimo utile ottenuto in anni consecutivi

Anno	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Profitto M€	8	10	15	-4	-7	1	-25	32	-5	10	-10	4

Massimo utile in anni consecutivi

- All'ACME inc. vogliono conoscere il massimo utile ottenuto in anni consecutivi

Anno	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Profitto M€	8	10	15	-4	-7	1	-25	32	-5	10	-10	4

Dal 2001 al 2003 il profitto è stato di 3+10+15 = **33**

Massimo utile in anni consecutivi

- All'ACME inc. vogliono conoscere il massimo utile ottenuto in anni consecutivi

Anno	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Profitto M€	8	10	15	-4	-7	1	-25	32	-5	10	-10	4

Dal 2001 al 2003 il profitto è stato di 3+10+15 = 33

Dal 2006 al 2010 il profitto è stato di 1-25+32-5+10 = **13**

Massimo utile in anni consecutivi

- All'ACME inc. vogliono conoscere il massimo utile ottenuto in anni consecutivi

Anno	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Profitto M€	8	10	15	-4	-7	1	-25	32	-5	10	-10	4

Dal 2001 al 2003 il profitto è stato di 3+10+15 = 33

Dal 2006 al 2010 il profitto è stato di 1-25+32-5+10 = 13

Dal 2008 al 2010 il profitto è stato di 32-5+10 = **37** **Questo è anche il massimo**

Massimo utile in anni consecutivi

- All'ACME inc. vogliono conoscere il massimo utile ottenuto in anni consecutivi

Anno	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
Profitto M€	8	10	15	-4	-7	1	-25	32	-5	10	-10	4

Dal 2001 al 2003 il profitto è stato di 3+10+15 = 33

Dal 2006 al 2010 il profitto è stato di 1-25+32-5+10 = 13

Dal 2008 al 2010 il profitto è stato di 32-5+10 = **37** **Questo è anche il massimo**

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Prima soluzione: ***Brute force***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Prima soluzione: ***Brute force***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

i *j*

Prima soluzione: ***Brute force***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

i *j*

```
maxsomma1(A, n)
 M = -∞
 For i = 0, …, n-1
     For  j = i, …, n-1
          sum = 0
          For  k = i, …, j
               somma = somma + A[k]
               M = max{somma, M}
return M
```


Prima soluzione: ***Brute force***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

i *j*

Q1: Risolve il problema?

Q2: E' efficiente?

Q2.1: Quanto efficiente?

```
maxsomma1(A, n)
 M = -∞
 For i = 0, …, n-1
     For  j = i, …, n-1
          sum = 0
          For  k = i, …, j
               somma = somma + A[k]
               M = max{somma, M}
return M
```


Prima soluzione: ***Brute force***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

i *j*

vengono eseguite un numero di volte pari a:

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \cong \tfrac{1}{6} n^3$$

```
maxsomma1(A, n)
 M = -∞
 For i = 0, …, n-1
     For  j = i, …, n-1
          sum = 0
          For  k = i, …, j
               somma = somma + A[k]
               M = max{somma, M}
return M
```

Seconda soluzione: ***Brute force migliorata***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

i *j-1* *j*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Seconda soluzione: ***Brute force migliorata***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j]
Ritorna il massimo

i *j-1* *j*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Per calcolare la somma da A[i] ad A[j]
usiamo la somma ***già calcolata*** da A[i] ad A[j-1]

Seconda soluzione: ***Brute force migliorata***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j] = **(A[i]+…+A[j-1])** + **A[j]**
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Per calcolare la somma da A[i] ad A[j]
usiamo la somma ***già calcolata*** da A[i] ad A[j-1]

Seconda soluzione: ***Brute force migliorata***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j] = **(A[i]+…+A[j-1])** + **A[j]**
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

```
maxsomma2(A, n)
 M = -∞
 For i = 0, …, n-1
     sum = 0
     For  j = i, …, n-1
          somma = somma + A[j]
          M = max{somma, M}
return M
```

Seconda soluzione: ***Brute force migliorata***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Per ogni *i* e *j*
calcola A[i]+A[i+1]+…+A[j] = **(A[i]+…+A[j-1])** + **A[j]**
Ritorna il massimo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

vengono eseguite un numero di volte pari a:

$$\sum_{i=0}^{n-1}(n-i+1) \cong \tfrac{1}{2}n^2$$

```
maxsomma2(A, n)
 M = -∞
 For i = 0, …, n-1
     sum = 0
     For  j = i, …, n-1
         somma = somma + A[j]
         M = max{somma, M}
return M
```


Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Il sottoarray che ottiene il massimo si trova:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

1. tutto nella prima metà (sinistra)

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1} \{A[i] + A[i+1] + \ldots + A[j]\}$

Il sottoarray che ottiene il massimo si trova:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

1. tutto nella prima metà (sinistra)

2. tutto nella seconda metà (destra)

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

Il sottoarray che ottiene il massimo si trova:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

1. tutto nella prima metà (sinistra)

2. tutto nella seconda metà (destra)

3. contiene A[7] e A[8] (in mezzo)

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Questi li posso trovare usando la stessa procedura sulla parte sinistra e destra, ricorsivamente

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Questi li posso trovare usando la stessa procedura sulla parte sinistra e destra, ricorsivamente

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
left = maxsomma3(A,i, m)
right = maxsomma3(A,m+1,j)
across = ??
          // max subarray che contiene A[m] e A[m+1]
return max{left, right, across}
```

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Come calcolo **across** ???

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = ??
          // max subarray che contiene A[m] e A[m+1]
  return max{left, right, across}
```


Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Come calcolo **across** ???

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Come calcolo **across** ???

across = A[7]+A[6]+…+A[ℓ]
+A[8]+A[9]+…+A[r]
per qualche
$0 \le \ell \le 7$ e $8 \le r \le n-1$

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Come calcolo **across** ???

across = $\max_{\ell}$ A[7]+A[6]+…+A[ℓ]
+ $\max_{r}$ A[8]+A[9]+…+A[r]

$0 \le \ell \le 7$ e $8 \le r \le n-1$

Il calcolo di **across** usa "lunghezza-di-A" operazioni

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

In generale
per A[i..j] con m = (i+j)/2

across = $\max_{\ell} A[m]+A[m-1]+\ldots+A[\ell]$
+ $\max_{r} A[m+1]+A[m+2]+\ldots+A[r]$

$i \le \ell \le m$ e $m+1 \le r \le j$

Come calcolo **across** ???

Il calcolo di **across** usa "lunghezza-di-A" operazioni (j-i+1)

Terza soluzione: ***Ricorsiva***

Maximum subarray problem

- **Input**: un array A[0,…,n-1]
- **Output**: $\max_{0 \le i \le j \le n-1}\{A[i]+A[i+1]+\ldots+A[j]\}$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

Il max sottoarray è quindi il massimo tra:
il max sottoarray a sinistra
il max sottoarray a destra
il max sottoarray contenente A[7] e A[8]

Questi li posso trovare usando la stessa procedura sulla parte sinistra e destra, ricorsivamente

calcolo **across** in "*lunghezza-di-A*" operazioni

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+…+A[m]) + max_r(A[m+1]+…+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```


Livelli di Ricorsione e costo per livello

op-across

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

16

0	1	2	3	4	5	6	7
8	10	15	-4	-7	1	-25	32

8	9	10	11	12	13	14	15
10	-5	-10	4	-3	2	-1	2

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+…+A[m]) + max_r(A[m+1]+…+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```


Livelli di Ricorsione e costo per livello

op-across

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

16

8

0	1	2	3	4	5	6	7
8	10	15	-4	-7	1	-25	32

8

8	9	10	11	12	13	14	15
10	-5	-10	4	-3	2	-1	2

16

0	1	2	3
8	10	15	-4

4	5	6	7
-7	1	-25	32

8	9	10	11
10	-5	-10	4

12	13	14	15
-3	2	-1	2

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+…+A[m]) + max_r(A[m+1]+…+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```

Livelli di Ricorsione e costo per livello

op-across

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	10	15	-4	-7	1	-25	32	-5	10	-10	4	-3	2	-1	2

16

8

0	1	2	3	4	5	6	7
8	10	15	-4	-7	1	-25	32

8

8	9	10	11	12	13	14	15
10	-5	-10	4	-3	2	-1	2

16

4

0	1	2	3
8	10	15	-4

4

4	5	6	7
-7	1	-25	32

4

8	9	10	11
10	-5	-10	4

4

12	13	14	15
-3	2	-1	2

16

2

0	1
8	10

2

2	3
15	-4

...

1 1 1

0	1	2	3
8	10	15	-4

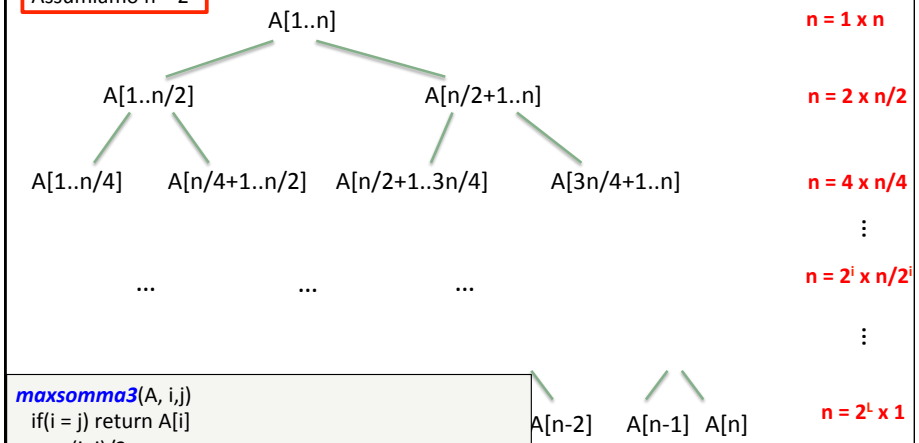
...

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+...+A[m]) + max_r(A[m+1]+...+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```


Livelli di Ricorsione e costo per livello

Assumiamo $n = 2^L$

op-across



n = 1 x n

n = 2 x n/2

n = 4 x n/4

⋮

$n = 2^i \times n/2^i$

⋮

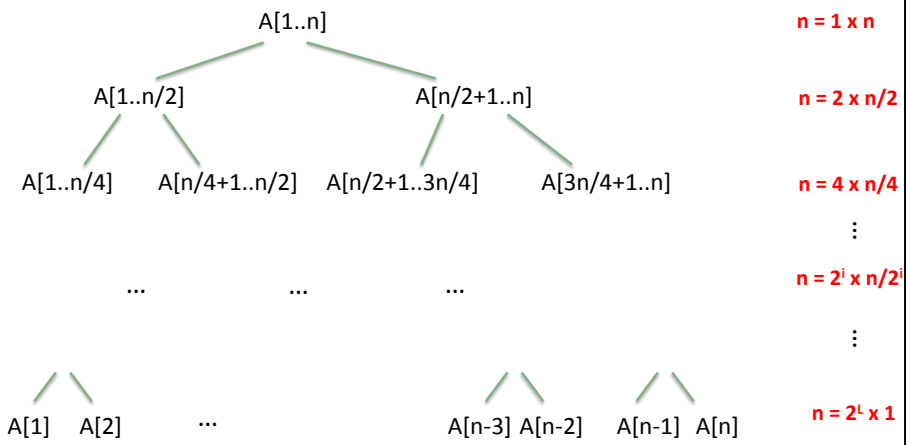
$n = 2^L \times 1$

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+...+A[m]) + max_r(A[m+1]+...+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```

L = numero di livelli

Livelli di Ricorsione e costo per livello

op-across



n = 1 x n

n = 2 x n/2

n = 4 x n/4

⋮

$n = 2^i \times n/2^i$

⋮

$n = 2^L \times 1$

Costo totale ≅ n x L = n $\log_2$ n

L = numero di livelli dell'albero
= # volte che dobbiamo dividere n per 2 per ottenere un valore ≤ 1

Maximum subarray problem: 3 soluzioni

```
maxsomma1(A, n)
  M = -∞
  For i = 0, ..., n-1
      For  j = i, ..., n-1
           sum = 0
           For  k = i, ..., j
                somma = somma + A[k]
                M = max{somma, M}
  return M
```

Numero stimato di operazioni ≅ $\frac{1}{6}n^3$

```
maxsomma2(A, n)
  M = -∞
  For i = 0, ..., n-1
      sum = 0
      For  j = i, ..., n-1
           somma = somma + A[j]
           M = max{somma, M}
  return M
```

Numero stimato di operazioni ≅ $\frac{1}{2}n^2$

Running time su questo computer

n	maxsomma 1	maxsomma 2	maxsomma 3
#op	$n^3/6$	$n^2/2$	$n \log_2 n$
10	0.000004	0.000003	0.000004
10^3	0.539s	0.002s	0.0001 s
10^4	476.3 s	0.183s	0.0014 s
10^5	126 ore	19.54s	0.009 s
10^6	15 anni	32 min	0.125 s
10^7	---	53 ore	1.169 s

```
maxsomma3(A, i,j)
  if(i = j) return A[i]
  m = (i+j)/2
  left = maxsomma3(A,i, m)
  right = maxsomma3(A,m+1,j)
  across = max_ℓ(A[ℓ]+...+A[m]) + max_r(A[m+1]+...+A[r])
           // j-i+1 operazioni:  i ≤ ℓ ≤ m  e  m+1 ≤ r ≤ j
  return max{left, right, across}
```

Numero stimato di operazioni ≅ $n \log_2 n$

Conclusioni

- Abbiamo visto che
  - Lavorare sull'algoritmo paga… e molto
  - Tecniche algoritmiche possono migliorare di molto il comportamento dei programmi
    - *Brute force* vs. *divide et impera*
  - Per stimare il comportamento di un programma possiamo contare il numero di operazioni rispetto alla taglia dell'input

Domande / Esercizi /Quesiti

- L'algoritmo ricorsivo è il meglio possibile? (~ n log n)?
  - No! Esiste un algoritmo più efficiente (~ n)
    - Riuscite a trovarlo da soli?
  - Qual è il minimo numero di operazioni che ogni algoritmo deve necessariamente (almeno) fare?
    - è giusto dire che tutti gli n^2 possibili sottoarray devono essere calcolati?
  - Se ACME fosse interessata non al massimo profitto, ma al massimo profitto medio, come procedereste ?

Domande / Esercizi /Quesiti

- Provare le relazioni matematiche usate per stimare il numero di operazioni degli algoritmi brute force

$$\sum_{i=0}^{n-1}(n-i+1) \cong \tfrac{1}{2}n^2 \qquad \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) \cong \tfrac{1}{6}n^3$$

- Cosa cambia se nell'algoritmo ricorsivo dividiamo l'array non a metà ma in due parti, una di taglia n/4 e l'altra 3n/4?

Arrivederci…

… alla prossima lezione