Laboratorio di Algoritmi e Strutture Dati

Primo esercizio, seconda parte: *MergeSort*, *HybridSort* (punti: 2)

"THE PROBABILITY OF SOMEONE WATCHING YOU IS PROPORTIONAL TO THE STUPIDITY OF YOUR ACTIONS."

Verso l'efficienza

Quando un'implementazione è **efficiente**? Assumendo che il codice scritto sia corretto e perfettamente funzionante, gli elementi che contribuiscono all'efficienza di una soluzione sono almeno due

1. un'**idea algoritmica** corretta, completa, terminante, e asintoticamente efficiente (quando possibile, ottima), e un'**implementazione** che sfrutti le caratteristiche del linguaggio, delle strutture dati, e
2. ottimizzazioni locali che non influenzano il comportamento asintotico (almeno non in generale).

Gli esercizi di laboratorio vanno nella direzione del secondo punto.

Migliorare *MergeSort*

In questo esercizio vogliamo costruire un nuovo algoritmo di ordinamento basato sui confronti. L'obbiettivo è quello di sfruttare i punti forti di due algoritmi che abbiamo visto: *InsertionSort* e *MergeSort*. Il risultato sará un algoritmo misto, per così dire, che si comporti meglio, dal punto di vista sperimentale, di entrambi.

Migliorare *MergeSort*: idea

Da un punto vista analitico, il tempo di esecuzione di *MergeSort* è $\Theta(n \cdot log(n))$ (in tutti i casi), e quello di *InsertionSort* è $\Theta(n^2)$ (nel caso peggiore). Prendiamo unicamente il limite superiore nel caso peggiore.

Abbiamo, per *MergeSort*:

$$T(n) \leq c_1 \cdot n \cdot log(n) \Rightarrow O(n \cdot log(n))$$

e per *InsertionSort*:

$$T(n) \leq c_2 \cdot n^2 \Rightarrow O(n^2)$$

Migliorare *MergeSort*: idea

Possiamo mostrare che, per n **sufficientemente piccolo**, succede che:

$$c_2 \cdot n^2 < c_1 \cdot n \cdot log(n).$$

Questa disequazione è vera per tutti i valori di n **piú piccoli** di un certo valore k che bisogna trovare. Dimostrarlo formalmente richiede, primo, trovare le costanti c_1, c_2 (che dipendono dall'implementazione e dalla macchina), e, poi, risolvere un'equazione esponenziale, che implica, a sua volta, usare la funzione W di Lambert.

Noi, invece, cercheremo di mostrarlo in maniera sperimentale.

Migliorare *MergeSort*

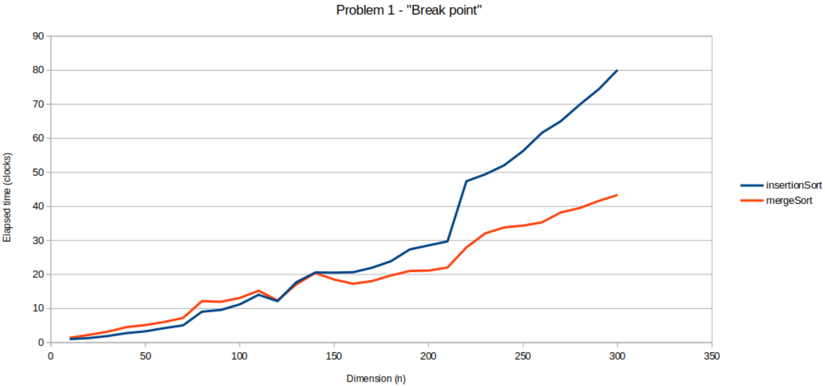
Vogliamo dunque realizzare un esperimento implementando sia *InsertionSort* (cosa che abbiamo giá fatto) che *MergeSort*, misurando il tempo di esecuzione sperimentale per input di lunghezza crescente, con multiple ripetizioni per la stessa lunghezza, su entrambi gli algoritmi.

Vogliamo dedurre il valore di k (punto massimo di incrocio tra le curve) per la propria macchina e implementazione. Il risultato richesto prevede:

- Una rappresentazione grafica delle curve di tempo;
- L'identificazione del valore massimo k oltre il quale *MergeSort* è sempre piú efficiente di *InsertionSort*.

Migliorare *MergeSort*

Un esempio di possibile risultato è:



Migliorare *MergeSort*

Come usiamo questa informazione per costruire una versione piú efficiente (sperimentalmente) di *MergeSort*?

Poichè l'idea dell'algoritmo è quella di ordinare, ricorsivamente, array sempre piú piccoli (i pezzi dell'array originale), ad un certo punto, durante la ricorsione, si arriverá ad avere un array di dimensione inferiore a k. In quel momento non è piú conveniente richiamare *MergeSort*, ma conviene chiamare *InsertionSort*.

proc ***HybridSort*** (A, p, r)
- **if** $(r - p + 1 > k)$
  - **then**
    - $q = [(p + r)/2]$
    - $HybridSort(A, p, q)$
    - $HybridSort(A, q + 1, r)$
    - $Merge(A, p, q, r)$
  - **else** $AdaptedInsertionSort(A, p, r)$

Migliorare *MergeSort*

La funzione *AdaptedInsertionSort* si comporta come *InsertionSort*, ma in maniera da poter essere eseguita su un array A di dimensione arbitraria del quale ordinamo unicamente le posizioni dalla p alla r comprese. Se non si adatta in questo modo, *InsertionSort* ordinerá tutto l'array A molte volte risultando in una versione di *MergeSort* molto piú inefficiente di quella originale. Da un punto di vista asintotico, la soluzione proposta **non** è piú efficiente di *MergeSort*:

$$T(n) = \begin{cases} 2 \cdot T(\frac{n}{2}) + \Theta(n) & \text{se } n > k \\ k^2 & \text{altrimenti} \end{cases}$$

Si puó utilizzare il metodo dello sviluppo per convincersi che è ancora vero che:

$$T(n) = \Theta(n \cdot log(n)).$$

Ma, come vedremo, questa variante comporta un miglioramento sperimentale.

Migliorare *MergeSort*

Quindi vogliamo realizzare un esperimento dove alle implementazioni precedenti aggiungiamo *HybridSort*, quest'ultimo implementato utilizzando la costante k precedentemente trovata, misurando il tempo di esecuzione sperimentale per input di lunghezza crescente, con multiple ripetizioni per la stessa lunghezza, su entrambi gli algoritmi.

Il risultato richesto prevede una rappresentazione grafica delle curve di tempo, dove si vede che *HybridSort* migliora sempre il tempo di esecuzione di *MergeSort*, e yna dimostrazione sperimentale di correttezza attraverso test randomizzati e funzioni antagoniste.

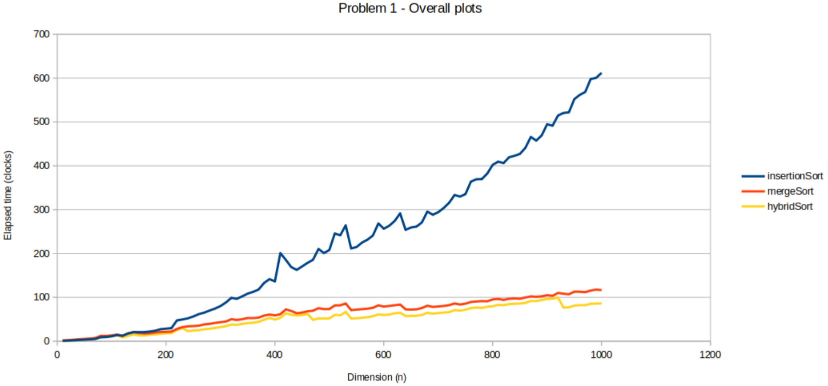
Migliorare *MergeSort*

```
proc SingleExperiment (length, max_instances, alg)
  t_tot = 0
  for (instance = 1 to max_instances)
    A = GenerateRandom(length)
    if (alg = IS)
      then
        t_start = clock()
        InsertionSort(A)
        t_end = clock()
    if (alg = MS)
      then
        t_start = clock()
        MergeSort(A)
        t_end = clock()
    if (alg = HS)
      then
        t_start = clock()
        HybridSort(A)
        t_end = clock()
    t_elapsed = t_end − t_start
    t_tot = t_tot + t_elapsed
  t_final = t_tot/max_instances
  return t_final
```

Migliorare *MergeSort*

```
proc Experiment (min_length, max_length, seed)
  max_instances = 5
  step = 10
  for (length = min_length to max_length step step)
    Reset(seed)
    timeIS = SingleExperiment(A, length, max_instances, IS)
    Reset(seed)
    timeMS = SingleExperiment(A, length, max_instances, MS)
    Reset(seed)
    timeHS = SingleExperiment(A, length, max_instances, HS)
    print(timeIS, timeMS, timeMS)
    seed = seed + 1
```

Da un punto di vista sperimentale ci aspettiamo un risultato del genere (in questo esempio, $k = 120$):



Conclusione

In questo esercizio abbiamo visto come possiamo migliorare un algoritmo ben noto, *MergeSort*, usando un'**euristica** sperimentale. Questa stessa idea può essere usata in molti altri ambiti.