Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica
Curriculum SR (Sistemi e Reti)

Laboratorio di Algoritmi a.a. 2009-10

Compito 6

Parte 1: fusione

Si assuma che ogni insieme (finito) di interi sia rappresentato da un array ordinato senza elementi ripetuti.

In un file merge.c si scrivano le procedure:

void unione(int a[], int b[], int c[], int m, int n, int *p)
mette in c l'unione insiemistica degli insiemi a e b, e in *p il numero dei suoi elementi;

void intersezione(int a[], int b[], int c[], int m, int n, int *p)
mette in c l'intersezione insiemistica degli insiemi a e b, e in *p il numero dei suoi elementi;

void diff(int a[], int b[], int c[], int m, int n, int *p)
mette in c la differenza insiemistica fra a e b, e in *p il numero dei suoi elementi (la differenza fra a e b è l'insieme degli elementi di a che non appartengono a b);

un main che provi le tre procedure.

Parte 2: sorting

In un file sorting-optimal.c si scrivano le procedure:

void mergesort(int a[], int n)
ordina l'array a di lunghezza n implementando l'algoritmo di *mergesort*;

void mergesort2(int a[], int n)
ordina l'array a di lunghezza n implementando l'algoritmo di *mergesort*, con l'ottimizzazione illustrata a lezione per cui non si fa la fusione fra due parti di array se l'ultimo elemento della prima parte è minore o uguale al primo della seconda parte.

Si scrivano come commento nel file, oppure in un file pdf separato, le equazioni di ricorrenza per il caso migliore di *mergesort2*, e le si risolva trovando così la complessità temporale del caso migliore di *mergesort2*.

void quicksort1(int a[], int n)
ordina l'array a di lunghezza n implementando l'algoritmo di *quicksort* nella versione in cui:
- come pivot si sceglie sempre l'ultimo elemento della parte considerata;
- la partizione avviene tenendo la parte da esaminare al fondo (come nella prima versione illustrata a lezione);

void quicksort2(int a[], int n)
ordina l'array a di lunghezza n implementando l'algoritmo di *quicksort* nella versione in cui:
- come pivot si sceglie sempre l'ultimo elemento della parte considerata;
- la partizione avviene tenendo al fondo i maggiori o uguali (come nella seconda versione illustrata a lezione).

void quicksort3(int a[], int n)
ordina l'array a di lunghezza n implementando l'algoritmo di *quicksort* nella versione in cui:
- come pivot si sceglie un elemento a caso;
- si effettua la tripartizione (< del pivot, = al pivot, > del pivot) e ovviamente si fanno le chiamate ricorsive solo sulle parti dei minori e dei maggiori.

Si scriva poi un main che confronti le varie procedure su array di lunghezze via via crescenti, sia non ordinati che ordinati, sia con pochi elementi ripetuti che con molti elementi ripetuti, misurando i tempi di esecuzione.
Si faccia poi, in un file separato, una tabella dei risultati ottenuti.