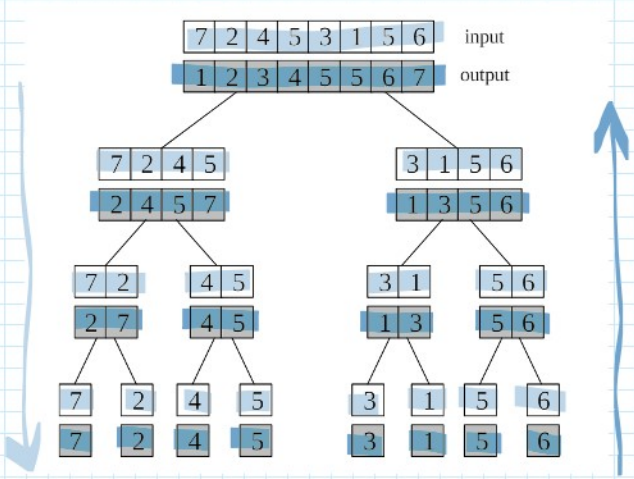
complessità merge sort

giovedì 20 giugno 2024 15:53

COMPLESSITÀ MERGE SORT

• idea della procedura merge sort

se l'array contiene solo un elemento, return
else {
trova la metà dell'array
riordina la prima metà dell'array (chiamando mergesort ms)
riordina la seconda metà dell'array (chiamando mergesort ms)
fondi le due metà così ordinate in un unico array
}




```
void mergeSort(vector& v) {
    ms(v, 0, v.size()-1);
}

void ms(vector& v, unsigned int inizio, unsigned int fine) {
    if (inizio < fine) {
        unsigned int centro = (inizio+fine)/2;
        ms(v, inizio, centro);
        ms(v, centro+1, fine);
        fondi(v, inizio, centro, fine);
    }
}
```

```
void fondi(vector& v, unsigned int inizio, unsigned int centro,
unsigned int fine) {
    vector vsinistra, vdestra;
5
    for (unsigned int i = inizio; i <= centro; ++i) {
        vsinistra.push_back(v[i]);
    }
    for (unsigned int i = centro+1 ; i <= fine; ++i) {
        vdestra.push_back(v[i]);
    }

    unsigned int indicesinistra = 0;
    unsigned int maxsin = vsinistra.size();
    unsigned int indicedestra = 0;
    unsigned int maxdes = vdestra.size();

    for (unsigned int i = inizio; i <= fine; ++i) {
        if (indicesinistra < maxsin && indicedestra < maxdes) {
            if (vsinistra[indicesinistra] < vdestra[indicedestra]) {
                v[i] = vsinistra[indicesinistra];
                indicesinistra++; continue;
            } else {
                v[i] = vdestra[indicedestra];
                indicedestra++; continue;
            }
        }
        if (indicesinistra == maxsin && indicedestra < maxdes) {
            v[i] = vdestra[indicedestra];
            indicedestra++; continue;
        }
        if (indicedestra == maxdes && indicesinistra < maxsin) {
            v[i] = vsinistra[indicesinistra];
            indicesinistra++; continue;
        }
    }
}
```

• complessità

Vale Θ(n log n) sia nel caso migliore che peggiore, infatti non essendo un algoritmo adattivo, effettua tutte le chiamate ricorsive necessarie, e tutti i confronti, anche nel caso in cui l'array preso in considerazione sia già ordinato.
La complessità deriva dal calcolo dell'espressione:

n° livelli · costo ciascun problema

livello	numero problemi	dimensione problema
0	1	n

0	1	n
1	2	$n/2$
2	4	$n/4$
3	8	$n/8$
j	2^j	$n/2^j$

COSTO DI OGNI LIVELLO

Ad ogni livello j si hanno 2^j sottoproblemi di tipo merge, ognuno lungo $n/2^j$ e quindi risolvibile in $\Theta(n/2^j)$. Per conoscere il costo di ogni livello, bisogna moltiplicare il costo merge per il numero di volte che viene chiamato:

$$2^j \cdot \frac{n}{2^j} = n \Rightarrow \Theta(n)$$

NUMERO DI LIVELLI

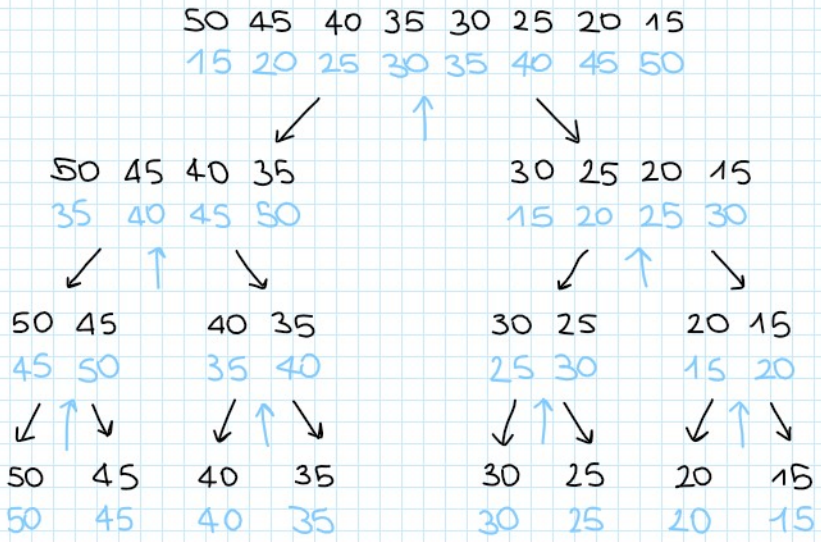
Sapendo che all'ultimo livello della ricorsione, i sottoproblemi assumano dimensione 1, e che su ogni livello j, i sottoproblemi sono di dimensione $n/2^j$, per quale j si ha che $n/2^j = 1 ? \rightarrow j = \log_2 n$. Quindi il numero di livelli è: $\log_2 n + 1$ [+1 perché si parte dal livello 0]

In conclusione il costo di merge sort è dato da: $\Theta(n) \cdot \log_2 n \Rightarrow \Theta(n \log n)$
Sia nel caso migliore che in quello peggiore
(il log n deriva dal numero dei livelli dell'albero della ricorsione)

merge sort esempio esame

- Si simuli un'esecuzione della chiamata di merge sort sulla sequenza riportata sotto. Questa situazione coincide con il caso migliore, peggiore o medio? Qual'è la complessità di merge sort in questo caso?
SEQUENZA: 50 45 40 35 30 25 20 15

simulazione chiamata:



- Vale $\Theta(n \log n)$ sia nel caso migliore che nel caso peggiore, infatti non essendo un algoritmo adattivo, effettua tutte le chiamate ricorsive necessarie, e tutti i confronti, anche nel caso in cui l'array sia già ordinato.
- La complessità dell'algoritmo deriva dal calcolo dell'espressione:
  n° livelli · costo operazioni di ogni livello
- Il costo operazioni singolo livello si calcola come:

- La complessità dell'algoritmo deriva dal calcolo dell'espressione:
  n° livelli · costo operazioni di ogni livello
- Il costo operazioni singolo livello si calcola come:
  numero problemi · dimensione problema $\rightarrow 2^j \cdot n/2^j = n \rightarrow \Theta(n)$

livello(j)	numero problemi(2^j)	dimensione problema ($n/2^j$)	sequenza
0	1	n	50 45 40 35 30 25 20 15
1	2	n/2	S1: 50 45 40 35 S2: 30 25 20 15
2	4	n/4	S1: 50 45 S2: 40 35 S3: 30 25 S4: 20 15
3	8	n/8	S1: 50 S2: 45 S3: 40 S4: 35 S5: 30 S6: 25 S7: 20 S8: 15
4	16	n/16	Ogni sequenza è formata da un solo elemento e da qui in poi si "torna su" nell'albero della ricorsione, ogni volta sistemando minore e maggiore