Chapter 1

Introduzione

1.1 Notazione O-Ω-Θ (pag 4)

Per semplificare l'analisi asintotica degli algoritmi sono state introdotte le seguenti notazioni:

$$O(f) = \{g | \exists a > 0 : \exists b > 0 : \forall N \in N : g(N) \leq af(N) + b\}$$

$$\Omega(f) = \{g | \exists c > 0 : \exists \text{ infinite n } : g(n) \geq cf(n)\}$$

In entrambi i casi si tratta di classi di funzioni. Quando $f \in O(g)$ e $f \in \Omega(g)$ al contempo si dice che $f \in \Theta(g)$. Esistono ulteriori definizioni alternative, ad esempio quelle utilizzate nel Blatt 1. Il seguente teorema può essere utile (dimostrazione nel Blatt 1):

Theorem 1 *Date due funzioni $f, g : N \to R^+$. Se $\lim_{x \to +\infty} \frac{f(n)}{g(n)}$ converge ad una costante $C \geq 0$ allora $f \in O(g)$.*

1.2 Algoritmo di Karatsuba (pag 12)

Un esempio di algoritmo più efficiente per multiplicare due numeri è il seguente:

$$65*28 = (2*6)*100+(2*6)*10+(5*8)*10+5*8+(6-5)*(8-2)*10 = 1820$$

In questo modo abbiamo solo 3 multiplicazioni elementari aniché 4. L'algoritmo può essere generalizzato grazie a divide and conquer dividendo ogni numero in due ed applicando l'algoritmo di base. Analizzando il tempo in base al numero delle multimplicazioni otteniamo che impieda circa $O(n^{1,58})$.

1.3 Maximum subarray (pag 20)

Dato un array di numeri il problema consiste nella ricerca del subarray con la somma degli elementi maggiore. Se essa è negativa il risultato è 0. Il metodo più efficiente per ricavare il risultato è il seguente:

```
A=array da 1, ... n
max=0
scan=0
for (i=1; i≤n; i++){
    scan+=A[i]
    if scan < 0
        scan=0
    if scan > max
        max=scan
}
```

In questo modo il problema viene risolto in tempo lineare.

Chapter 2

Sort

Per semplicità ammettiamo che si debba sempre ordinare un array (chiamato a) contenente n numeri (interi). In ogni caso con questi algoritmi è possibile ordinare qualsiasi oggetto appartenente ad un universo in cui vige un ordine totale.

2.1 Sortieren durch Auswahl (pag 82)

Selection sort consiste nel cercare ogni volta il minimo tra la posizione i e n. Una volta trovato esso viene scambiato con l'i-tesimo numero. Esempio:

15 2 43 17 4
2 15 43 17 4
2 4 43 17 15
2 4 15 17 43

Implementazione

```
int i,j, min, temp
for i=1:n-1
     min=i
     for j=(i+1):n
          if a[j]< a[min]
               min=j
     temp=a[min]
     a[min]=a[i]
     a[i]=temp
```

Analisi:

Dal doppio loop si vede semplicemente che l'algoritmo impiega $\Theta(n^2)$ comparazioni e nel peggior caso (i numeri sono ordinati dal più grande al più piccolo) O(n) scambi. Da notare che per trovare il minimo sono necessari almeno n-1 confronti (Satz 2.1), quindi l'algoritmo non può andare più veloce di così.

2.2 Sortieren durch Einfügen (pag 85)

In inglese si chiama insertion sort. Per induzione i numeri fino a i-1 sono già ordinati. Il principio consiste di piazzare l'i-tesimo elemento al giusto posto, se necessario scalando i restanti a destra di una posizione. Esempio:

2 15 / 43 17 4
2 15 43 / 17 4
2 15 17 43 / 4
2 4 15 17 43 /

Implementazione

```
int i,j,t
for i=2:n
    j=i
    t=a[i]
    while a[j-1] > t
        //scala a destra di una posizione
        a[j]=a[j-1]
        j=j-1
    a[j]=t
```

Si nota subito che se implementato così l'algoritmo può non fermarsi se t è il più piccolo numero. Serve quindi un elemento di stop, ad esempio inserendo a[0]=t prima del while loop.

Analisi:

Nel peggior caso $\Theta(n^2)$ comparazioni ed altrettanti spostamenti. Nel miglior caso $O(n^2)$ comparazioni e spostamenti. Il caso medio rispecchia il peggiore.

2.3 Bubblesort

Il principio di questo algoritmo è semplicissimo: ad ogni iterazione viene scambiato l'elemento a[i] con a[i+1] (chiaramente solo se maggiore). In questo modo l'elemento più grande si sposta verso destra. Esempio:

15 2 43 17 4
2 15 43 17 4
2 15 17 43 4
2 15 17 4 43
2 15 4 17 43
2 4 15 17 43

Implementazione:

```
do
    flag=true
    for i=1:n-1
    if a[i] > a[i+1]
        swap(a[i],a[i+1])
        flag=false
while (!flag)
```

Analisi:

Nel miglior caso, se l'array è già ordinato, abbiamo n-1 paragoni e nessuno scambio. Nel caso medio e peggiore l'algoritmo necessita di $\Theta(n^2)$ scambi e paragoni.