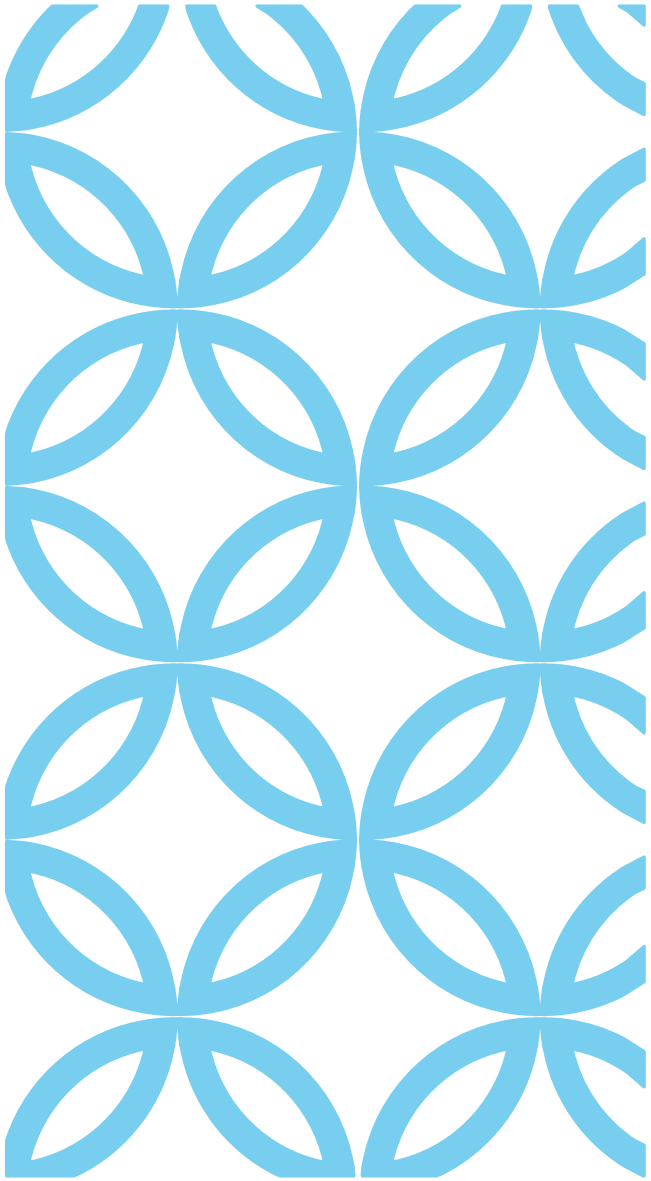


Corso di Programmazione e Strutture Dati

Docente di Laboratorio: Marco Romano
Email: marromano@unisa.it

RICORSIONE 2

RICORSIONE E VALUTAZIONE DELLA COMPLESSITÀ

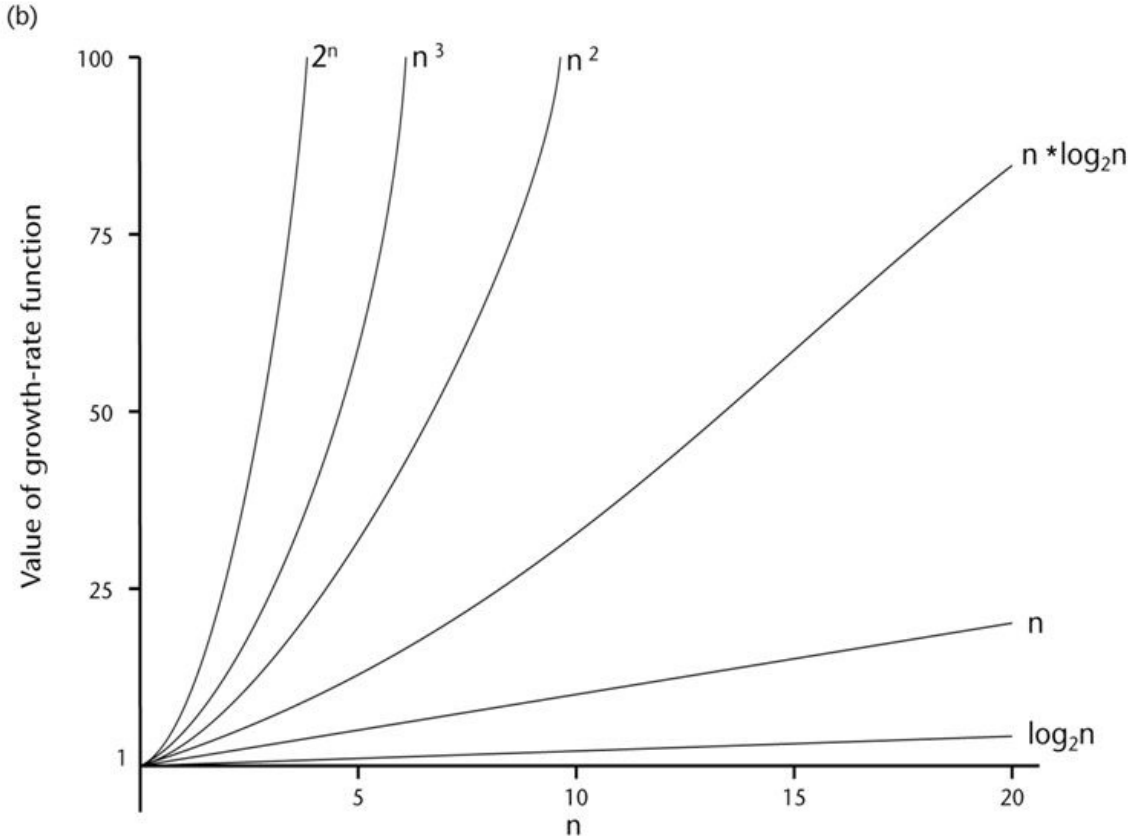
1. **Lavoro di combinazione costante**
   a) $T(n) = a_1\ T(n-1) + a_2\ T(n-2) + \ldots\ a_h\ T(n-h) + b$ per $n > h$
      1. Esponenziale con n: se sono presenti almeno 2 termini (l'algoritmo contiene almeno 2 chiamate ricorsive)
      2. Lineare con n: se è presente un solo termine (singola chiamata ricorsiva)

   b) $T(n) = a\ T(n/p) + b$ per $n > 1$
      1. $\log n$ se $a = 1$ (singola chiamata ricorsiva)
      2. $n^{\log_p a}$ se $a > 1$ (più chiamate ricorsive)

2. **Lavoro di combinazione lineare**
   a) $T(n) = T(n-h) + b\ n + d$ per $n > h$
      Quadratico con n

   b) $T(n) = a\ T(n/p) + b\ n + d$
      1. Lineare con n se $a < p$
      2. n log n se $a = p$
      3. $n^{\log_p a}$ se $a > p$

A Comparison of Growth-Rate Functions (cont.)



ESERCIZIO 1

- Si implementi la versione ricorsiva del Selection Sort. In particolare, occorre completare i file vettore.c, inserendo il nuovo operatore generico (**deve ordinare degli Item**) *selection_sort_ric*.
- Si scriva anche la sua complessità asintotica, fornendo una breve giustificazione sulla risposta.

SELECTION_SORT_RIC

```
36  void selection_sort_ric(Item a[], int n) {
37    if(n==1)
38      return;
39
40    int min = minimo(a+1, n-1)+1;
41
42    if (cmpItem(a[min],a[0])<0)
43      swap(&a[0], &a[min]);
44
45    selection_sort_ric(a+1, n-1);
46  }
```

MINIMO

```
15  int minimo(Item *a, int n) {
16      int min = 0, i;
17
18      for(i=1; i<n; i++)
19          if (cmpItem(a[i],a[min])<0)
20              min=i;
21
22      return min;
23  }
```

SELECTION_SORT_RIC

```
36  void selection_sort_ric(Item a[], int n) {
37    if(n==1)
38      return;
39
40    int min = minimo(a+1, n-1)+1;
41
42    if (cmpItem(a[min],a[0])<0)
43      swap(&a[0], &a[min]);
44
45    selection_sort_ric(a+1, n-1);
46  }
```

Lavoro di combinazione lineare

a) $T(n) = T(n-h) + b\,n + d$

Quadratico con n

ESERCIZIO 2

- Si implementi una procedura ricorsiva che inverte il contenuto di una coda.
- Si testi la procedura su una coda a valori interi.
- Si indichi anche la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

REVERSEQUEUE

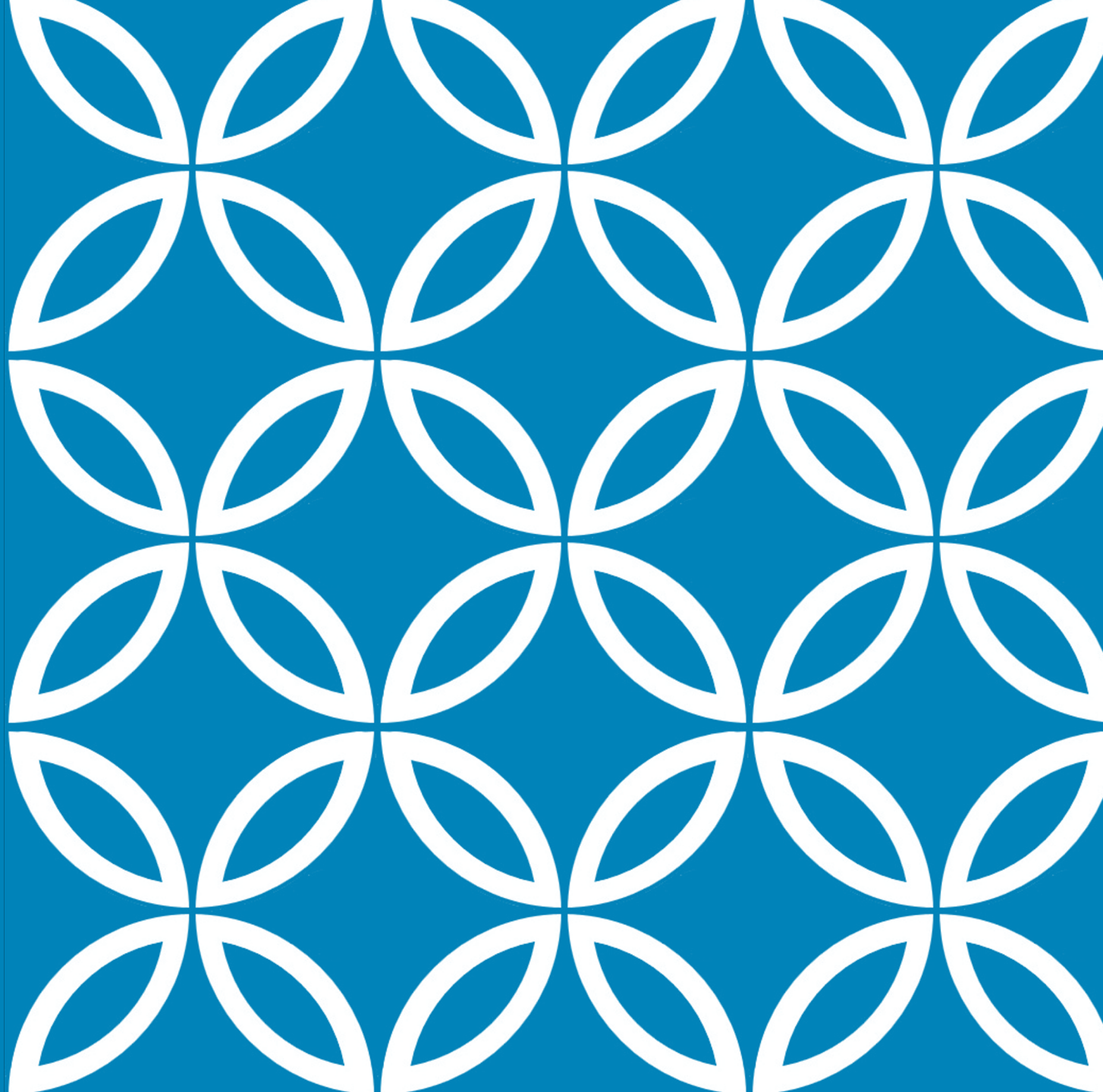
```
6   void reverseQueue(Queue q)
7   {
8      if(isEmptyQueue(q))
9         return;
10     Item it = dequeue(q);
11     reverseQueue(q);
12     enqueue(q,it);
13  }
```

REVERSEQUEUE

Lavoro di combinazione costante:
$T(n) = a_1\ T(n-1)$
Lineare con n

```
6   void reverseQueue(Queue q)
7   {
8      if(isEmptyQueue(q))
9         return;
10     Item it = dequeue(q);
11     reverseQueue(q);
12     enqueue(q,it);
13  }
```

CALCOLO DELLA COMPLESSITÀ SU OPERATORI RICORSIVI DI LIST



CALCOLO DELLA COMPLESSITÀ SU LISTA

```
277  void destroyNode(struct node *node)
278  {
279      if(node){
280          destroyNode(node->next);
281          free(node);
282      }
283  }
```

CALCOLO DELLA COMPLESSITÀ SU LISTA

```
260 int countItemNode (struct node *node, Item item)
261 {
262     if(node == NULL)
263         return 0;
264     else{
265         if (cmpItem (node -> item, item) == 0)
266             return countItemNode(node->next, item) + 1;
267         else
268             return countItemNode(node->next, item);
269     }
270 }
```

CALCOLO DELLA COMPLESSITÀ SU LISTA

```
239 Item searchNode (struct node *node, Item item, int* pos)
240 {
241     if (node == NULL)
242     {
243         *pos = -1;
244         return NULL;
245     }
246     if (cmpItem (node -> item, item) == 0)
247         return node -> item;
248     else
249     {
250         ++*pos;
251         return searchNode (node -> next, item, pos);
252     }
253 }
```

CALCOLO DELLA COMPLESSITÀ SU LISTA

```
226  void printNode (struct node *node)
227  {
228      if (node == NULL)
229          return;
230      outputItem (node -> item);
231      printNode (node -> next);
232  }
```