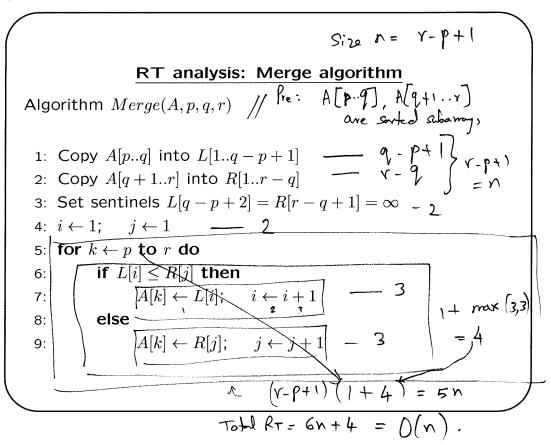
(=) k = a Logs: Def: logba = K in cs, b=2 In math/engq: default b = e, logn = logn Ynopentes: Inn = loger ling (AB) = log A + log B log (A/B) = log A - log B logkn = (logn) < polylog log(nk) = k.logn Exponentials: c' c= constant

X. \* X = X

(Xa) = X Polynomial in n:  $P(n) = a_0 + a_1 n_4 + ... \cdot a_k n_k$ k = constant,  $\alpha_k > 0$   $P(n) = O(n^k)$ . Theris: Polynomie PRT = fearible algorithe Exponentel or more = intersible.

ŧ

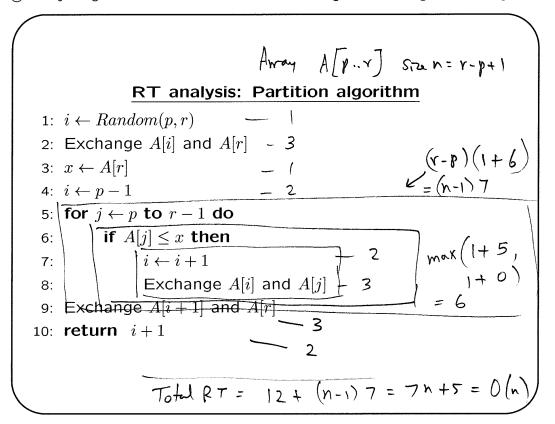
Sums of seguences: a,, a, ..., a, 1. Avitmete sequences:  $a_{i+1} - a_i = d = constant$ .  $E_{X}: 1, 2, 3, ..., N$  $2, 5, 8, 11, \ldots, 3n-1$ d < 0 = decreasy sequence d>0 = increasing sequence, Max term = an Max term = a,  $\begin{vmatrix} a_1 + a_2 + \cdots + a_n \end{vmatrix} = \sum_{i=1}^{n} a_i$   $\begin{vmatrix} a_1 + a_{n-1} + \cdots + a_i \end{vmatrix} = \sum_{i=1}^{n} a_i$  $\left(a_{i}+a_{n}\right)\left(a_{i}+a_{n}\right)\cdot\cdot\left(a_{i}+a_{n}\right)=2$   $\sum_{i=1}^{n}a_{i}=n\left(a_{i}+a_{n}\right)$  $\sum_{i=1}^{i} a_i = \frac{1}{2} n (a_i + a_n) = O(n, biggest term)$ 2. Geometric sepreces: Oo, a,..., a,  $\alpha_{i+1}/\alpha_i = r = constant$ . r > 1 = increasing squae r < 1 = decreasing squae Max ten = 0 n max ten = 0 max ten



© Balaji Raghavachari

4

Algorithm Design and Analysis



### RT analysis: Binary search

- 1: BSearch(A, p, r, x) // Search for x in sorted array A[p..r]
- 2: while  $p \leq r$  do

3: 
$$q \leftarrow (p+r)/2$$

4: if 
$$x < A[q]$$
 then

5: 
$$r \leftarrow q - 1$$

6: else if 
$$x > A[q]$$
 then

7: 
$$p \leftarrow q + 1$$

else 8:

**return** q // index of x in A 9:

10: return -1 // return -1 if x is not in A

# of iterations of while loop: 
$$1 + \log n$$
 $RT = O(\log n)$ 
 $N \to N/2 \to N/4 \to \cdots \to 0$ 

# styps:  $1 + \log n$ 

© Balaji Raghavachari

6

Algorithm Design and Analysis

Each time loop is executed,

n=r-pti

O(i) shroks by

1/2

## RT analysis: Bubble sort

1: BubbleSort(A, n) // Sort A[1..n]

2: for 
$$i \leftarrow 1$$
 to  $n$  do

3: **for** 
$$j \leftarrow n$$
 **downto**  $i+1$  **do**

4: if 
$$A[j] < A[j-1]$$
 then

5: 
$$\left( \begin{array}{c|c} A[j] < A[j-1] \text{ then} \\ Exchange } A[j] \text{ and } A[j-1] \end{array} \right) O(\iota)$$

$$\sum_{i=1}^{n} n - i = (n-1) + (n-2) + \dots + 1 = decreased arithmeter = 0 (n^2)$$
seque



Amortized analysis

# RT analysis: KMP algorithm for string matching

A useful loop invariant:  $\pi[k] < k$ , for k > 0.

1: 
$$\pi[1] \leftarrow 0$$

2: 
$$k \leftarrow 0$$

3: for 
$$q \leftarrow 2$$
 to  $m$  do

3: for 
$$q \leftarrow 2$$
 to  $m$  do

4: while  $k \gg 0$  and  $P[k+1] \neq P[q]$  do

5:  $k \leftarrow \pi[k]$ 

6: if  $P[k+1] = P[q]$  then

7:  $k \leftarrow k+1$ 

8:  $\pi[q] \leftarrow k$ 

6: 
$$|\overline{\mathbf{if}} P[k+1] = P[a] \mathbf{then}$$

$$k \leftarrow k+1$$

8: 
$$\pi[q] \leftarrow k$$

9: **return**  $\pi$ 

0(m)

### **Asymptotic notation**

Let f and g be discrete, non-negative functions. In the following definitions, b, c and  $n_0$  are positive constants.

- Upper bound (big oh): f(n) = O(g(n)), if there exist c and  $n_0$  such that  $f(n) \le c \cdot g(n)$  for  $n \ge n_0$ .
- Lower bound (big omega):  $f(n) = \Omega(g(n))$ , if there exist c and  $n_0$  such that  $f(n) \ge c \cdot g(n)$  for  $n \ge n_0$ .
- Tight bound (theta):  $f(n) = \Theta(g(n))$ , if there exist b, c, and,  $n_0$  such that  $b \cdot g(n) \le f(n) \le c \cdot g(n)$  for  $n \ge n_0$ .
- Weak upper bound (little oh): f(n) = o(g(n)), if for any c, there exists  $n_0$  such that  $f(n) < c \cdot g(n)$  for  $n \ge n_0$ .
- Weak lower bound (little omega):  $f(n) = \omega(g(n))$ , if for any c, there exists  $n_0$  such that  $f(n) > c \cdot g(n)$  for  $n \ge n_0$ .

© Balaji Raghavachari

19

Algorithm Design and Analysis

#### The limit method

The following rules can be derived from the definitions, and are more convenient to derive asymptotic bounds, when the limit  $k = \lim_{n \to \infty} \frac{f(n)}{g(n)}$  exists. Often, L'Hôpital's rule is used to calculate this limit.

- f(n) = O(g(n)), if  $k \neq \infty$ .
- $\bullet \ f(n) = \Omega(g(n)), \ \text{if} \ k \neq 0.$
- $f(n) = \Theta(g(n))$ , if  $0 < k \neq \infty$ .
- $\bullet \ f(n)=o(g(n))\text{, if } k=0.$
- $\bullet \ f(n)=\omega(g(n))\text{, if }k=\infty.$

### **Proof techniques**

The following are valid techniques for writing proofs:

- Induction. Prove that  $T(n)=2^n-1$  is the solution to the recurrence T(n)=2T(n-1)+1, T(1)=1. Prove that  $\lim_{n\to\infty}\frac{\log^k n}{n^\epsilon}=0$ , for any positive constants k and  $\epsilon$ .
- ullet Contradiction. If f is a maximum flow, then the residual network  $G_f$  does not have a path from s to t.
- Contrapositive. If the product of two integers is even, then at least one of them must be even.
- Example/Counterexample. If S, V S is a cut crossed by an edge e of a minimum spanning tree T, prove that it is not necessarily a light edge for this cut.
- Construction. If f is a maximum flow, then there exists a cut (S,T) such that |f|=c(S,T).

© Balaji Raghavachari

33

Algorithm Design and Analysis

#### **Proof by induction**

Given a sequence of propositions,  $P_0, P_1, P_2, ...$ , we can show that they are true as follows:

- Base cases: From first principles, prove that  $P_0, P_1, P_2, \dots, P_{n_0}$  are true, for some  $n_0 \ge 0$ .
- Induction step: Prove that for any  $n > n_0$ ,  $P_0, \ldots, P_{n-1}$ , the Induction hypothesis (IH), implies  $P_n$ :

$$P_0 \wedge \ldots \wedge P_{n-1} \implies P_n$$

ullet Based on the above, we can claim  $P_n$  is true, for  $n\geq 0$ . Why? Suppose  $k\geq 0$  is smallest value for which  $P_k$  is false. By the base case,  $k>n_0$ . Since k is smallest counterexample,  $P_0,P_1,P_2,\ldots,P_{k-1}$  are true. By the induction step,  $P_k$  is true, which contradicts the claim that  $P_k$  is false.

### **Loop invariants**

Proving correctness of programs with loops:

```
// Precondition: Pre while Condition c do Statements S // Postcondition: Post (to be proved)
```

To prove that the postcondition is satisfied, we write a mathematical proposition, known as a <u>Loop Invariant</u> (LI), and a proof structured as follows:

ullet Initialization: Pre implies LI

• Maintenance: LI, c, and execution of S, implies LI.

• Termination: LI, and, not c, implies Post.

Show that the number of iterations is finite separately, or include it as part of the LI.