GREEDY ALGORITHMS

INTRODUCTION

- Change Making Problem
- You Purchase a Idli & Half Tea at a canteen.
- Total bill (including GST) is 23rs.
- You pay 100rs note to the cashier.
- The balance amount is 77rs.
- What will be the approach of the cashier to pay you the balance?

INTRODUCTION

- Cashier will first pick 50rs note.
- Then he will pick 20rs note
- OR
- He will pick Two 10rs notes.
- Then he will pick a 5rs coin.
- Then finally a 2rs coin or Two 1rs coins.
- OR
- A 2rs Chocolate

INTRODUCTION

- Change Making Problem
- At least subconsciously, by millions of cashiers all over the world: give change for a specific amount n with the least number of coins of the denominations d1>d2 > . . .>dm used in that locale.

INTRODUCTION

- A greedy algorithm is an **algorithmic paradigm** that follows the problem solving heuristic of making the **locally optimal choice** at each stage with the hope of finding a **global optimum**.

INTRODUCTION

- The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
- On each step the choice made must be:
  - **Feasible**, i.e., it has to satisfy the problem's constraints
  - **Locally Optimal**, i.e., it has to be the best local choice among all feasible choices available on that step

INTERVAL SCHEDULING PROBLEM

- We have a set of requests {1, 2, . . . , n}.
- The ith request corresponds to an interval of time starting at s(i) and finishing at f (i).
- We'll say that a subset of the requests is compatible if no two of them overlap in time.
- **Goal** is to accept as large a compatible subset as possible.

- Compatible sets of maximum size will

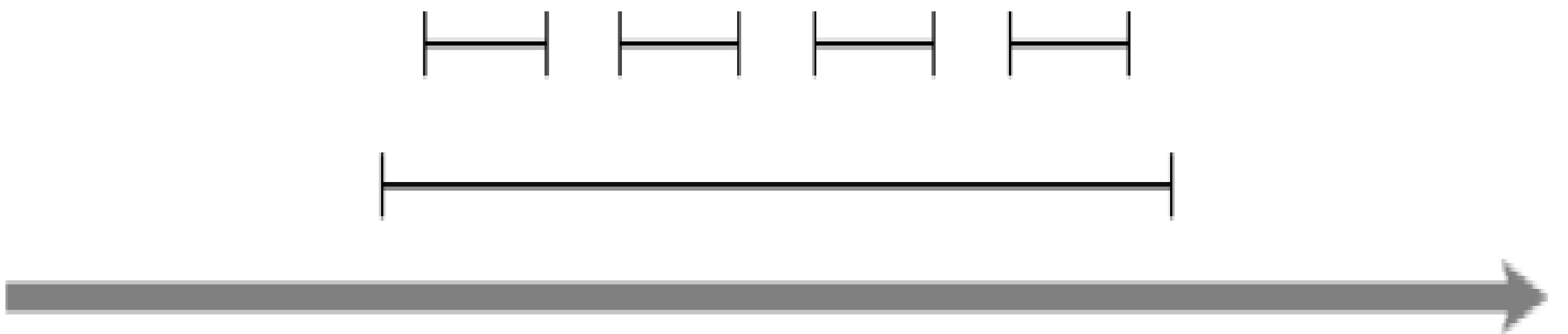
INTERVAL SCHEDULING PROBLEM

- **Designing a Greedy Algorithm using natural approaches:**
  - The basic idea in a greedy algorithm for interval scheduling is to use a simple rule to select a first request i1.
  - Once a request i1 is accepted, we reject all requests that are not compatible with i1.
  - We then select the next request i2 to be accepted, and again reject all requests that are not compatible with i2 and so on.

INTERVAL SCHEDULING PROBLEM

- **Natural Rules that can be used to solve the interval scheduling problem are :**
  - Select the available request that starts earliest - that is, the one with minimal start time s(i) or earliest start time.
  - This way our resource starts being used as quickly as possible.
  - If the earliest request i is for a very long interval, then by accepting request i we may have to reject a lot of requests for shorter time intervals.

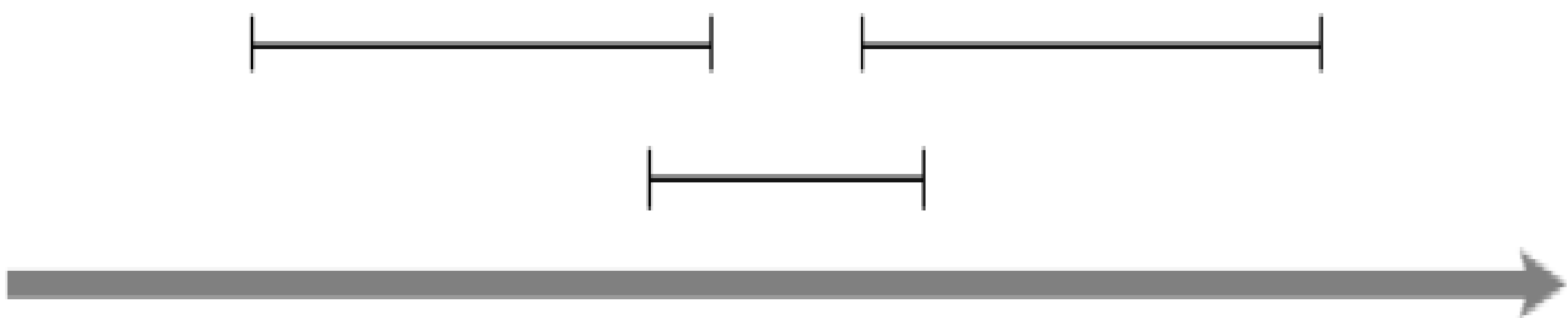
INTERVAL SCHEDULING PROBLEM



INTERVAL SCHEDULING PROBLEM

- **Natural Rules that can be used to solve the interval scheduling problem are :**
  - By accepting the request that requires the smallest interval of time—namely, the request for which $f(i) - s(i)$ is as small as possible.
  - As it turns out, this is a somewhat better rule than the previous one, but it still can produce a suboptimal schedule.

INTERVAL SCHEDULING PROBLEM

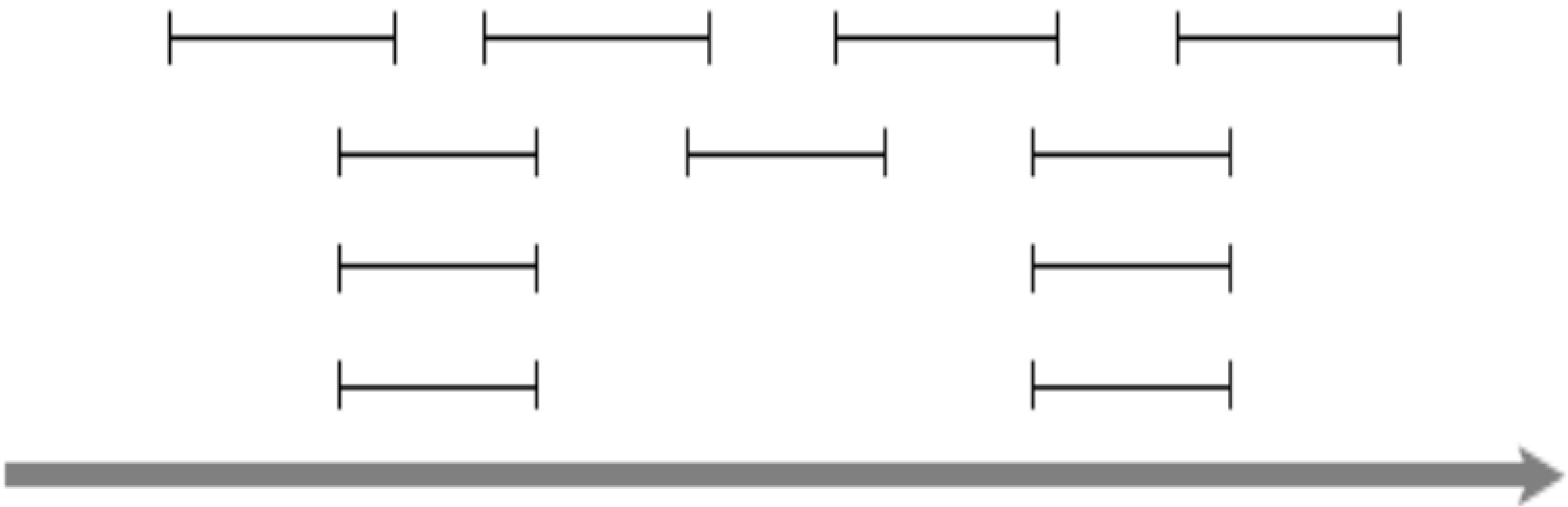


INTERVAL SCHEDULING PROBLEM

- **Natural Rules that can be used to solve the interval scheduling problem are :**
  - A greedy algorithm that is based on this idea:

  - For each request, we count the number of other requests that are not compatible, and accept the request that has the fewest number of non-compatible requests.
  - In other words, we select the interval with the fewest “conflicts”.

INTERVAL SCHEDULING PROBLEM



INTERVAL SCHEDULING PROBLEM

- **Natural Rules that can be used to solve the interval scheduling problem are :**
  - A greedy rule that does lead to the optimal solution is based on a fourth idea:
  - We should accept first the request that finishes first, that is, the request i for which f (i) is as small as possible.

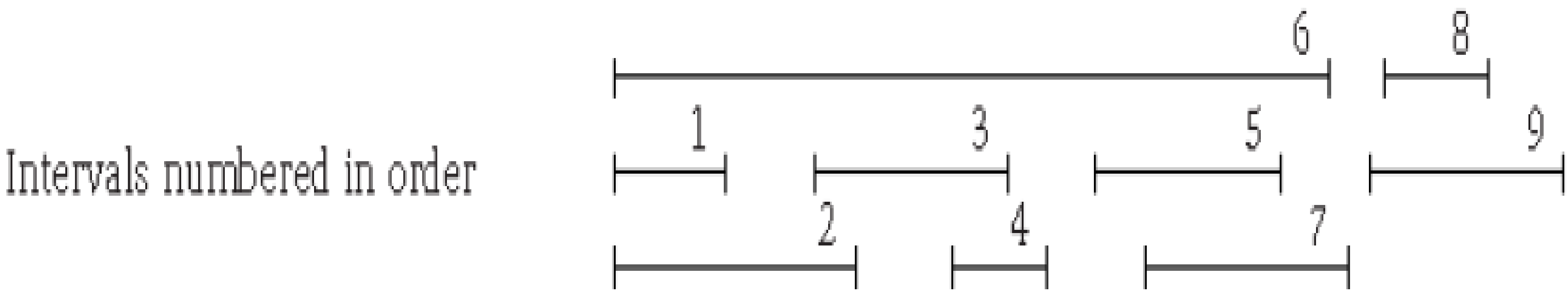
INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Algorithm**

```
Initially let R be the set of all requests, and let A be empty
    While R is not yet empty
        Choose a request i ∈ R that has the smallest finishing
        time
        Add request i to A
        Delete all requests from R that are not compatible with
        request i
    EndWhile
Return the set A as the set of accepted requests
```

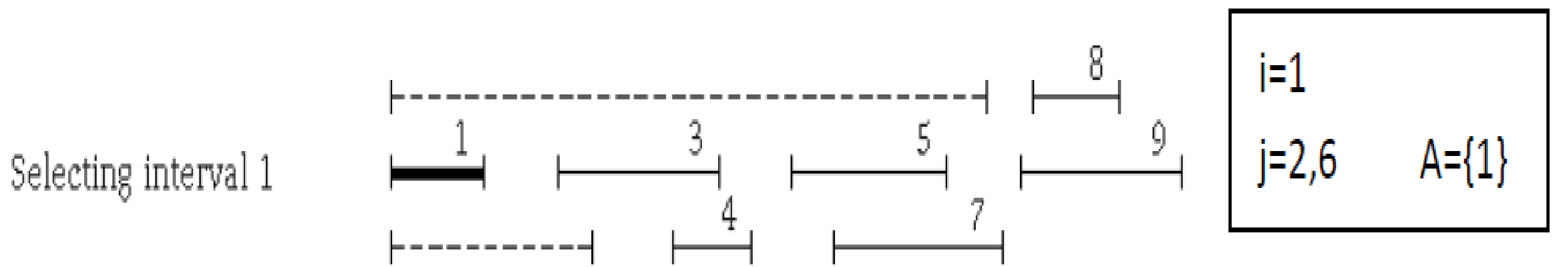
INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Example with Smallest Finish Time (Greedy)**



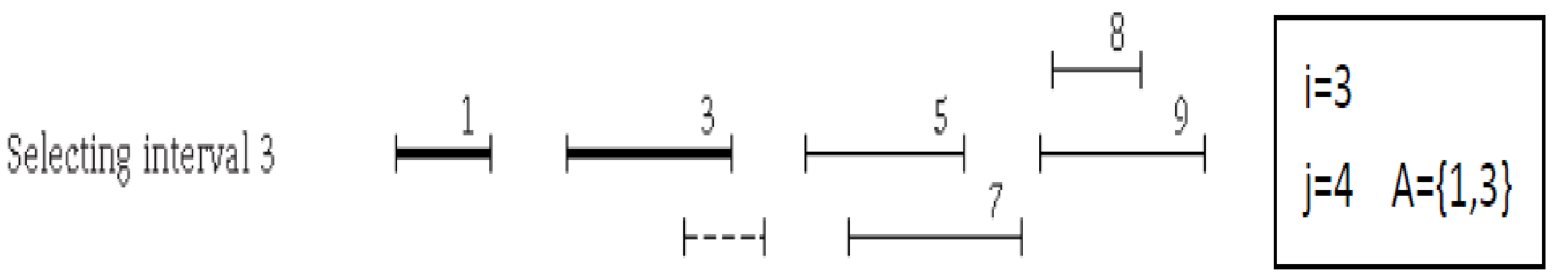
INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Example with Smallest Finish Time (Greedy)**



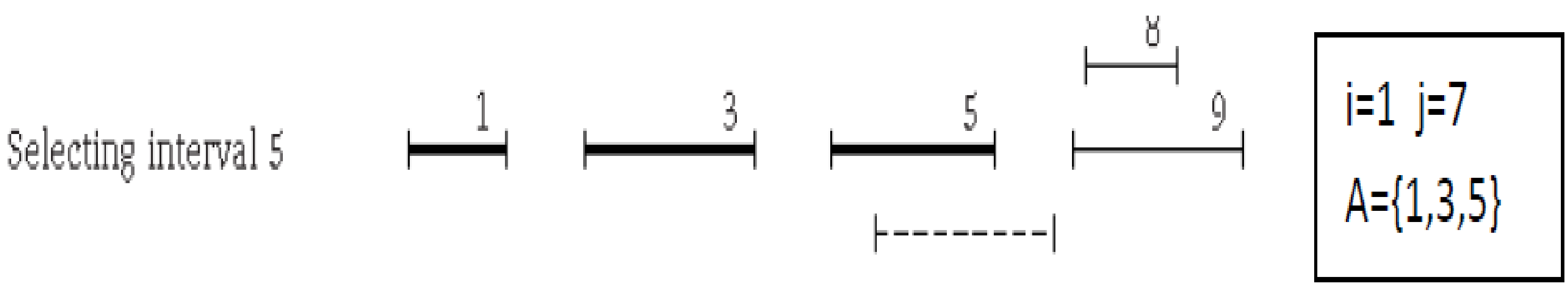
INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Example with Smallest Finish Time (Greedy)**



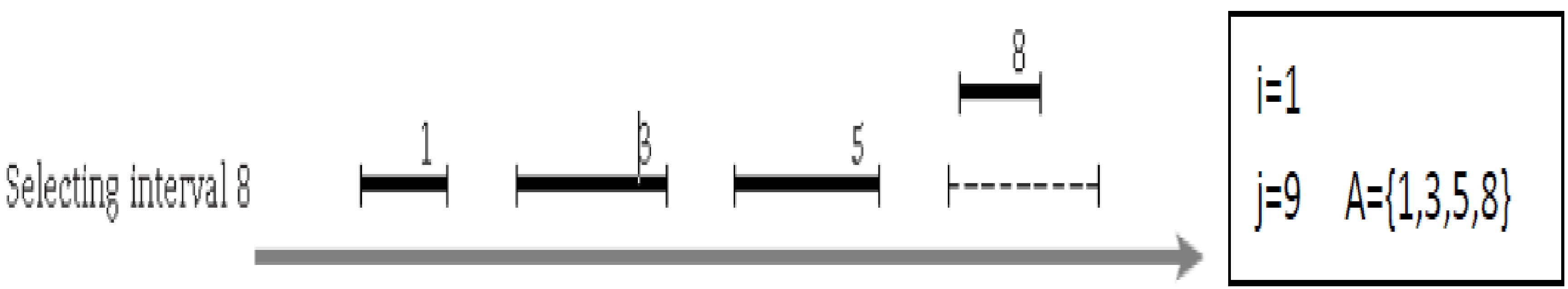
INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Example with Smallest Finish Time (Greedy)**



INTERVAL SCHEDULING PROBLEM

- **Interval Scheduling Example with Smallest Finish Time (Greedy)**



INTERVAL SCHEDULING PROBLEM

- Implementation and Running Time of Interval Scheduling:
- The running time of the algorithm shown is O(n log n) as follows.
- We begin by sorting the n requests in order of finishing time and labeling them in this order; that is, we will assume that $f(i) \leq f(j)$ when $i < j$.
- This takes time O(n log n).
- In an additional O(n) time, we construct an array S[1. . . n] with the property that S[i] contains the value s(i).
- We now select requests by processing the intervals in order of increasing f(i).

INTERVAL SCHEDULING PROBLEM

- We always select the first interval; we then iterate through the intervals in order until reaching the first interval j for which $s(j) \geq f\ (1)$; we then select this one as well.
- More generally, if the most recent interval we've selected ends at time f , we continue iterating through subsequent intervals until we reach the first j for which $s(j) \geq f$.
- In this way, we implement the greedy algorithm analyzed above in one pass through the intervals, spending constant time per interval. Thus this part of the algorithm takes time O(n)

SCHEDULING TO MINIMIZE LATENESS

- Consider again a situation in which we have a single resource and a set of n requests to use the resource for an interval of time.
- Assume that the resource is available starting at time S and each request is now more flexible.
- Instead of a start time and finish time, the request i has a deadline di, and it requires a contiguous time interval of length ti, but it is willing to be scheduled at any time before the

SCHEDULING TO MINIMIZE LATENESS

- Suppose that we plan to satisfy each request, but we are allowed to let certain requests run late.
- Thus, beginning at our overall start time S, we will assign each request i an interval of time of length ti;
  - Let us denote this interval by [s(i), f (i)], with f (i) as
  - f (i) = s(i) + ti.
  - where, s(i)= start time of the interval i
  - f(i)=finish time of the interval i
  - ti= time length of the interval i

SCHEDULING TO MINIMIZE LATENESS

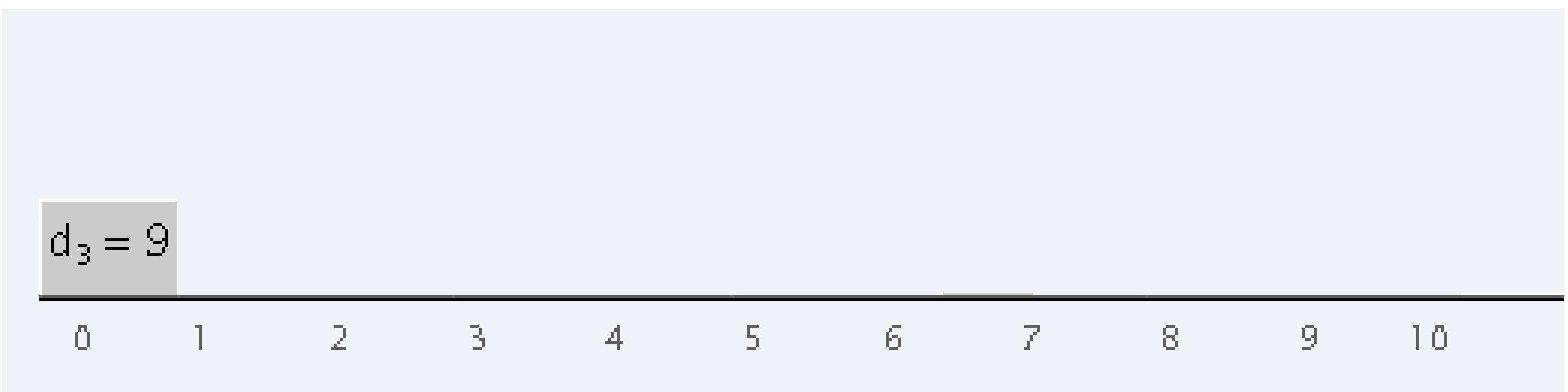
- The algorithm must determine a start time (and hence a finish time) for each interval.
- We say that a request i is late if it misses the deadline, that is, if **f (i) > di**.
- The lateness of such a request i is defined to be
  - **li = f (i) − di**
- We will say that li = 0 if request i is not late.
- The goal in our new optimization problem will be to schedule all requests, using non-overlapping intervals, so as to

SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

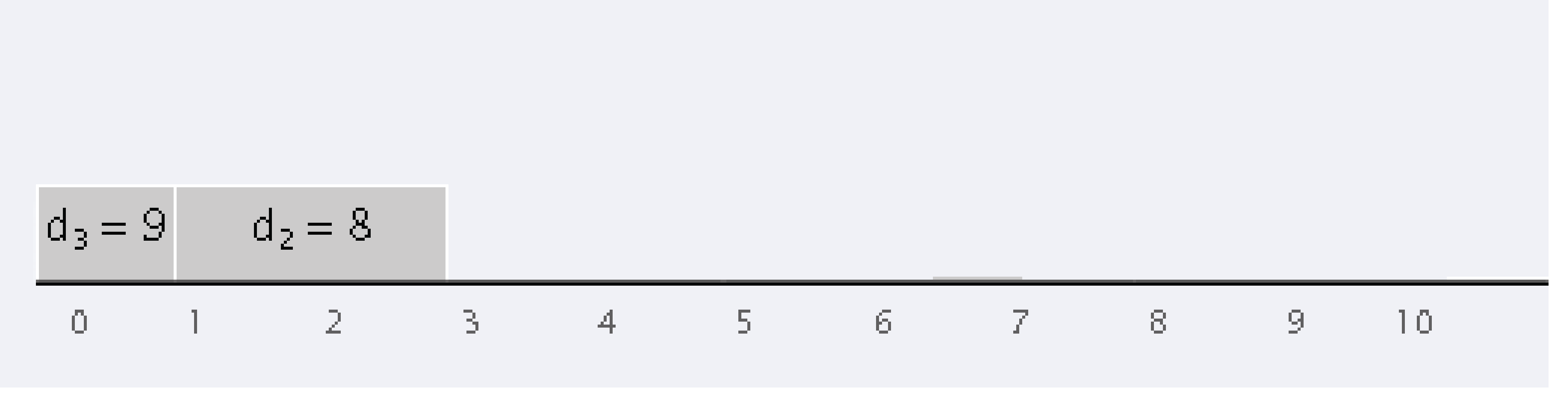
SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



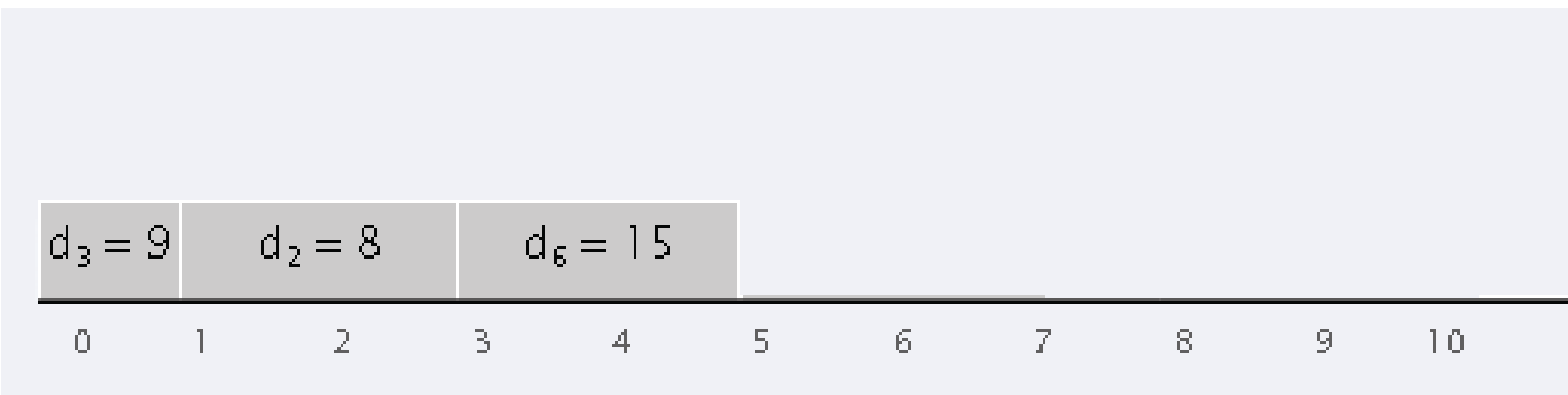
SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



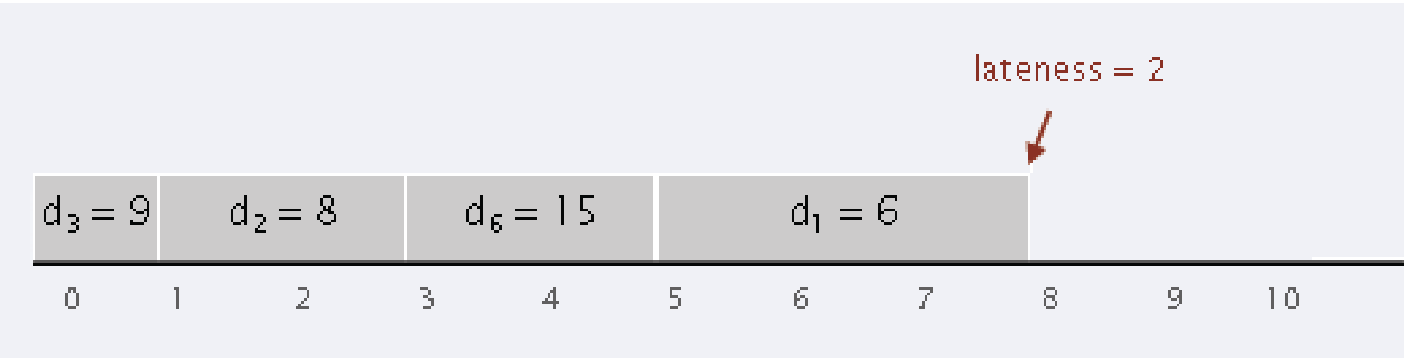
SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



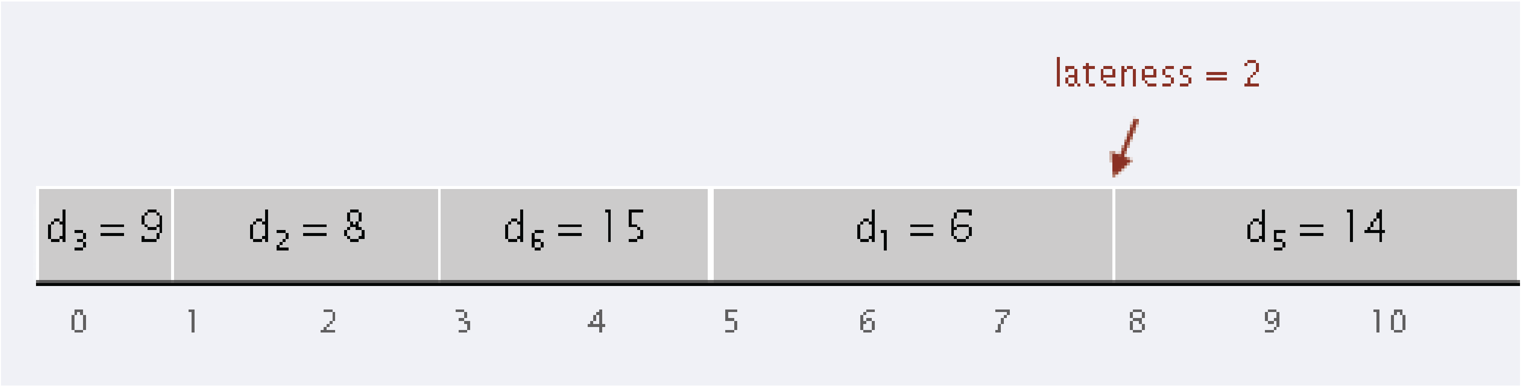
SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



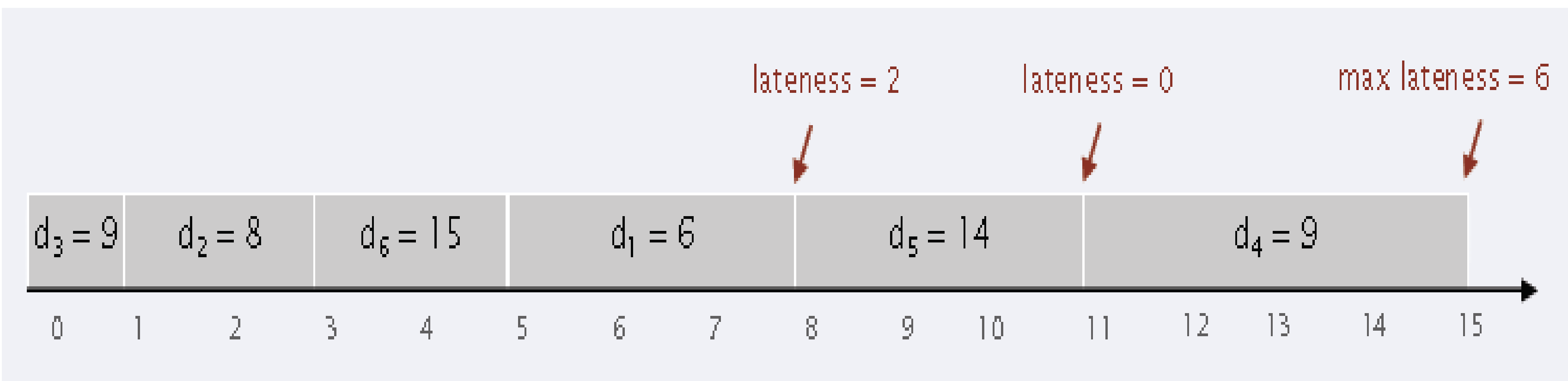
SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



SCHEDULING TO MINIMIZE LATENESS

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



SCHEDULING TO MINIMIZE LATENESS

- Natural Approaches to the Problem.
  - Schedule the jobs in order of incr

Request	1	2
t_j	1	3
d_j	2	5

$t_1=1$	$t_2=3$

0 1 3

SCHEDULING TO MINIMIZE LATENESS

Request	1	2
t_j	1	10
d_j	100	10

$t_1=1$, $d_1=100$	$t_2=10$, $d_2=10$

0 1 11

Lateness for the request 2 is 1

Counter example

If request 2 is scheduled first

$t_2=10$ $d_2=10$	$t_1=1$ $d_1=100$

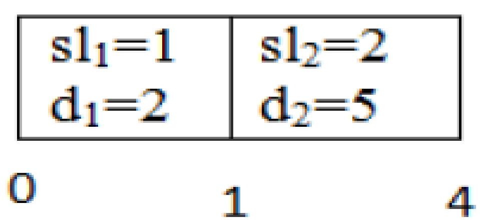
0 10 11

Lateness for the request 2 is 0

SCHEDULING TO MINIMIZE LATENESS

- Natural Approaches to the Problem.
  - Slack Time: di – ti is very small.

Request	1	2
t_j	1	3
d_j	2	5
Sl_i (Slack)	1	2



SCHEDULING TO MINIMIZE LATENESS

Counter example

Request	1	2
t_j	1	10
d_j	2	10
sl_i	1	0

If request 1
Was scheduled
first

$sl_2=0$ $d_2=10$	$sl_1=1$ $d_1=2$

0 10 11

Lateness for the request 1 is 9

$sl_1=1$ $d_1=2$	$s_2=0$ $d_2=10$

0 1 11

Lateness for the request 2 is 1

SCHEDULING TO MINIMIZE LATENESS – GOAL REVISITED

- The goal in our new optimization problem will be to schedule all requests, using non-overlapping intervals, so as to minimize the maximum lateness, $L = \max_i l_i$

SCHEDULING TO MINIMIZE LATENESS – GOAL REVISITED

Algorithm

Order the jobs in order of their deadlines
Assume for simplicity of notation that $d_1 \leq ... \leq d_n$
Initially, $f = s$
Consider the jobs $i = 1, ..., n$ in this order
    Assign job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$
    Let $f = f + t_i$
End
Return the set of scheduled intervals $[s(i), f(i)]$ for $i = 1, ..., n$

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

1. i=1 Add Job J1 to the schedule.

J1 d1=6

0 3

$s(1) = f = 0$

$f(1) = s(1) + t(1) = 0 + 3 = 3$

$f = f + t(1) = 0 + 3 = 3$

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

2. i=2 Add Job J2 to the schedule.

J1 d1=6	J2 d2=8

0 3 5

s(2) = f =3

f(2)= s(2)+t(2)=3+2=5

f= f + t(2)=3+2=5

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

3. i=3 Add Job J3 to the schedule.

J1 d1=6	J2 d2=8	J3 d3=9

0 3 5 6

$s(3) = f = 5$

$f(3) = s(3) + t(3) = 5 + 1 = 6$

$f = f + t(3) = 5 + 1 = 6$

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

4. i=4 Add Job J4 to the schedule.

s(4) = f =6

f(4)= s(4)+t(4)=6+4=10

f= f + t(4)=6+4=10

J1 d1=6	J2 d2=8	J3 d3=9	J4 d4=9
0 – 3	3 – 5	5 – 6	6 – 10

Lateness=1

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

5. i=5 Add Job J5 to the schedule.

s(5) = f =10

f(5)= s(5)+t(5)=10+3=13

f= f + t(5)=10+3=13

J1	J2	J3	J4	J5
d1=6	d2=8	d3=9	d4=9	d5=14

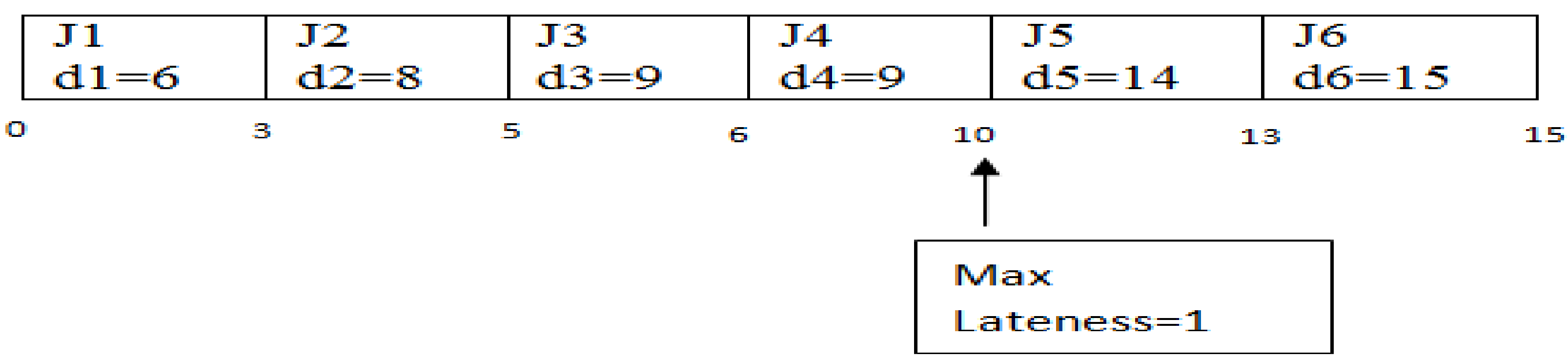
0 3 5 6 10 13

SCHEDULING TO MINIMIZE LATENESS – EXAMPLE

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

6. i=6 Add Job J6 to the schedule.

s(6) = f =13
f(6)= s(6)+t(6)=13+2=15
f= f + t(6)=13+2=15



OPTIMAL CACHING-PROBLEM

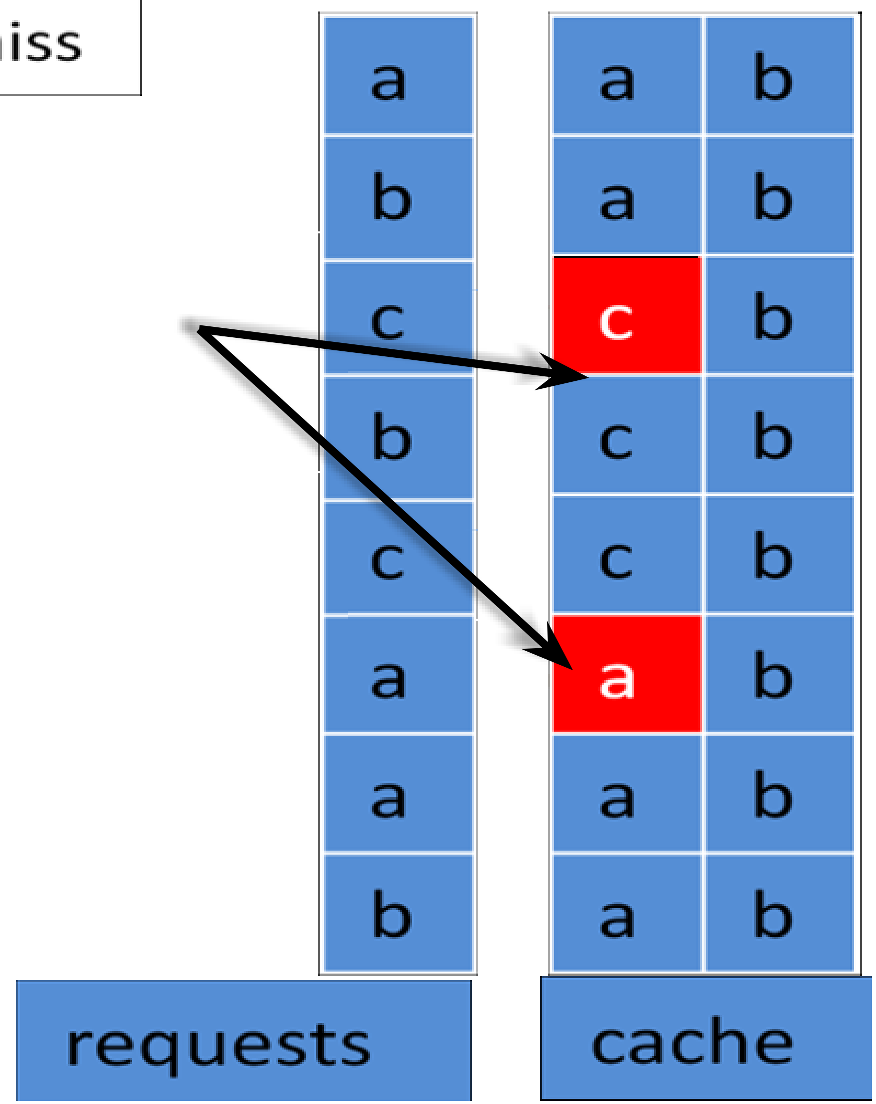
- Caching.
  - Cache with capacity to store k items.
  - Sequence of m item requests d1, d2, …, dm.
  - Cache hit: item already in cache when requested.
  - Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

OPTIMAL CACHING-PROBLEM

- Goal - Eviction schedule that minimizes number of cache misses.
  - Example
  - k = 2, initial cache = a b, requests: a, b, c, b, c, a, a, b.
  - Optimal eviction schedule: 2 cache misses.

OPTIMAL CACHING-PROBLEM

red = cache miss



OPTIMAL CACHING-ALGORITHM -Farthest-in-Future

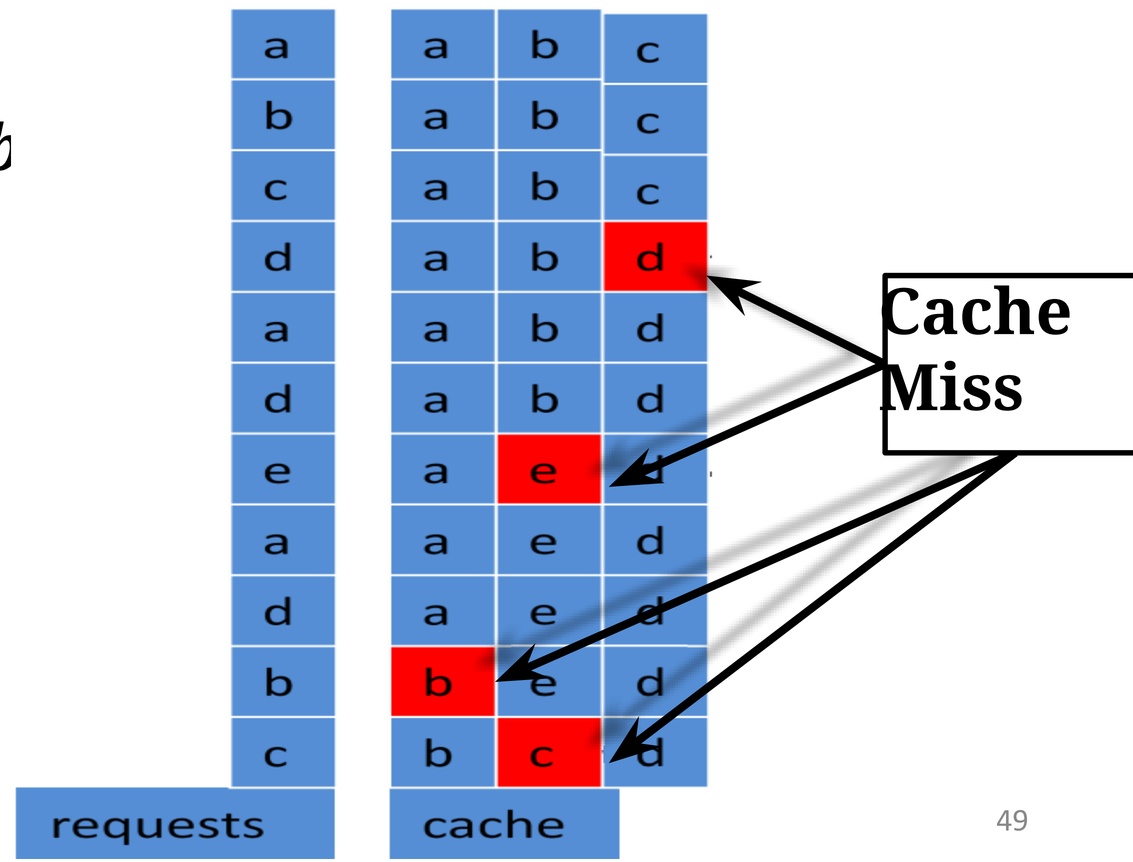
Algorithm

```
When di needs to be brought into the cache,
      evict the item that is needed the farthest into the future
```

Example

a, b, c, d, a, d, e, a, d, b

k=3



THANK YOU