4.3 Scheduling

4.3.1 Minimizing Total Time in the System

- **Time in the system**
  - **Time spent both waiting and being served**
- **Objective**
  - **Minimize total time**
- **Ex 4.2) $t_1 = 5$, $t_2 = 10$, $t_3 = 4$ (service time)**
  - **Possible schedules: n!**
  - **[1, 2, 3]: $TT = 5 + (5+10) + (5+10+4) = 39$**
  - **[3, 1, 2]: $TT = 4 + (4+ 5) + (4+5+10) = 32$**
- **Greedy approach**
  - **Schedule the job with the smallest service time first**

$$T(n) \in \Theta(n \log n)$$

Theorem 4.3 GA is optimal

- **Proof**
  - **Suppose that jobs are not scheduled in nondecreasing order by service time**
  - **Then, there exists *i* s.t.** $t_i > t_{i+1}$
  - **Schedule** **total time**

$$S\ :\ t_1\ t_2\ \cdots\ t_i\quad t_{i+1}\ \cdots\ t_n \qquad T$$

$$S'\ :\ t_1\ t_2\ \cdots\ t_{i+1}\ t_i\quad \cdots\ t_n \qquad T'$$

$$x = t_1 + \cdots + t_{i-1};\ \mathrm{X} = TT \text{ excluding } t_i\ \&\ t_{i+1}$$

$$T\ = X + (x + t_i) + (x + t_i + t_{i+1})$$

$$T' = X + (x + t_{i+1}) + (x + t_{i+1} + t_i)$$

$$T' = T + t_{i+1} - t_i < T \rightarrow \text{Contradiction!!}$$

Generalization to multiple server scheduling

$$t_1 < t_2 < \cdots < t_m < t_{m+1} < \cdots < t_n$$

Server 1	1	m+1	2m+1
Server 2	2	m+2	2m+2
	…	…	…
Server m	m	2m	3m

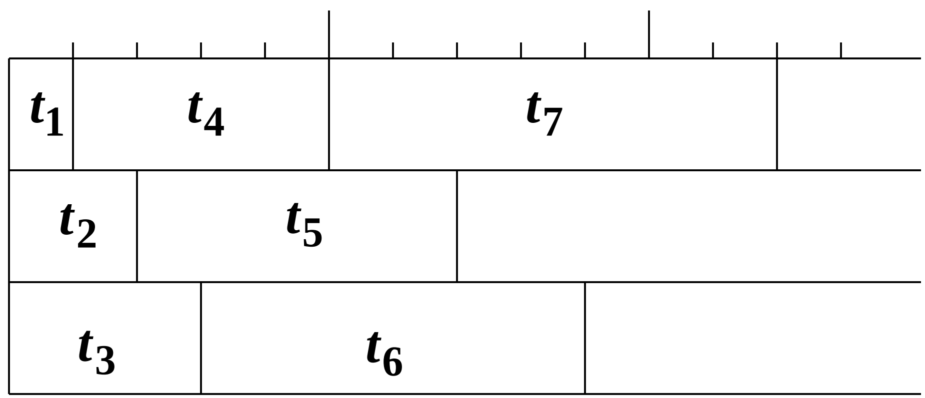
Example

- $N = 7, m = 3, \quad t_i = i$
- Let r_i be a one greater than the # of jobs following job i on its server.

Server 1

Server 2

Server 3



$r_1 = 3, r_4 = 2, r_7 = 1$

$r_2 = 2, r_5 = 1$

$r_3 = 2, r_6 = 1$

- Jobs with the same r_i can be scheduled on different servers.

Theorem Generalized GA is optimal

- **Proof**
  - ***TT* is given by** $TT = \sum_{i=1}^{n} r_i t_i$
  - **In any given scheduling, for any given *n*, there can be at most *m* jobs for which $r_i = j$.**
  - **From Th. 4.3, it follows that *TT* is minimized if the *m* longest jobs have $r_i = 1$, the next m longest jobs have $r_i = 2$ and so on.**

4.3.2 Scheduling with Deadlines

- **Job i processing time $t_i = 1$**
  deadline d_i
  profit p_i

- **Goal**
  - **Maximize the total profit**

- **Ex 4.3) $D = (2, 1, 2, 1)$, $P = (30, 35, 25, 40)$**
  - **Schedule total profit**
  - **[2, 1] $35 + 30 = 65$**
  - **[4, 1] $40 + 30 = 70$**

High-level greedy algorithm

```
Sort the jobs in nonincreasing order by profit;
S = ∅;
while (the instance is not solved) {
    select next job;                            // selection procedure
    if (S is feasible with this job added)      // feasibility check
        add this job to S;
    if (there is no more jobs)                  // solution check
        the instance is solved;
}
```

Example – ex. 4.4

$P = (40 \quad 35 \quad 30 \quad 25 \quad 20 \quad 15 \quad 10)$

$D = (3 \quad 1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2)$

S: [1] → [2, 1] → [2, 1, 4]

- See Alg. 4.4
- $J[i]$ is the i-th job in the optimal solution satisfying

$$D[J[i]] \leq D[J[i+1]]$$

Algorithm 4.4 Scheduling with Deadlines (1/2)

- **Problem**: Determine the schedule with maximum total profit given that each job has a profit that will be obtained only if the job is scheduled by its deadline.
- **Inputs**: n, the number of jobs, and array of integers *deadline*, indexed from 1 to n, where *deadline*[i] is the deadline for the i-th job. The array has been sorted in nonincreasing order according to the profits associated with the jobs.
- **Outputs**: an optimal sequence J for the jobs.

Algorithm 4.4 Scheduling with Deadlines (2/2)

```
void schedule (int n,
               const int deadline[ ],
               sequence_of_integer& J)
{
    index i;
    sequence_of_integer K;
    J = [1];
    for (i = 2; i <= n; i++) {
        K = J with i added according to nondecreasing values of deadline[i];
        if (K is feasible)
            J = K;
    }
}
```

Example – ex. 4.4

$p_1 \geq p_2 \geq \cdots \geq p_n$

$D = (\ 2 \quad 3 \quad 1 \quad 2 \quad 4)$

Step: $\quad J$

1 : J_1

2 : $J_1\ J_2$

3 : $J_3\ J_1\ J_2$

4 : $J_3\ J_1\ J_2$ **job 4 → rejected**

5 : $J_3\ J_1\ J_2\ J_5$

- **$W(n)$ of Alg. 4.4** $\boxed{W(n) \in \Theta(n^2)}$

Algorithm FJS using merge and find

Assign job i to the slot $[\alpha - 1, \alpha]$ where α is the largerst integer r such that $1 \le r \le d_i$ and the slot $[\alpha - 1, \alpha]$ is free.

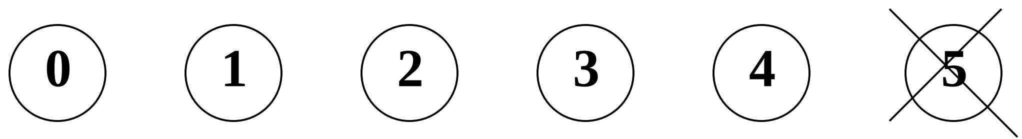
pointed by f

$$b = \min\{n,\ \max(D[i])\};$$

```
initial(b);                    /* modify b+1 trees → 0, 1, 2, … , b */
for (i = 0; i <= b; i++) f[i] = i; // Initialize trees
k = 0;
for (i = 1; i <= n; i++) {
    q = find(min(n, D[i]));
    if (f[q]) {
        k++; J[k] = i;
        m = find(f[q] - 1);  merge(m, q);  f[q] = f[m]; // To update f
    }
}
```

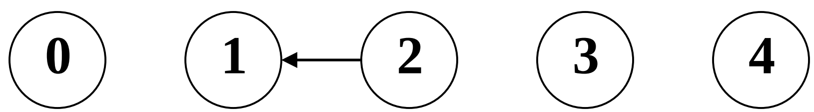
Example (1/2)

- **Step 0**



f : 0 1 2 3 4

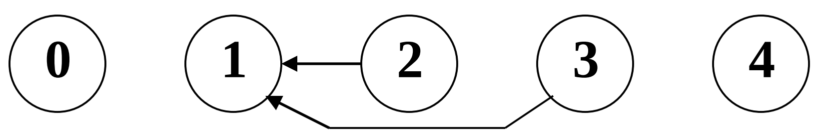
- **Step 1**



f : 0 1 1 3 4

***J*[1] = 1**

- **Step 2**

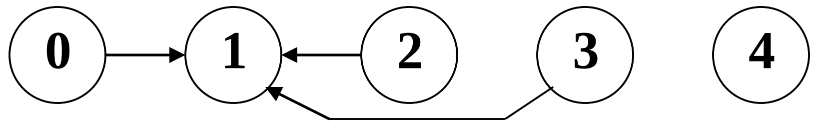


f : 0 1 1 1 4

***J*[2] = 2**

Example (2/2)

- **Step 3**



f : 0 0 1 1 4

***J*[3] = 3**

- **Step 4**

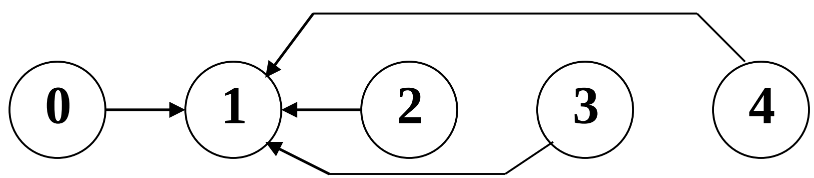
q = find (2) = 1

f[q] = 0

reject job 4

$T(n) \in \Theta(n \log n)$
sorting time

- **Step 5**



f : 0 0 1 1 0

***J*[4] = 5**

***J* ={1, 2, 3, 5} → sorting → S = {3, 1, 2, 5}**