## Scheduling

- Minimizing time in the system
- Scheduling with deadlines

## Minimizing time in the system

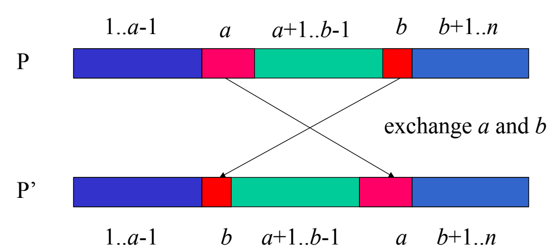
- Assume we submit $n$ jobs into a system at the same time.
  - The service time of job $i$, $t_i$, know in advance
- Problem
  - Design an algorithm that minimizing the average response time
  - This is equivalent to
    - Minimizing the total response time
    - $T = \sum_{i=1}^{n}$ (response time of job j)

## A greedy algorithm

- Algorithm
  1. Sort the jobs by their service times
  2. Repeat
     1. Serve the job with minimal service time among the remaining jobs
- Analysis
  - Step 1: O(n log n)
  - Step 2: Θ(n)
  - Total: O(n log n)

## Optimality of the greedy scheduling algorithm

- Theorem 6.5.1. The greedy algorithm is optimal




Compares schedules P and P', job $a$ at P' leaves at the same time as job $b$ in P. Jobs $b$ and $a$+1 to $b$-1 in P' leaves earlier then the corresponding jobs in P.

## Optimality of the greedy scheduling algorithm

$$
\begin{aligned}
T(P) &= s_1 + (s_1 + s_2) + \ldots (s_1 + s_2 + \ldots s_n) \\
&= ns_1 + (n-1)s_2 + \ldots + 1s_n \\
&= \sum_{k=1}^{n} (n-k+1)s_k
\end{aligned}
$$

$$T(P) = (n-a+1)s_a + (n-b+1)s_b + \sum_{k=1, k\neq a,b}^{n} (n-k+1)s_k$$

$$T(P') = (n-a+1)s_b + (n-b+1)s_a + \sum_{k=1, k\neq a,b}^{n} (n-k+1)s_k$$

$$T(P) - T(P') = (n-a+1)(s_a - s_b) + (n-b+1)(s_b - s_a) = (b-a)(s_a - s_b) > 0$$

## Scheduling with deadlines

- We have a set of jobs to execute, each of which take unit time.
- No concurrency and parallel: at any time we can execute exactly one job
- Job $i$ earn us a profit $g_i > 0$ if and only if it is executed no later than time $d_i$
- Problem: find a scheduling that maximize the total profits.

## Example

| | | | | |
|---|---|---|---|---|
| i | 1 | 2 | 3 | 4 |
| $g_i$ | 50 | 10 | 15 | 30 |
| $d_i$ | 2 | 1 | 2 | 1 |

| Sequence | Profit | Sequence | Profit | |
|---|---|---|---|---|
| 1 | 50 | 2, 1 | 60 | |
| 2 | 10 | 2, 3 | 25 | |
| 3 | 15 | 3, 1 | 65 | |
| 4 | 30 | 4, 1 | 80 | ← Optimum |
| 1, 3 | 65 | 4, 3 | 45 | |

## A greedy algorithm

- Feasible
  - A set of jobs is feasible if there exists at least one sequence (also called feasible) that allows all jobs in the set to be executed no late than their respective deadlines.
- Greedy algorithm
  - At each step, add the job with the highest profit ($g_i$) among those not yet considered, providing that the chosen set of jobs remains feasible.
- Theorem: the greedy algorithm above is optimal

## Proof of the optimality of the greedy algorithm

- Suppose the greedy algorithm choose the set of jobs I, and the set J is optimal. Let $S_I$ and $S_J$ be feasible sequences, possibly including gaps
  1. Rearrange the jobs in $S_I$ and $S_J$, we can obtain two feasible sequences $S_I$' and $S_J$', such that every job common to I and J is scheduled at the same time in both sequences
  2. Compare each pair of slots at the same time and show that I' and J' yield the same profit

## Rearranging Jobs, case I

- Job *a* occurs in both feasible sequences and is scheduled at time $t_I$ and $t_J$ respectively
  - $t_I = t_J$, we do not need to do anything

| | | | | | | $t_I = t_J$ | | |
|---|---|---|---|---|---|---|---|---|
| $S_I$ | | | | | | a | | |
| $S_J$ | | | | | | a | | |

## Rearranging Jobs, case II

- Job *a* occurs in both feasible sequences and is scheduled at time $t_I$ and $t_J$ respectively
  - $t_I < t_J$, we will try to move job a at $t_I$ to the slot $t_J$
    - Slot $t_J$ is a gap. Simply move
    - Job b is at slot $t_J$. Exchange job a and b.

| | | | $t_I$ | | | $t_J$ | | |
|---|---|---|---|---|---|---|---|---|
| $S_I$ | | | a | | | b | | |
| $S_J$ | | | | | | a | | |

## Rearranging Jobs, case III

- Job *a* occurs in both feasible sequences and is scheduled at time $t_I$ and $t_J$ respectively
  - $t_I > t_J$, similar to case II.

## Compare jobs in the rearranged sequences

- If some job *a* is schedule in $S_I$' opposite a gap in $S_J$', *a* does not belong J. J ∪ {*a*} is a feasible sequence and more profitable.
- If some job *b* is scheduled in $S_J$' opposite a gap in $S_I$'. The set I ∪ {*b*} will be feasible and the greedy algorithm would have included *b* in I.
- The only remaining possibility is that some job *a* is schedules in $S_I$' opposite a different job *b* in $S_J$'.
  - $g_a > g_b$. J is not optimal
  - $g_a < g_b$. The greedy algorithm should have chosen b first.
  - The only remaining possibility is $g_a = g_b$.

## Lemma

- Let J be a set of jobs. Suppose without loss of generality that the jobs are numbered so that

  $d_1 \le d_2 \le \ldots \le d_k$

- Then the set J is feasible if and only if the sequence 1, 2, …, k is feasible
  - Key to the proof: the pigeonhole principle

## An implementation

Worst Case: $O(n^2)$

```
Sequence(d[n+1])
{
   // assuming the profit is sorted in decreasing order
   d[0] = s[0] = 0; // sentinels
   s[1] = 1; // the first job is scheduled
   k = 1;  // # jobs already scheduled

   for (i=2; i<=n; i++) { // try job i
      // invariant: d[s[0]] <= d[s[1]]…<=d[s[k]]
      r = k;
      while (d[s[r]] > max(d[i], r)) r--; // r+1 is the earliest slot for job i
      if (d[i] > r) { // feasible if insert i
         for (j=k; j>r; j--)
             s[j+1] = s[j];
         s[r+1] = i;
         k++;
      }
   }
   return s;
}
```

## Example

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $g_i$ | 20 | 15 | 10 | 7 | 5 | 3 |
| $d_i$ | 3 | 1 | 1 | 3 | 1 | 3 |

d[s[i]]

s[i]

Initialization:

| | | | | | |
|---|---|---|---|---|---|
| 3 | | | | | |
| 1 | | | | | |

Try 2

| | | | | | |
|---|---|---|---|---|---|
| 1 | 3 | | | | |
| 2 | 1 | | | | |

Try 3 unchanged

Try 4

| | | | | | |
|---|---|---|---|---|---|
| 1 | 3 | 3 | | | |
| 2 | 1 | 4 | | | |

Try 5, 6 unchanged

Optimal sequence: 2, 1, 4

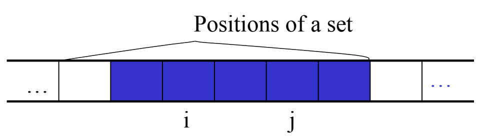
## A more efficient algorithm begins with the lemma below

- Lemma 6.6.4
  - A set of jobs $J$ is feasible if and only if we can construct a feasible sequence including all the jobs in $J$ as follows. Start with an empty schedule of length $n$. Then for each job $i \in J$ in turn, schedule i at time $t$, where $t$ is the largest integer such that $1 \le t \le min(n, d_i)$ and the job to be executed at time $t$ is not yet decided.
    - The "if" is obvious
    - The "only if"

## Proof of "only of"

- If the sequence is feasible, then it can be schedule in n slots.
- When we try to add a new job, the sequence being built always contains a gap
- Assume by contradiction that we are unable to add a job of deadline *d*
  - *s* is the next available gap with *s>d*
  - We can show that all scheduled jobs in slot 1 to s-1 have deadlines <= *s-1*, which results in s jobs with deadline <= *s-1* and thus no schedule is feasible

## Use disjoint set to construct feasible sequence




$n_t = max\{k \le t \mid position\ k\ is\ free\}$
Two positions $i$ and $j$ are in the same set if $n_i = n_j$

$F(K)$ is the smallest member of set $K$

## Algorithm

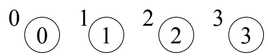
```
Sequence2(d[n+1])
{
   // assuming the profit is sorted
   // in decreasing order
   for (i=0; i<=n; i++) {
      s[i] = 0;
      F[i] = i;
      initialize set i;
   }

   // greedy loop
   for (i=1; i<=n; i++) { // try job i
      k = find(min(n, d[i]));
      m = F[k];
      if (m != 0) {
         s[m] = i;
         l = find(m-1);
         F[k] = F[l];
         merge(k,l);
      }
   }

   // compress the solution
   k=0
   for (i=1; i<=n; i++) {
      if (s[i] >0) {
         k = k+1;
         s[k] = s[i];
      }
   }
   return s;
}
```

Analysis: 2*n find and n merge
Cost: O(n α(2n, n))
Sorting cost for the assumption.

## Example

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $g_i$ | 20 | 15 | 10 | 7 | 5 | 3 |
| $d_i$ | 3 | 1 | 1 | 3 | 1 | 3 |

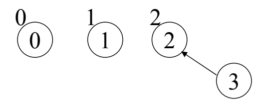
Initialization: $l=\min(6, \max(d_i)) = 3$

F:


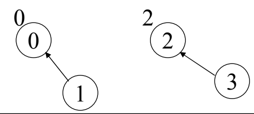

## Example

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $g_i$ | 20 | 15 | 10 | 7 | 5 | 3 |
| $d_i$ | 3 | 1 | 1 | 3 | 1 | 3 |

Try 1: d1=3, assign task 1 to position 3

F:



Try 2: d2=1, assign task 2 to position 1

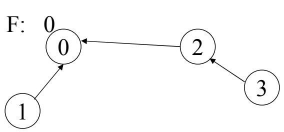
F:



## Example

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $g_i$ | 20 | 15 | 10 | 7 | 5 | 3 |
| $d_i$ | 3 | 1 | 1 | 3 | 1 | 3 |

Try 3: d3=1, no free slot is available

Try 4: d4=3, assign task 4 to position 2

F:



Try 5 and try 6, no free position is available