Runtime of interval Scheduling: n intervals

1. sort intervals by finish time.
2. Schedule the interval with the earliest finish time.
3. Remove all conflicts with the added schedule.
4. Return schedule when there are no tasks left.

---

1. $O(n \log n)$ ← your favorite $O(n \log n)$ sorting algorithm
2. $O(1)$ Pick first element in linked list or element at index i for array
3. $O(n)$ could have a task that conflicts with all other tasks. $O(n)$ time to check them all.

($O(n)$ = 2 and 3)

4. $O(1)$

runtime $= O(n \cdot \log n) + O(n) + O(1) = O(n \log n)$

$O(n \log n + n + 1)$ is $O(n \log n)$

$\lim_{n \to \infty} \frac{n \cdot \log n}{n} = \infty$

Side note

$\lim_{n \to \infty} \frac{2^{100n}}{2^n} = \frac{2^{99n} \cdot 2^n}{2^n} = \infty$

$\therefore 2^n$ is $O(2^{100n})$

2^{100n} is not $O(2^n)$

Schedule to minimize maximum lateness

Input: n intervals/tasks

each interval i is defined by (d_i, t_i)

d_i = deadline for task i

t_i = duration for task i

global start time: s (assume $s = 0$)

Output: Schedule of intervals: each i gets an $s(i)$

$s(i)$ = start time for task i

thus, i is scheduled for $[s(i), s(i) + t_i)$ = $[s(i), f(i))$

define $f(i) = (s(i) + t_i)$

- no two tasks can conflict.

goal: Minimize maximum lateness (for individual tasks)

$\ell_i = \max(f(i) - d_i, 0)$ = lateness for task i

$L = \max_{1 \le i \le n} \ell_i$

Output a schedule that minimizes L

Greedy Algorithms in general: Define a number for each i and sort by that number.

Greedily choose the i's based on this sorted order.

Prove by "Greedy stays ahead" or "Exchange Argument"

↑ last week

↑ Today / Next lecture

Runtime of the algo: n intervals

Algo

1. Sort the tasks by deadline s.t. $d_1 \leq d_2 \leq \cdots \leq d_n$
2. $f = s$ ← global start ($s=0$ is fine)
3. for $i = 1..n$
   - $s(i) \leftarrow f$
   - $f(i) \leftarrow s(i) + t_i$
   - $f \leftarrow f(i)$

   } Schedule all tasks in order of deadline.

   (inside 3)
4. return the resulting schedule

runtime

1. $O(n \log n)$ favorite sort algorithm with $O(n \log n)$ runtime.
2. $O(1)$
3. $O(n)$
   inside 3, $O(1)$
4. $O(1)$

runtime

$$T(n) = O(n \log n) + O(1) + O(n)O(1) + O(1)$$
$$= O(n \log n)$$

if we had

Side note

$$O(n)O(n) = O(n^2)$$

Output of greedy Algo has 0 idle time and no inversions

Idle time: $\max_{1 \le i \le n-1} s(i+1) - f(i)$ ✓

inversion: pair (i,j) s.t. i is scheduled before j and $d_i > d_j$ ✓

Proof of Correctness: The output of the greedy algorithm is optimal.

Lemma 1: If two valid schedules both have 0 idle time and no inversions, then both have the same max lateness (L).

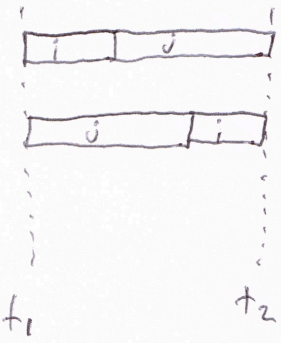
Lemma 2: There exists an optimal schedule with 0 idle time and no inversions.

If we can prove lemmas 1 and 2, then our algo is optimal. There is an optimal solution by Lemma 2 and ours has the same max(L) by Lemma 1.

Proof of Lemma 1: Let S_1 and S_2 be schedules with 0 idle time and no inversions,

- we know all jobs j with $d_j < d_i$ get scheduled before i in both schedules or else ther would be an inversion.
- The order must be the same except when $d_j = d_i$

for any two (or more) tasks i, j with $d_i = d_j = d$, switching the tasks doesn't change the max lateness.



$\leftarrow$ lateness $= \max(t_2 - d, 0)$

$\leftarrow$ lateness $= \max(t_2 - d, 0)$

$t_2 = t_1 + t_i + t_j$

works for arbitrary # of ties.

$$t_2 = t_1 + \sum_i t_i$$

proof of Lemma 1 ▨