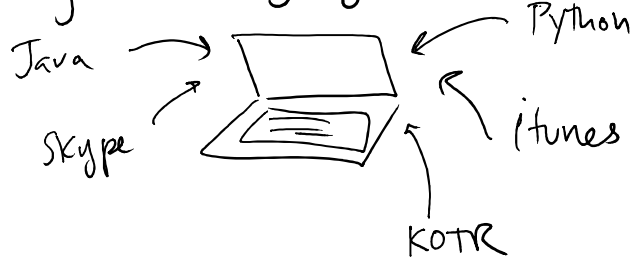
S. KIMMEL

Scheduling

Suppose you are trying to run many applications on a processor



What order is best?

Each job i; $i \in \{1, \ldots, n\}$

- weight (importance) w_i
- time t_i to complete

def: Completion time C_j of job j is sum of times required to complete all jobs run before j, plus t_j.

Scheduling Goal:

Minimize $\sum_{j=1}^{n} w_j C_j$ $\Leftarrow$ "Objective function" A

S. KIMMEL

Greedy algorithm: picks best job now

⇒ What is best? Create function $f(w_i, t_i)$, find job with largest f-value. Put next. Repeat!

Q:
- Come up with at least 2 functions f that seem reasonable. (Want small time / large weight to have a larger f-value)
- Test whether your functions give the right ordering for this example:

job	time	weight
1	3	1
2	5	2

S. KIMMEL

Q: Create a good f:

A: Good jobs have large weight, short time

- $f_1 = \frac{w_i}{t_i}$
- $f_2 = w_i - t_i$

job	weight	time	f_1	f_2
1	1	3	$1/3 = 2/6$	-2
2	2	5	$2/5$	-3

If follow $f_1 \rightarrow$ order $(2, 1)$

If follow $f_2 \rightarrow$ order $(1, 2)$

order	A
1 2	$1 \cdot 3 + 2 \cdot 8 = 19$
2 1	$2 \cdot 5 + 1 \cdot 8 = 18$

f_1 is better!

Maybe you do a few more examples and f_1 always gives you the correct ordering. Then you are ready to try to prove it <u>always</u> gives an optimal ordering.