# More Invariants

```
Sum(A)
    sum ← 0
    for i = 0 to n-1
        sum ← sum + A[i]
    return sum
```

INVARIANT: after $i$ iterations $sum = \sum_{j=0}^{i-1} A[j]$

Basis: $\sum_{j=0}^{-1} A[j] = 0$, init makes $sum \leftarrow 0$

so T after 0 iterations

Ind: Suppose $sum = \sum_{j=0}^{i-1} A[j]$ after $i$ iterations and $i < n$

So $i' = i+1$ $\quad i = i'-1$

and $sum' = sum + A[i]$

$= \sum_{j=0}^{i-1} A[j] + A[i]$

$= \sum_{j=0}^{i} A[j]$

$= \sum_{j=0}^{i'-1} A[j]$

Term: $i = n$,

So $sum = \sum_{j=0}^{n-1} A[j]$

```
Contains(A, key)
    index ← 0
    while index < n and A[index] ≠ key
        index ← index + 1
    return index < n
```

INV: $\underbrace{A[index-1] \neq key}_{\text{not enough!}}$, or $index = 0$

$A[0], \ldots, A[index-1]$ all $\neq$ key

Basis: inv vacuously true ($A[0], \ldots, A[index-1]$ is empty list)

Ind: Suppose $A[0], \ldots, A[index-1]$ all $\neq$ key and $index < n$ and $A[index] \neq key$

then $index' = index + 1$

(need $A[0], \ldots, A[index'-1]$ all $\neq$ key)

So $A[0], \ldots, A[index'-2]$ all $\neq$ key

and $A[index] = A[index'-1] \neq key$

so $A[0], \ldots, A[index'-1]$ all $\neq$ key

Term: if $index < n$, $A[index] = key$ (else loop doesn't stop)

code returns $index < n$, which is T; should have returned T since $A[index] = key$

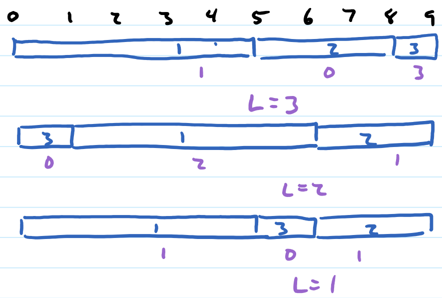
if $index = n$ then $A[0], \ldots, A[\underset{n-1}{index-1}]$ all $\neq$ key, so code should return F

code does return $index < n$, which is F

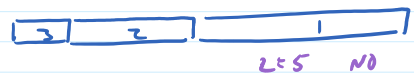
Given requests with lengths $t_1, \dots, t_n$, deadlines $d_1, \dots, d_n$,
find schedule that minimizes maximum lateness.
↳ one request at a time

Ex:

| $i$ | $t_i$ | $d_i$ |
|---|---|---|
| 1 | 5 | 4 |
| 2 | 3 | 8 |
| 3 | 1 | 6 |

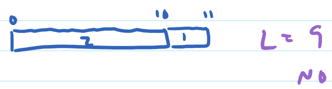



shortest first?




most pressing first?
break ties by shortest


| $i$ | $t_i$ | $d_i$ |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 10 | 10 |



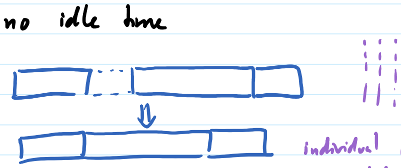

earliest deadline first?


↓

schedule consecutively in order of ↑ deadline (break ties arbitrarily)
note: no idle time, no inversions
↳ $i$ scheduled before $j$ where $d_i > d_j$

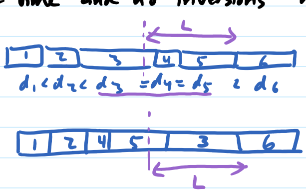
1) There is an optimal schedule with no idle time

$\Theta$ =



individual latenesses ↓ or ↔
max lateness ↓ or ↔

2) All schedules with no idle time and no inversions have same max lateness




scheds differ in order of jobs
w/ = deadline
just change which one is latest,
same max

3) There is a optimal schedule with no inversions, no idle time.
[our schedule has no inversions, no idle,
some opt has no inv, no idle,
ours and that opt have same max late
ours is also opt]

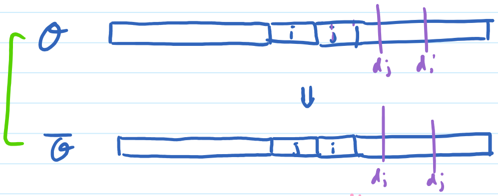
↓

Can find opt sched $\Theta$ with no idle (1)

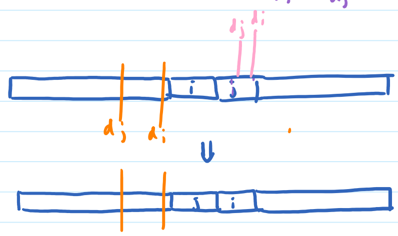
Can find opt sched $O$ with no idle (1)

If $O$ has no inversions then ü

Else find inversion between adj in schedule




deadlines both after j finishes




lateness of $i = \ell_i = f(i) - d_i$

$j = \ell_j = f(j) - d_j$

b/c $d_j < d_i$

lateness of $i = \bar{\ell}_i = \bar{f}(i) - d_i < f(j) - d_j = \ell_j$

$= f(j) - d_i = \ell_i + t_j$

$\ell_i + t_j = f(j) - d_i$

$< f(j) - d_j = \ell_j$

$j = \bar{\ell}_j = \ell_j - t_i$

$\max(\bar{\ell}_i, \bar{\ell}_j) = \max(\ell_i + t_j, \ell_j - t_i) \le \max(\ell_j, \ell_j - t_i) = \ell_j \le \max(\ell_i, \ell_j)$

$\ell_i + t_j \le \ell_j$

$\max(x, z) \le \max(y, z)$

$x \le y$

**Optimal Caching**

Memory Hierarchy : expensive

- registers — fast, small
- compiler
- L1 cache
- L2 cache — CPU
- CPU managed
- L3 cache
- main memory
- flash storage — OS
- disk storage
- cheap — tape storage — slow, large

caching : keep soon-to-be-accessed data in fast memory

spatial locality

temporal locality

Optimal caching : given cache size $k$, sequence of requests $d_1, \ldots, d_m$ for data items, find eviction schedule to minimize cache misses

Ex: $k=2$ , initial cache $= 1, 2$ requests $1,2,3,2,1,3,4,2,3,1$