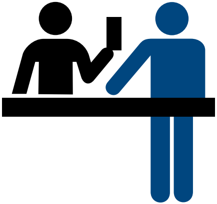
Algorithms for Programming Contests - Week 7

Tobias Meggendorfer, Philipp Meyer,
Christian Müller, Gregor Schwarz
conpra@in.tum.de

28.11.2018

Change making

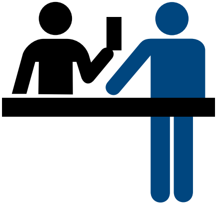
2\$ | 1\$ | 25¢ | 10¢ | 5¢ | 1¢



How to give change back
with the minimum number of coins?

Change making

2$ 1$ 25¢ 10¢ 5¢ 1¢

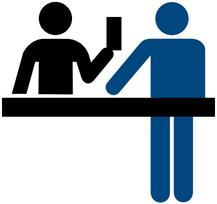


How to give change back
with the minimum number of coins?

Be **greedy**: go for the largest coins first!

Change making

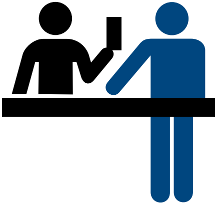
2\$ | 1\$ | 25¢ | 10¢ | 5¢ | 1¢



$$
\begin{aligned}
5.82\$ - 2 \times 2\$ &= 1.82\$ \\
1.82\$ - 1 \times 1\$ &= 0.82\$ \\
0.82\$ - 3 \times 25¢ &= 0.07\$ \\
0.07\$ - 1 \times 5¢ &= 0.02\$ \\
0.02\$ - 2 \times 1¢ &= 0.00\$
\end{aligned}
$$

Change making

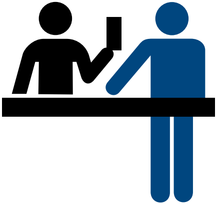
2\$ 1\$ 25¢ 10¢ 5¢ 1¢



Approach still works if we introduce 20¢ ?

Change making

2\$ 1\$ 25¢ 10¢ 5¢ 1¢



No, for 40¢ it returns 25¢ + 10¢ + 5¢ instead of 2 × 20¢

Change making: greedy approach

procedure GREEDY-CHANGE-MAKING($c_1, \ldots, c_n, m$)
  sort $c_1, \ldots, c_n$ in descending order
  $S \leftarrow [\,]$
  $i \leftarrow 1$, $rem \leftarrow m$
  while $i \leq n$ **and** $rem > 0$ **do**
    if $c_i \leq rem$ **then**
      $rem \leftarrow rem - c_i$
      add c_i **to** S
    else
      $i \leftarrow i + 1$
  if $rem = 0$ **then return** S
  else return impossible

Change making: greedy approach

procedure GREEDY-CHANGE-MAKING($c_1, \ldots, c_n, m$)
  sort $c_1, \ldots, c_n$ in descending order
  $S \leftarrow [\,]$
  $i \leftarrow 1$, $rem \leftarrow m$
  while $i \leq n$ **and** $rem > 0$ **do**
    if $c_i \leq rem$ **then**
      $rem \leftarrow rem - c_i$
      add c_i **to** S
    else
      $i \leftarrow i + 1$
  if $rem = 0$ **then return** S
  else return impossible

GREEDY-CHANGE-MAKING is optimal for \$ (CAD) and € (EUR)

Change making: greedy approach

procedure GREEDY-CHANGE-MAKING$(c_1, \ldots, c_n, m)$
  sort $c_1, \ldots, c_n$ in descending order
  $S \leftarrow [\,]$
  $i \leftarrow 1$, $rem \leftarrow m$
  while $i \leq n$ **and** $rem > 0$ **do**
    if $c_i \leq rem$ **then**
      $rem \leftarrow rem - c_i$
      add c_i **to** S
    else
      $i \leftarrow i + 1$
  if $rem = 0$ **then return** S
  else return impossible

The solution of GREEDY-CHANGE-MAKING can be arbitrarily bad

Let $n > 2$. On input $(c_1 = n + 2, c_2 = n + 1, c_3 = n, c_4 = 1, m = 2n + 1)$, GREEDY-CHANGE-MAKING returns n coins instead of 2 coins $\square$

Change making: greedy approach

procedure GREEDY-CHANGE-MAKING($c_1, \ldots, c_n, m$)
  sort $c_1, \ldots, c_n$ in descending order
  $S \leftarrow [\,]$
  $i \leftarrow 1$, $rem \leftarrow m$
  while $i \leq n$ **and** $rem > 0$ **do**
    if $c_i \leq rem$ **then**
      $rem \leftarrow rem - c_i$
      add c_i **to** S
    else
      $i \leftarrow i + 1$
  if $rem = 0$ **then return** S
  else return impossible

Finding an optimal solution is NP-hard for arbitrary currencies

Greedy algorithms

- Paradigm for solving optimization problems
- Make local choices, never global
- Do not reconsider choices

– Often non optimal

+ Can be good heuristics
+ Can be good approximations
+ Simple
+ Fast

Greedy algorithms: general template

```
procedure GREEDY(candidates)
    S ← ∅
    while |candidates| > 0 and ¬solution(S) do
        c ← select(candidates)
        remove c from candidates
        if feasible(S, c) then
            add c to S
    if solution(S) then
        return S
    else
        return impossible
```

Greedy algorithms: general template

procedure GREEDY(*candidates*)
    $S \leftarrow \emptyset$
    while $|candidates| > 0$ **and** $\neg$**solution**(S) **do**
        $c \leftarrow$ **select**(*candidates*)
        remove c **from** *candidates*
        if feasible(S, c) **then**
            add c **to** S
    if solution(S) **then**
        return S
    else
        return impossible

Kruskall algorithm

candidates: edges
select: smallest edge
feasible: connects two connected components?
solution: contains $|V| - 1$ edges?

Greedy algorithms: general template

procedure GREEDY(*candidates*)
    $S \leftarrow \emptyset$
    while $|candidates| > 0$ **and** $\neg$**solution**(S) **do**
        $c \leftarrow$ **select**(*candidates*)
        remove c **from** *candidates*
        if feasible(S, c) **then**
            add c **to** S
    if solution(S) **then**
        return S
    else
        return impossible

Prim algorithm

candidates: edges
select: smallest edge with an endpoint in explored nodes
feasible: has no cycle?
solution: covers every node?

Greedy algorithms: general template

procedure GREEDY(*candidates*)
    $S \leftarrow \emptyset$
    while $|candidates| > 0$ **and** $\neg$ **solution**(S) **do**
        $c \leftarrow$ **select**(*candidates*)
        remove c **from** *candidates*
        if feasible(S, c) **then**
            add c **to** S
    if solution(S) **then**
        return S
    else
        return impossible

Change making

candidates: coins
select: largest coin smaller or equal to remaining amount
feasible: —
solution: sums up to the amount?

Approximation algorithms

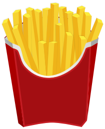
- Approximate optimal solution up to some factor
- Provable guarantees on such factors
- Way to circumvent NP-hardness
- Can be designed as efficient greedy algorithms

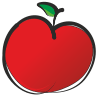
0/1 Knapsack problem

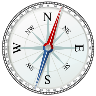
Given:
- backpack of capacity $W \in \mathbb{N}_{>0}$
- n objects of value $v_1, \ldots, v_n \in \mathbb{N}$ and weight $w_1, \ldots, w_n \in [1, W]$

Compute: subset of objects of maximal value among subsets of weight at most W

0/1 Knapsack problem





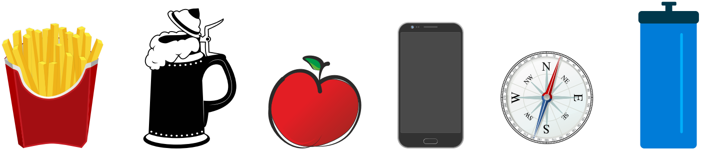


Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

What to bring in the backpack?

Capacity: 900g

0/1 Knapsack problem

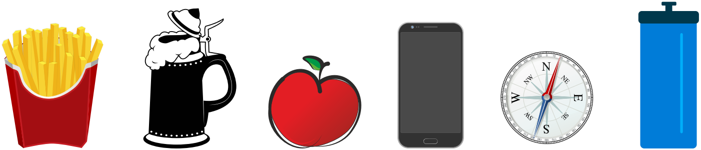


Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

Value: 32 (870g)

Capacity: 900g

0/1 Knapsack problem

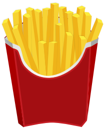
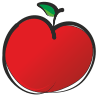
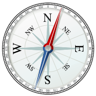


Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

Value: 70 (900g)

Capacity: 900g

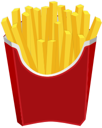
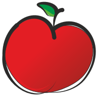
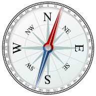
0/1 Knapsack problem

						
Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

Value: 75 (890g)

Capacity: 900g

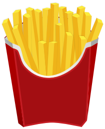
0/1 Knapsack problem

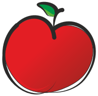
						
Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

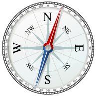
Capacity: 900g

Greedy way to obtain solution?

0/1 Knapsack problem





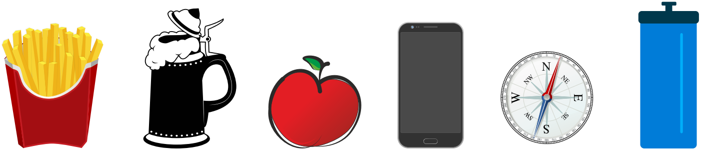


Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g

Capacity: 900g

Sort in desc. order w.r.t. v_i/w_i...

0/1 Knapsack problem

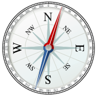
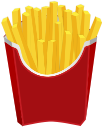


Value:	10	15	5	50	7	20
Weight:	150g	540g	100g	200g	70g	700g
Ratio:	1/15	1/36	1/20	1/4	1/10	1/35

Capacity: 900g

Sort in desc. order w.r.t. v_i/w_i...

0/1 Knapsack problem

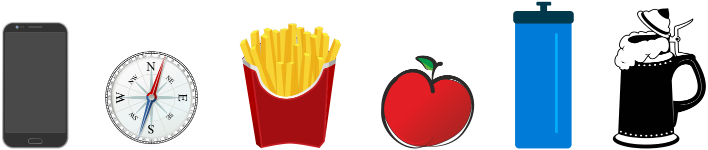
     

Value:	50	7	10	5	20	15
Weight:	200g	70g	150g	100g	700g	540g
Ratio:	$1/4$	$1/10$	$1/15$	$1/20$	$1/35$	$1/36$

Sort in desc. order w.r.t. v_i/w_i...

Capacity: 900g

0/1 Knapsack problem

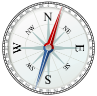


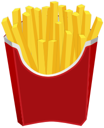
Value:	50	7	10	5	20	15
Weight:	200g	70g	150g	100g	700g	540g
Ratio:	1/4	1/10	1/15	1/20	1/35	1/36

Value: 72 (520g)

Capacity: 900g

0/1 Knapsack problem




Value:	50	7	10	5	20	15
Weight:	200g	70g	150g	100g	700g	540g
Ratio:	1/4	1/10	1/15	1/20	1/35	1/36

Capacity: 900g

Not optimal, but by how much?

0/1 Knapsack problem: greedy approach

procedure $\textsc{Greedy-Knapsack-Naive}(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  return $value$

0/1 Knapsack problem: greedy approach

procedure $\textsc{Greedy-Knapsack-Naive}(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $\mathit{value}, \mathit{weight} \leftarrow 0$
  $i \leftarrow 1$
  while $\mathit{weight} + w_i \leq W$ **and** $i \leq n$ **do**
    $\mathit{value} \leftarrow \mathit{value} + v_i$
    $\mathit{weight} \leftarrow \mathit{weight} + w_i$
    $i \leftarrow i + 1$
  return value

The solution of $\textsc{Greedy-Knapsack-Naive}$ can be arbitrarily bad

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK-NAIVE$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  return $value$

The solution of GREEDY-KNAPSACK-NAIVE can be arbitrarily bad

Let $W > 2$. On input $(v_1 = 2, w_1 = 1), (v_2 = W, w_2 = W)$,
GREEDY-KNAPSACK-NAIVE returns 2 while the optimal value is W $\square$

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK-FRAC$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  return $value + \frac{(W - weight)}{w_i} \cdot v_i$

0/1 Knapsack problem: greedy approach

procedure $\textsc{Greedy-Knapsack-Frac}(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  return $value + \frac{(W - weight)}{w_i} \cdot v_i$

$\textsc{Greedy-Knapsack-Frac}$ is optimal if objects can be taken partially by a factor $0 \leq \alpha \leq 1$

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK-FRAC$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  return $value + \frac{(W - weight)}{w_i} \cdot v_i$

GREEDY-KNAPSACK-FRAC is optimal if objects can be taken partially by a factor $0 \leq \alpha \leq 1$

Relatively straightforward proof

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  if $i \leq n$ **then return** $\max(value, v_i)$
  else return $value$

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK$(W, (v_1, w_1), \dots, (v_n, w_n))$
  sort $(v_1, w_1), \dots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $\mathit{value}, \mathit{weight} \leftarrow 0$
  $i \leftarrow 1$
  while $\mathit{weight} + w_i \leq W$ **and** $i \leq n$ **do**
    $\mathit{value} \leftarrow \mathit{value} + v_i$
    $\mathit{weight} \leftarrow \mathit{weight} + w_i$
    $i \leftarrow i + 1$
  if $i \leq n$ **then return** $\max(\mathit{value}, v_i)$
  else return value

The solution of GREEDY-KNAPSACK is at least $\frac{1}{2}$ of the optimal solution

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $\mathit{value}, \mathit{weight} \leftarrow 0$
  $i \leftarrow 1$
  while $\mathit{weight} + w_i \leq W$ **and** $i \leq n$ **do**
    $\mathit{value} \leftarrow \mathit{value} + v_i$
    $\mathit{weight} \leftarrow \mathit{weight} + w_i$
    $i \leftarrow i + 1$
  if $i \leq n$ **then return** $\max(\mathit{value}, v_i)$
  else return value

The solution of GREEDY-KNAPSACK is at least $\frac{1}{2}$ of the optimal solution

$$\underbrace{(v_1 + \ldots + v_{i-1})}_{\mathit{value}} + v_i \geq \mathit{opt}_{\text{frac}} \geq \mathit{opt} \implies \max(\mathit{value}, v_i) \geq \mathit{opt}/2 \quad \square$$

0/1 Knapsack problem: greedy approach

```
procedure GREEDY-KNAPSACK(W, (v_1, w_1), ..., (v_n, w_n))
    sort (v_1, w_1), ..., (v_n, w_n) in descending order w.r.t. v_i/w_i
    value, weight ← 0
    i ← 1
    while weight + w_i ≤ W and i ≤ n do
        value ← value + v_i
        weight ← weight + w_i
        i ← i + 1
    if i ≤ n then return max(value, v_i)
    else return value
```

Worst-case time complexity:

by sorting:	$O(n \cdot \log n)$
using recursion and linear time median:	$O(n)$

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $value, weight \leftarrow 0$
  $i \leftarrow 1$
  while $weight + w_i \leq W$ **and** $i \leq n$ **do**
    $value \leftarrow value + v_i$
    $weight \leftarrow weight + w_i$
    $i \leftarrow i + 1$
  if $i \leq n$ **then return** $\max(value, v_i)$
  else return $value$

In general, computing an optimal solution is NP-hard

0/1 Knapsack problem: greedy approach

procedure GREEDY-KNAPSACK$(W, (v_1, w_1), \ldots, (v_n, w_n))$
  sort $(v_1, w_1), \ldots, (v_n, w_n)$ in descending order w.r.t. v_i/w_i
  $\mathit{value}, \mathit{weight} \leftarrow 0$
  $i \leftarrow 1$
  while $\mathit{weight} + w_i \leq W$ **and** $i \leq n$ **do**
    $\mathit{value} \leftarrow \mathit{value} + v_i$
    $\mathit{weight} \leftarrow \mathit{weight} + w_i$
    $i \leftarrow i + 1$
  if $i \leq n$ **then return** $\max(\mathit{value}, v_i)$
  else return value

However, greedy approach is optimal when all weights are equal

Job scheduling

Given:
- n jobs of duration $d_1, \ldots, d_n \in \mathbb{N}_{>0}$
- m processors

Compute: smallest amount of time to complete all jobs

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

How to schedule the jobs on two processors?

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

Time: 31 ms.

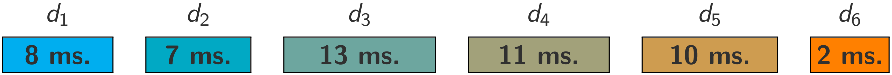
Processor 1:	2 ms.	8 ms.	11 ms.	10 ms.
Processor 2:	7 ms.	13 ms.		

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

Greedy way to obtain solution?

Job scheduling



Assign next job to less busy processor...

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

Time: 29 ms.

Processor 1:	8 ms.	11 ms.	10 ms.
Processor 2:	7 ms.	13 ms.	2 ms.

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

Assign longest job to less busy processor...

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

Time: 28 ms.

Processor 1:	13 ms.	8 ms.	7 ms.
Processor 2:	11 ms.	10 ms.	2 ms.

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

None are optimal!

Job scheduling

d_1	d_2	d_3	d_4	d_5	d_6
8 ms.	**7 ms.**	**13 ms.**	**11 ms.**	**10 ms.**	**2 ms.**

<u>Time: 26 ms.</u>

Processor 1:	10 ms.	8 ms.	7 ms.
Processor 2:	13 ms.	11 ms.	2 ms.

Job scheduling: greedy approach

procedure $\textsc{scheduling-greedy}(d_1, \ldots, d_n, m)$
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$

    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

First observe that $opt \geq \max(d_1, \ldots, d_n)$ and $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY$(d_1, \ldots, d_n, m)$
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$

    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

- $opt \geq \max(d_1, \ldots, d_n)$
- $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

First observe that $opt \geq \max(d_1, \ldots, d_n)$ and $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY$(d_1, \ldots, d_n, m)$
$P_1, \ldots, P_m \leftarrow 0$
$time \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**
find j such that P_j is minimal
$P_j \leftarrow P_j + d_i$
$time \leftarrow \max(time, P_j)$
return $time$

- $opt \geq \max(d_1, \ldots, d_n)$
- $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

Let i^*, j^* be s.t. $P_{j^*} = time$ and i^* is the last job assigned to processor j^*

Let P'_k be the load of processor k just before job i^* is assigned

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY$(d_1, \ldots, d_n, m)$
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $opt \geq \max(d_1, \ldots, d_n)$
- $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

$$m \cdot P'_{j*} \;\leq\; \sum_{1 \leq j \leq m} P'_j \;=\; \sum_{1 \leq i < i*} d_i \;\leq\; \sum_{1 \leq i \leq n} d_i \;\leq\; m \cdot opt$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY$(d_1, \ldots, d_n, m)$
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$

  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $opt \geq \max(d_1, \ldots, d_n)$
- $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$
- $opt \geq P'_{j*}$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

$$m \cdot P'_{j*} \;\leq\; \sum_{1 \leq j \leq m} P'_j \;=\; \sum_{1 \leq i < i*} d_i \;\leq\; \sum_{1 \leq i \leq n} d_i \;\leq\; m \cdot opt$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY$(d_1, \ldots, d_n, m)$
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$

    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

- $opt \geq \max(d_1, \ldots, d_n)$
- $opt \geq \frac{1}{m}(d_1 + \ldots + d_n)$
- $opt \geq P'_{j^*}$

The solution of SCHEDULING-GREEDY is at most twice the optimal one

$$time = P_{j^*} = P'_{j^*} + d_{i^*} \leq opt + opt = 2 \cdot opt \quad \square$$

Job scheduling: greedy approach

```
procedure SCHEDULING-GREEDY-ORD(d_1, ..., d_n, m)
    P_1, ..., P_m ← 0
    time ← 0
    sort d_1, ..., d_n in descending order
    for i ← 1 to n do
        find j such that P_j is minimal
        P_j ← P_j + d_i
        time ← max(time, P_j)
    return time
```

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$
    sort $d_1, \ldots, d_n$ in descending order
    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Let i^*, j^* be s.t. $P_{j^*} = time$ and i^* is the last job assigned to processor j^*

From previous proof: $P_{j^*} \leq opt + d_{i^*}$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
$P_1, \ldots, P_m \leftarrow 0$
$time \leftarrow 0$
sort $d_1, \ldots, d_n$ in descending order
for $i \leftarrow 1$ **to** n **do**
find j such that P_j is minimal
$P_j \leftarrow P_j + d_i$
$time \leftarrow \max(time, P_j)$
return $time$

- $P_{j^*} \leq opt + d_{i^*}$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Let i^*, j^* be s.t. $P_{j^*} = time$ and i^* is the last job assigned to processor j^*

From previous proof: $P_{j^*} \leq opt + d_{i^*}$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

If $i^* \leq m$, then solution is optimal. Thus, assume

$$i^* > m$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

If $i^* \leq m$, then solution is optimal. Thus, assume

$$i^* > m$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$
    sort $d_1, \ldots, d_n$ in descending order
    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Since $i^* > m$ and jobs are scheduled in desc. order: $d_m \geq d_{m+1} \geq d_{i^*}$.

Thus, $d_{i^*} \leq (d_m + d_{m+1})/2$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
- $P_1, \ldots, P_m \leftarrow 0$
- $time \leftarrow 0$
- **sort** $d_1, \ldots, d_n$ in descending order
- **for** $i \leftarrow 1$ **to** n **do**
  - **find** j such that P_j is minimal
  - $P_j \leftarrow P_j + d_i$
  - $time \leftarrow \max(time, P_j)$
- **return** $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Since $i^* > m$ and jobs are scheduled in desc. order: $d_m \geq d_{m+1} \geq d_{i^*}$.

Thus, $d_{i^*} \leq (d_m + d_{m+1})/2$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Since $n \geq i^* > m$, two jobs $k, k' \in [1, m+1]$ are assigned to the same processor. Thus:

$$d_m + d_{m+1} \leq d_k + d_{k'} \leq opt$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$
- $d_m + d_{m+1} \leq opt$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Since $n \geq i^* > m$, two jobs $k, k' \in [1, m+1]$ are assigned to the same processor. Thus:

$$d_m + d_{m+1} \leq d_k + d_{k'} \leq opt$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
    $P_1, \ldots, P_m \leftarrow 0$
    $time \leftarrow 0$
    sort $d_1, \ldots, d_n$ in descending order
    for $i \leftarrow 1$ **to** n **do**
        find j such that P_j is minimal
        $P_j \leftarrow P_j + d_i$
        $time \leftarrow \max(time, P_j)$
    return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$
- $d_m + d_{m+1} \leq opt$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Therefore:

$$time = P_{j^*} \leq opt + d_{i^*} \leq opt + \frac{d_m + d_{m+1}}{2} \leq opt + \frac{opt}{2} = \frac{3}{2} \cdot opt \quad \square$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD$(d_1, \ldots, d_n, m)$
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$
- $d_m + d_{m+1} \leq opt$

The solution of SCHEDULING-GREEDY-ORD is at most $\frac{3}{2} \cdot opt$

Therefore:

$$time = P_{j^*} \leq opt + d_{i^*} \leq opt + \frac{d_m + d_{m+1}}{2} \leq opt + \frac{opt}{2} = \frac{3}{2} \cdot opt \quad \square$$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$
- $d_m + d_{m+1} \leq opt$

Worst-case time complexity when implemented with min-heap:

SCHEDULING-GREEDY-ORD:	$O(m + n \cdot \log m + n \cdot \log n)$
SCHEDULING-GREEDY:	$O(m + n \cdot \log m)$

Job scheduling: greedy approach

procedure SCHEDULING-GREEDY-ORD($d_1, \ldots, d_n, m$)
  $P_1, \ldots, P_m \leftarrow 0$
  $time \leftarrow 0$
  sort $d_1, \ldots, d_n$ in descending order
  for $i \leftarrow 1$ **to** n **do**
    find j such that P_j is minimal
    $P_j \leftarrow P_j + d_i$
    $time \leftarrow \max(time, P_j)$
  return $time$

- $P_{j^*} \leq opt + d_{i^*}$
- $i^* > m$
- $d_{i^*} \leq (d_m + d_{m+1})/2$
- $d_m + d_{m+1} \leq opt$

Computing an optimal solution is NP-hard, even for two processors