Delay Tolerant Linear Partition

Nelson Jaimes

Rana Alsaadi

Team Introduction

Nelson Jaimes

- Specializing in Computer Science
- Neuroscience background
- Likes art

Rana Alsaadi

- Specializing in Computer Security
- Business Informatics background
- Likes cats

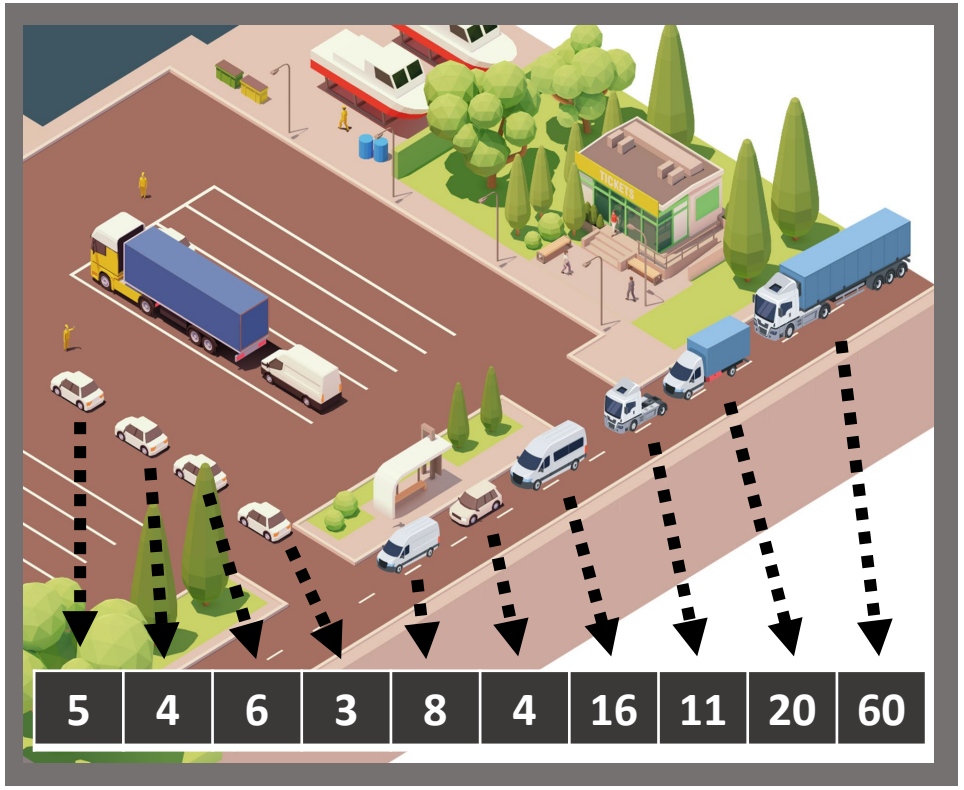
Topics Covered

- Problem Definition
- Problem Importance
- Solution
- Empirical Results

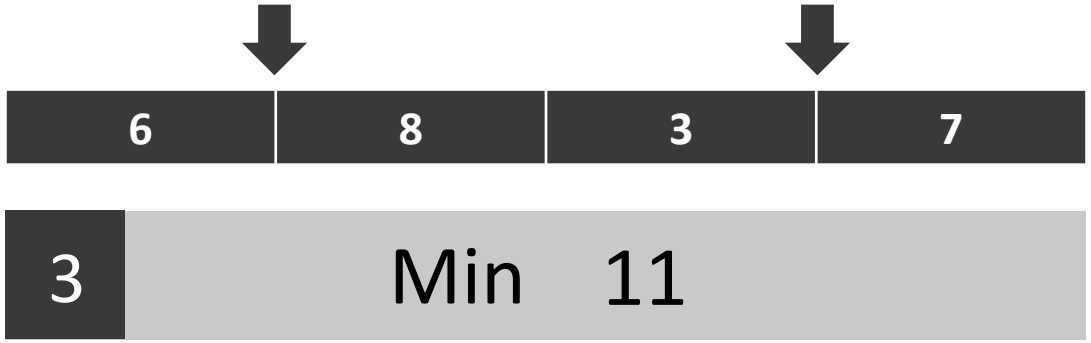
Topics Covered

- Problem Definition
  - Introduction
  - Overview
  - Formal Definition
- Problem Importance
- Solution
- Empirical Results

Problem Introduction



Problem Definition: Overview



- K contiguous subdivisions
- n numbers of various sizes
- Find minimum subdivision weight that can fit all objects in sequence

Formal Problem Definition

Let $A[1:n]$ be a real array, and let k be an integer, $1 \le k \le n$.

A ***linear k-partition* of *A***

a sequence of subarrays of the form

[1:x1],[x1+1:x2],[x2+1:x3],…,[x(k-1)+1:n], for some x1< x2 <… < x(k−1).

From this, we calculate the sums of each of k subarrays.

The ***weight*** of this linear k-partition is the maximum of those sums

Goal: minimize the weight of the linear k-partition of the given array in O(nk).

http://notexponential.com/382/minimizing-weight-of-a-linear-partition

Topics Covered

- Problem Definition
- Problem Importance
- Solution
- Empirical Results

Topics Covered

- Problem Definition
- Problem Importance
- Solution
- Empirical Results

Problem Importance

- Finds minimum container capacity given known weights
- Has similarities to the bin packing problem
- Examples
  - Loading trucks with weight capacity constraints [1]
  - Video-on-demand [2]
  - Multiprocessor scheduling [3]
  - Job scheduling [4]
  - Cloud computing [5]

Topics Covered

- Problem Definition
- Problem Importance
- Solution
- Empirical Results

Topics Covered

- Problem Definition
- Problem Importance
- Solution
  - Overview
  - Recursive Definition
  - Base Case and Next Case
  - Optimality
  - Sample Execution
  - Insights
- Empirical Results

Solution: Overview

- Use Dynamic Programming
- Populate array with the minimum weights for each number of partitions and array lengths.
- Reuse those minimum weights of k-1 partitions to calculate partition k’s weights
- The last item calculated is the result of interest.

Solution: Recursive Definition

- Let S[i,j] represent the minimum weight of the linear i-partition of array[1:j]
- S[i,j] = min_for_all x_from_1_to_ j (max(S[i-1,x] , sum(array[x+1:j]))

http://notexponential.com/382/minimizing-weight-of-a-linear-partition

Solution: Base Case and Next Case

Base Case

- S[1,j] = sum(array[1:j])

Next Case

- Let S[i-1,j] = minimum weight of the linear (i-1) partition of array[1:j]
- Increase i from 2 to k
- S[i,j] = min_for_all x_from_1_to_ j (max(S[i-1,x] , sum(array[x+1:j]))

Stored in S[n,k] array

- S[k,n] is solution

http://notexponential.com/382/minimizing-weight-of-a-linear-partition

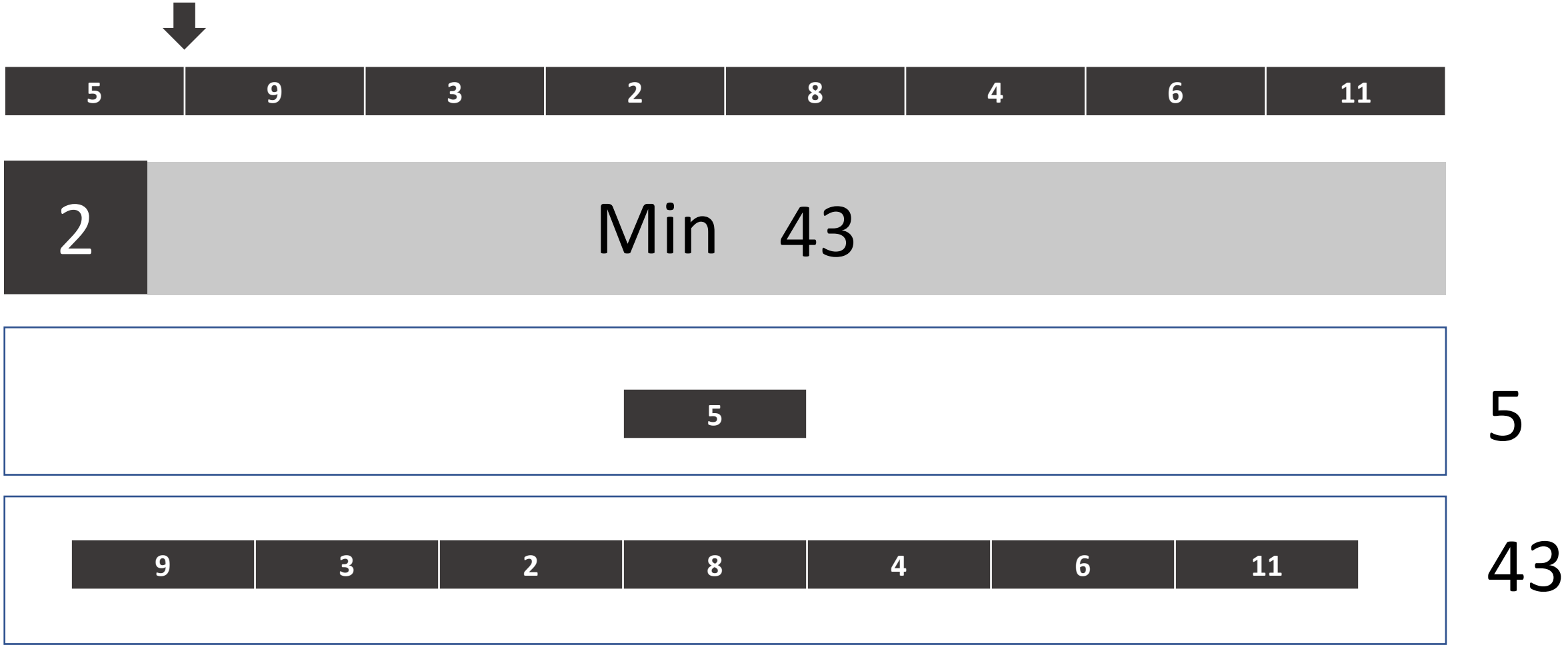
Solution Optimality

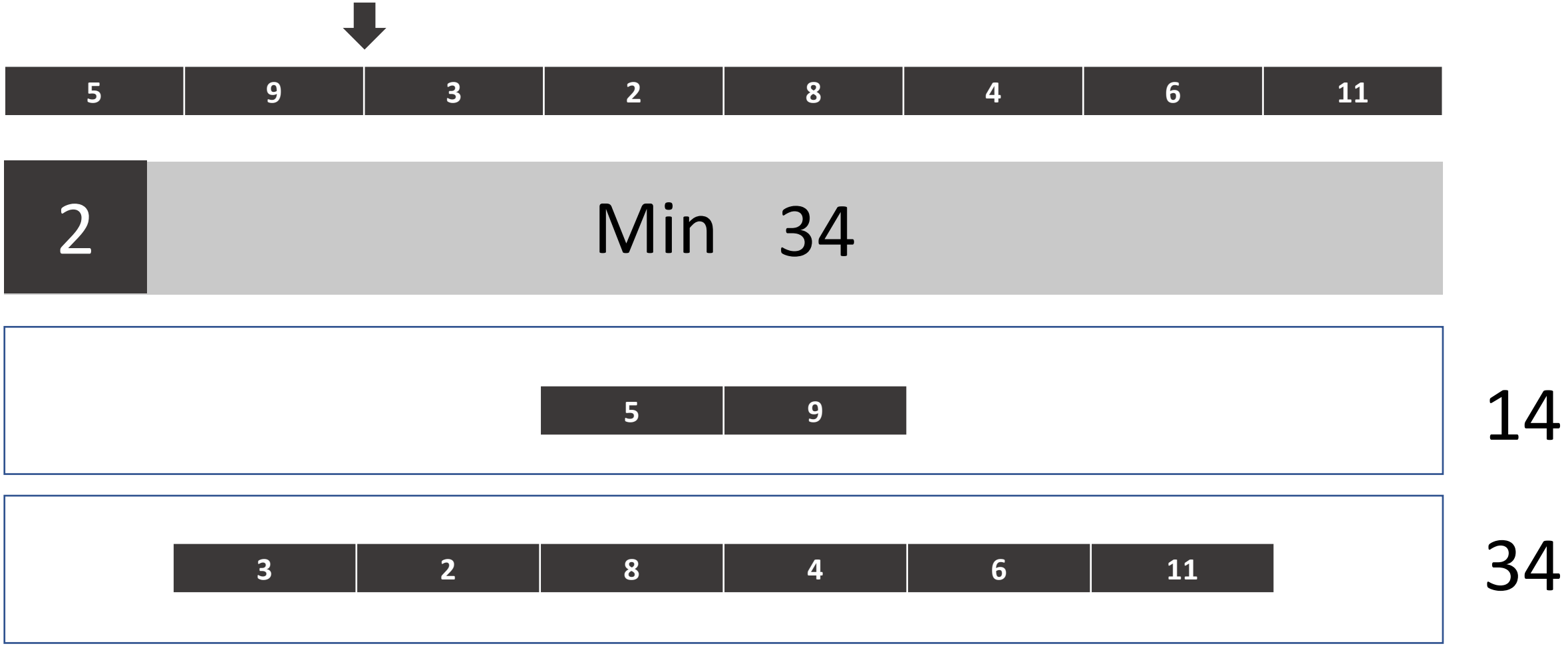
Optimal substructure

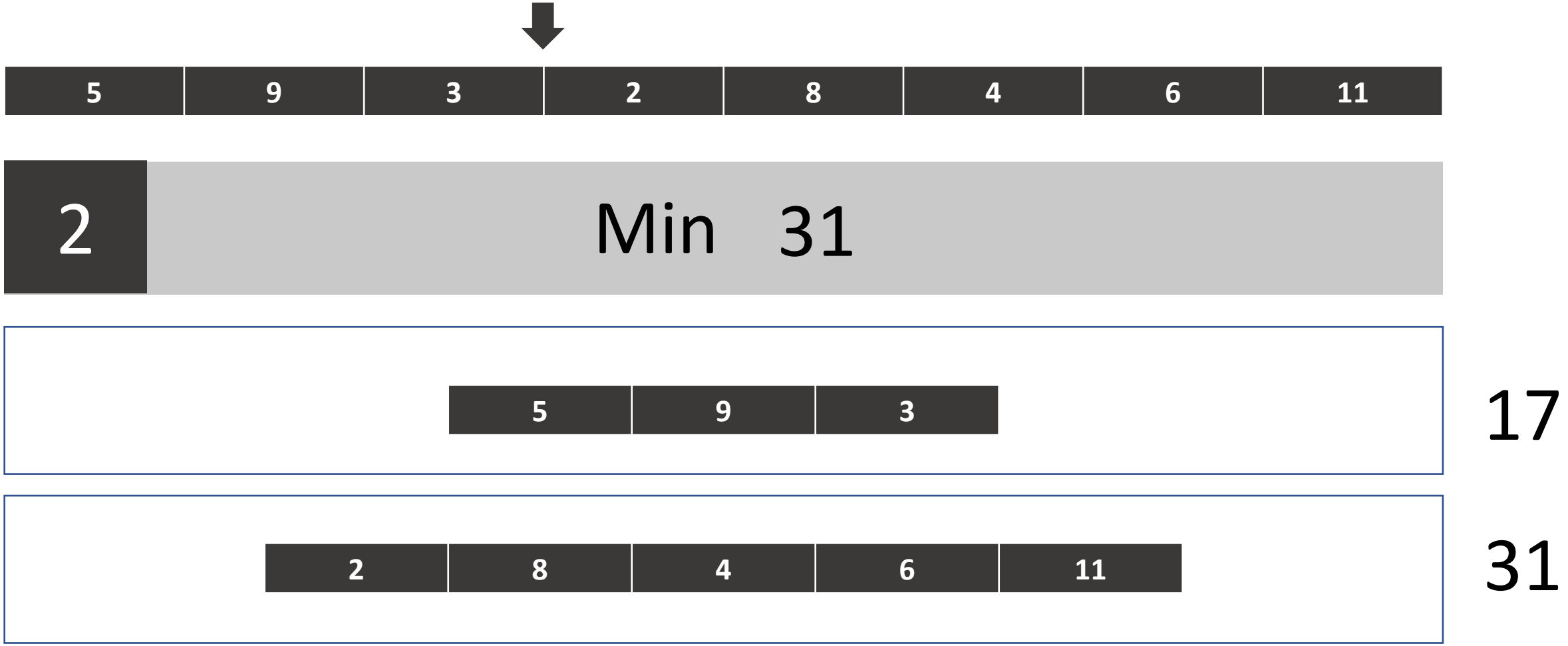
- Weights are computed by finding the min of all relevant possibilities.
- If for any index,
  - a weight smaller than the one selected exists
  - it would have been selected instead
- Array S[i,j] contains the optimal weights, for i partitions and length j

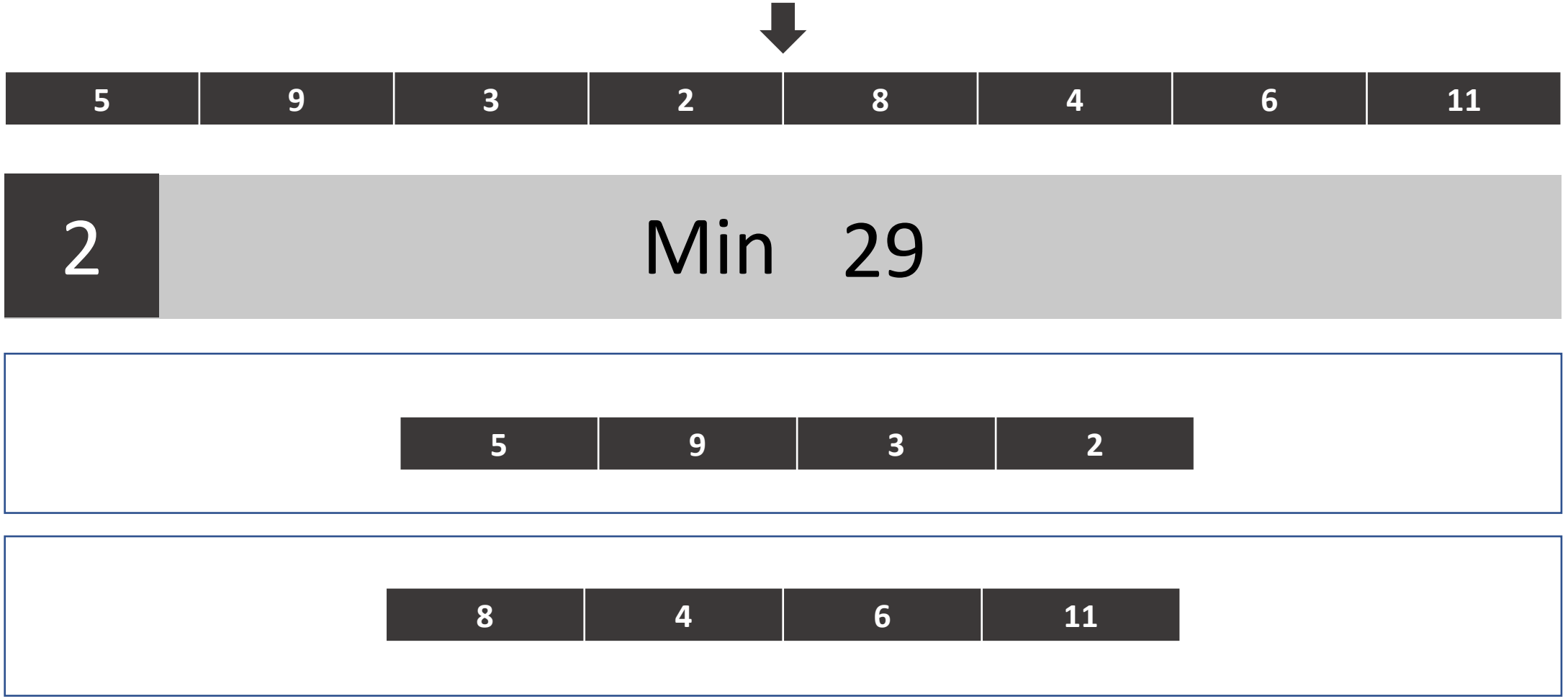
http://notexponential.com/382/minimizing-weight-of-a-linear-partition

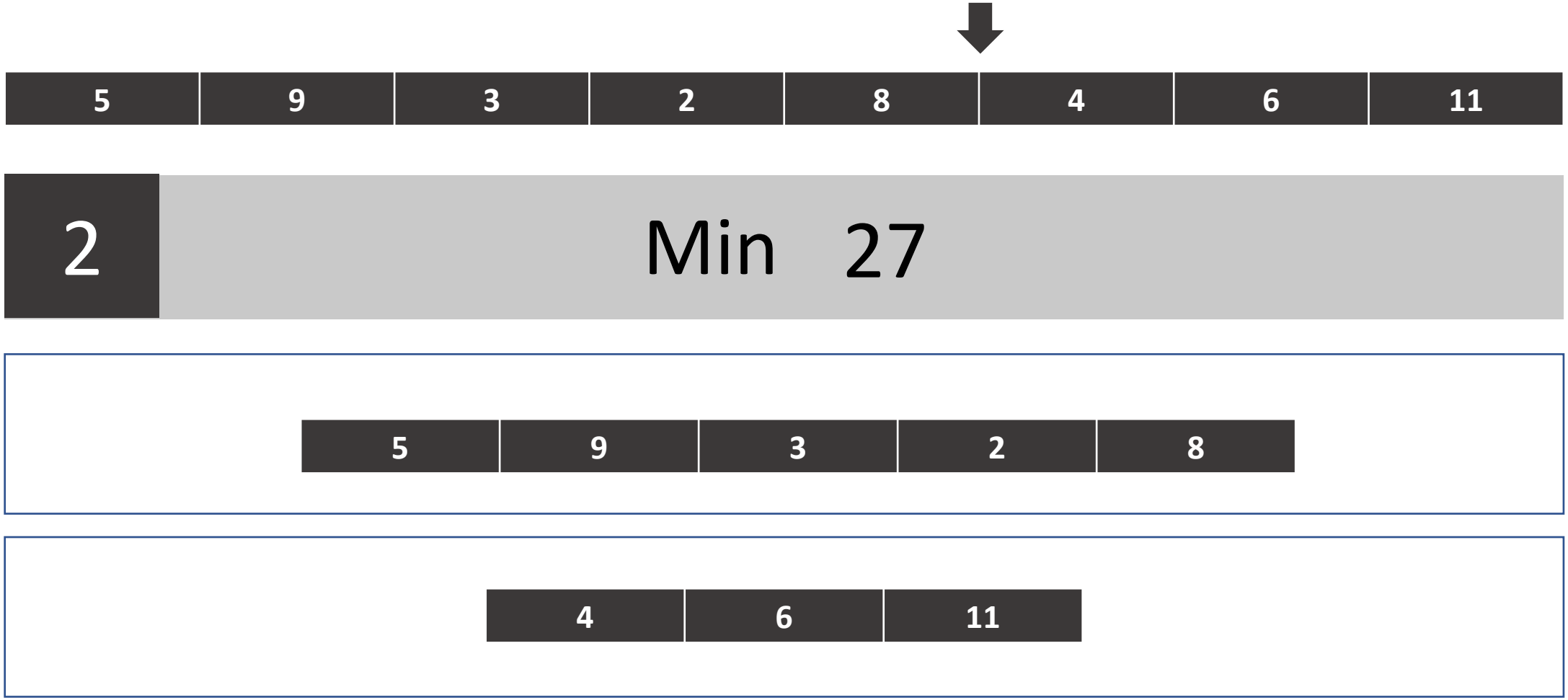
Solution: Sample Execution

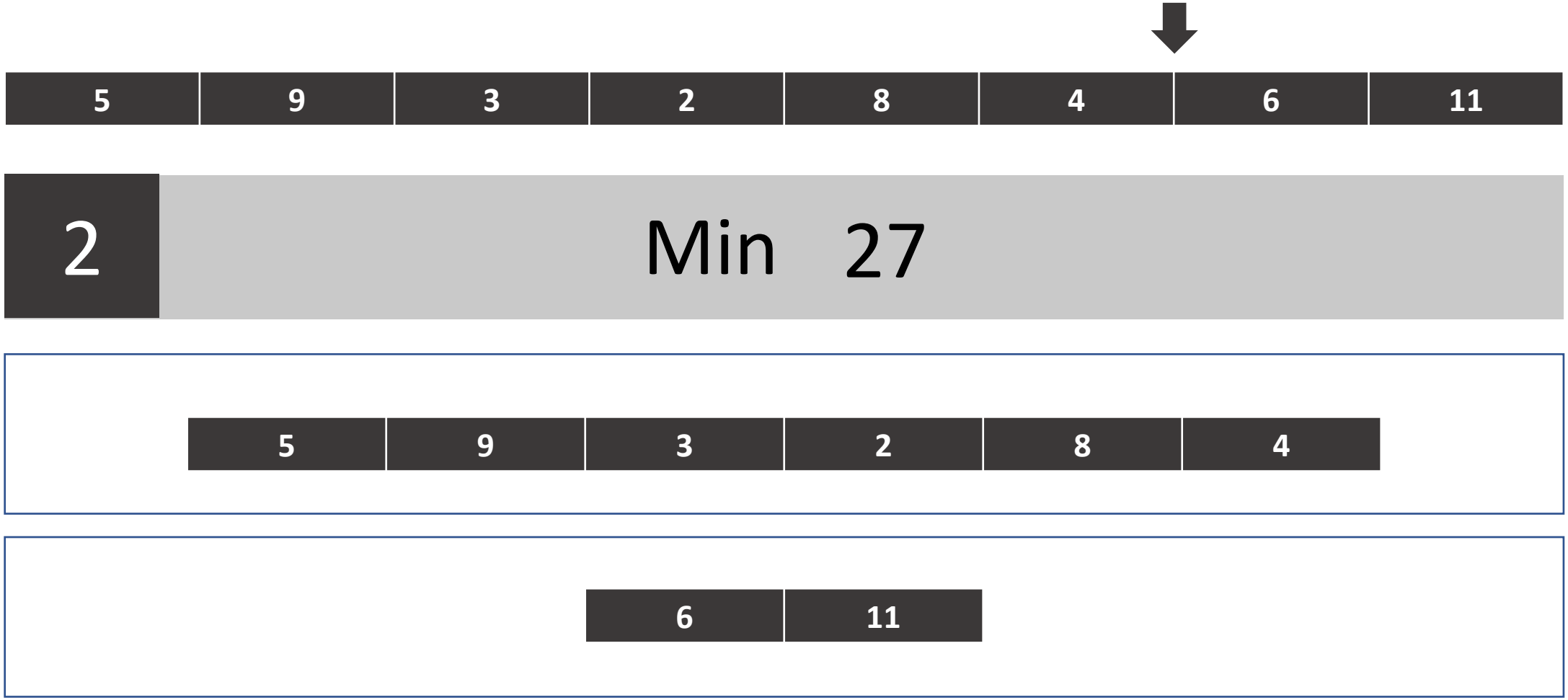

5
9
3
2
8
4
6
11
2
Min 43
5
5
9
3
2
8
4
6
11
43

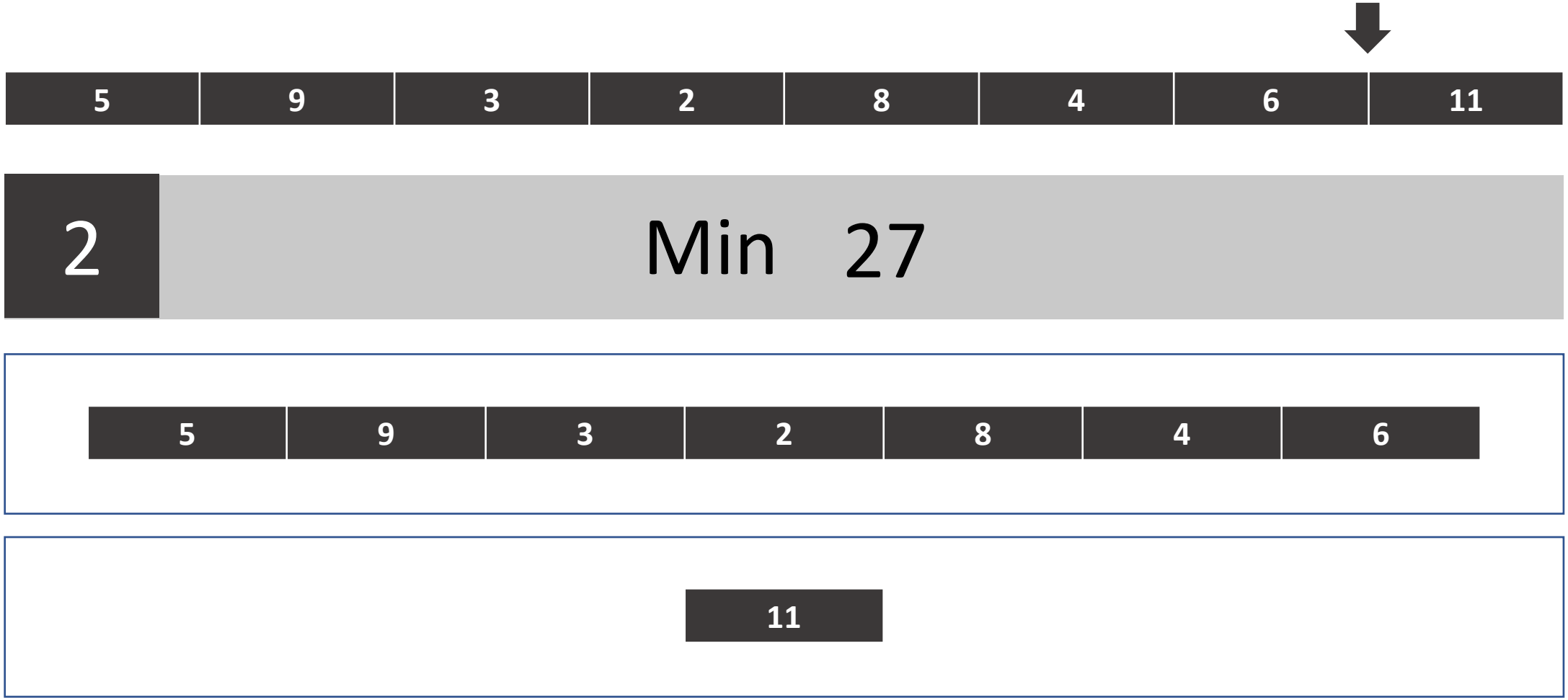

5
9
3
2
8
4
6
11
2
Min 34
5
9
14
3
2
8
4
6
11
34


5
9
3
2
8
4
6
11
2
Min 31
5
9
3
17
2
8
4
6
11
31

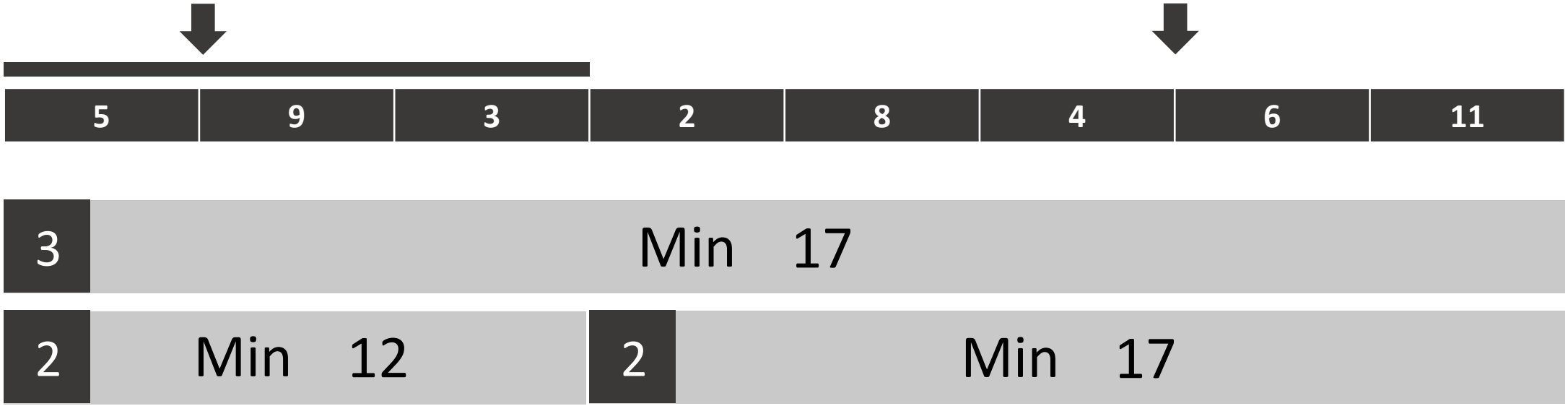

5
9
3
2
8
4
6
11
2
Min 29
5
9
3
2
19
8
4
6
11
29


5
9
3
2
8
4
6
11
2
Min 27
5
9
3
2
8
27
4
6
11
21


5
9
3
2
8
4
6
11
2
Min 27
5
9
3
2
8
4
31
6
11
17

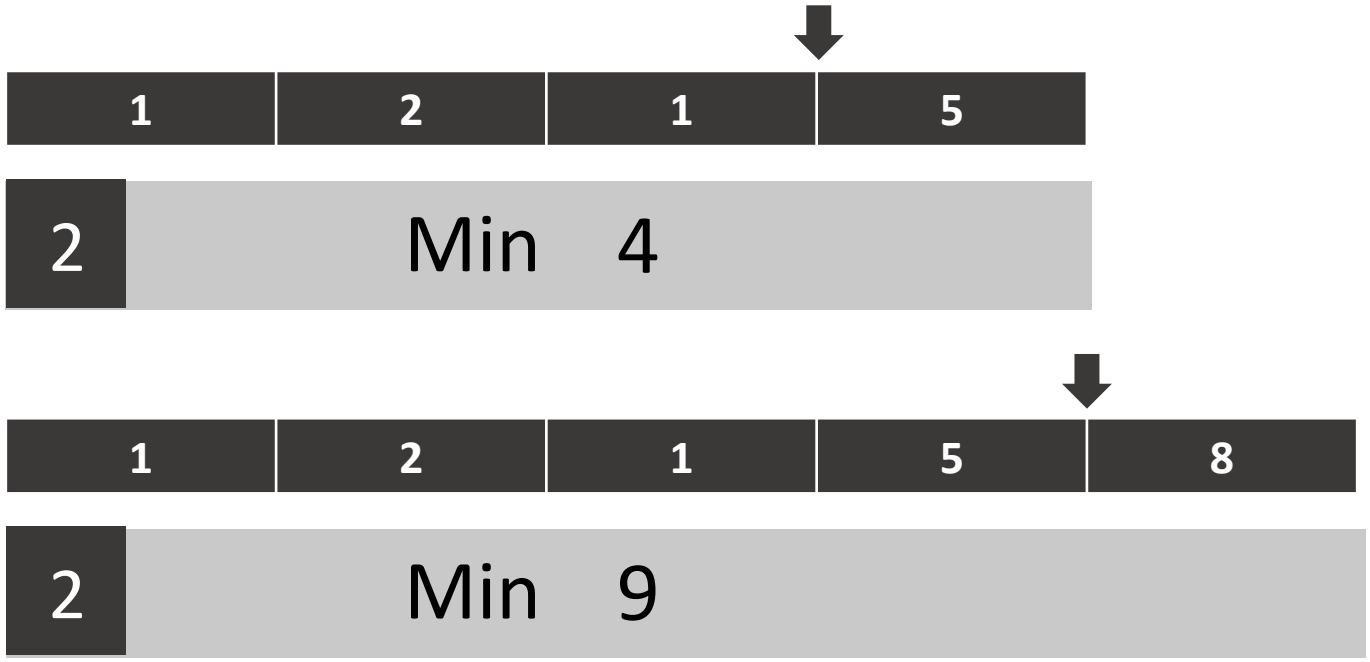

5
9
3
2
8
4
6
11
2
Min 27
5
9
3
2
8
4
6
37
11
11

Solution: Insights (1 of 3)



- Min weight of (k-1 partitions) useful for min weight of (k partitions)

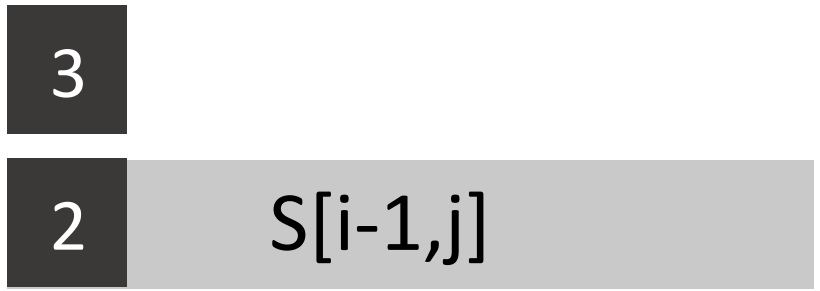
Solution: Insights (2 of 3)



if (increasing partition lengths from left)
and (left component < right component)

→ boundary will not move left

Solution: Insights (3 of 3)



S[i-1,j] with no delay tolerance

+

Sum(j+1 : j + <delay tolerance>)

For a delay tolerant solution, add the next b items to the calculated weight of any portion of the array.

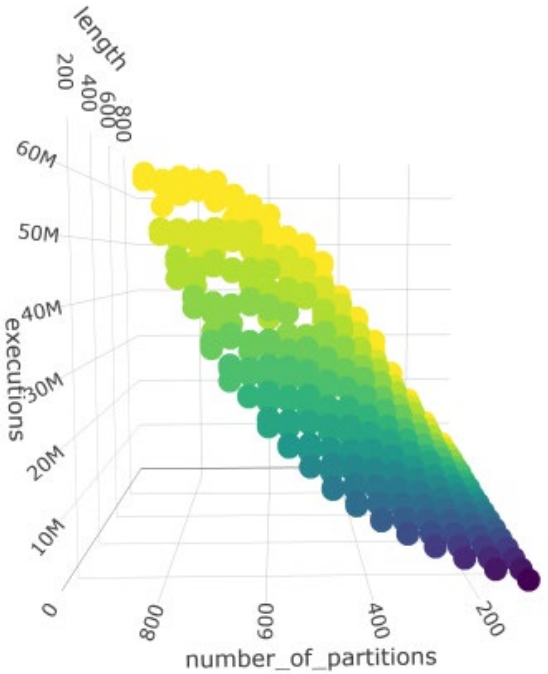
Topics Covered

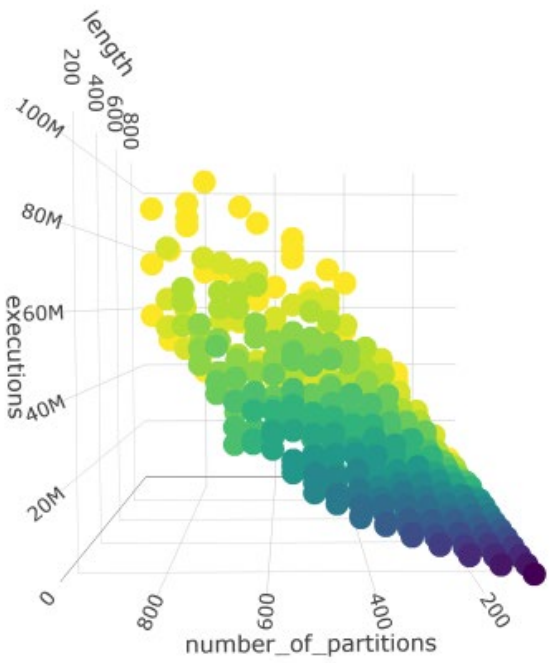
- Problem Definition
- Problem Importance
- Solution
- Empirical Results

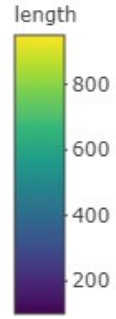
Topics Covered

- Problem Definition
- Problem Importance
- Solution
- Empirical Results

Empirical Results







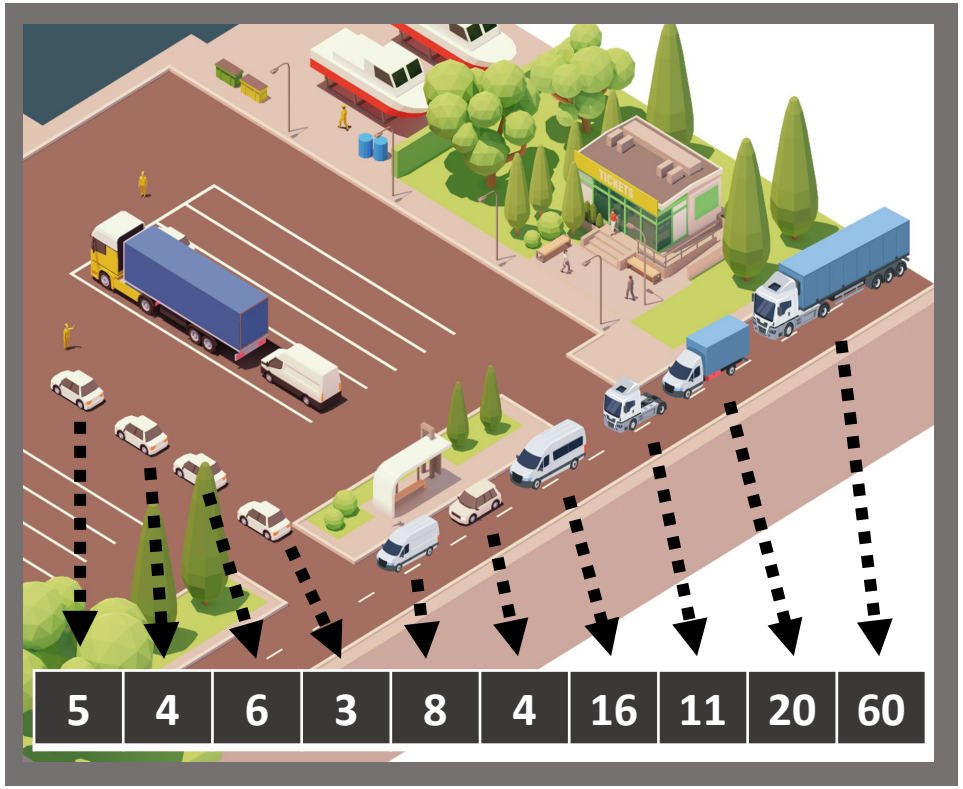
$$60\ n\ k = O(\ n\ k\ )$$

References

1. Wikipedia “bin packing problem”
2. Aydın, N., Muter, İ. and Birbil, Ş.İ., 2020. Multi-objective temporal bin packing problem: An application in cloud computing. Computers & Operations Research, 121, p.104959.
3. Coffman, Jr, E.G., Garey, M.R. and Johnson, D.S., 1978. An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing, 7(1), pp.1-17.
4. Leinberger, W., Karypis, G. and Kumar, V., 1999, September. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In Proceedings of the 1999 International Conference on Parallel Processing (pp. 404-412). IEEE.
5. Aydın, N., Muter, İ. and Birbil, Ş.İ., 2020. Multi-objective temporal bin packing problem: An application in cloud computing. Computers & Operations Research, 121, p.104959.
6. http://notexponential.com/382/minimizing-weight-of-a-linear-partition

Thanks for listening!

- Questions?




```
num_partition_index: 2 array_length_index: 7 sfl[array_length_index]: 91 ow[num_partition_index-1][array_length_index-1]: 26
new min is 26.
left window 0 to 6 right window 7 to 7 comparison: 26 vs 11
left window 0 to 5 right window 6 to 7 comparison: 15 vs 27
skipping index where left window ends at: 4
skipping index where left window ends at: 3
skipping index where left window ends at: 2
skipping index where left window ends at: 1
num_partition_index: 2 array_length_index: 8 sfl[array_length_index]: 80 ow[num_partition_index-1][array_length_index-1]: 30
new min is 80. now skipping indices less than 7
left window 0 to 7 right window 8 to 8 comparison: 30 vs 80
skipping index where left window ends at: 6
skipping index where left window ends at: 5
skipping index where left window ends at: 4
skipping index where left window ends at: 3
skipping index where left window ends at: 2
skipping index where left window ends at: 1
num_partition_index: 2 array_length_index: 9 sfl[array_length_index]: 60 ow[num_partition_index-1][array_length_index-1]: 80
new min is 80.
left window 0 to 8 right window 9 to 9 comparison: 80 vs 60
new min is 80. now skipping indices less than 7
left window 0 to 7 right window 8 to 9 comparison: 30 vs 80
skipping index where left window ends at: 6
skipping index where left window ends at: 5
skipping index where left window ends at: 4
skipping index where left window ends at: 3
skipping index where left window ends at: 2
skipping index where left window ends at: 1
executions: 45 indices_skipped: 36
final optimal_weights
[[5, 9, 15, 18, 26, 30, 46, 57, 77, 137], [0, 5, 9, 9, 15, 15, 26, 30, 80, 77], [0, 0, 6, 9, 9, 12, 16, 26, 80, 80]]
optimal weight of input array: 80
partitionAndEvaluate results: arr.length: 10, number_of_partitions: 3, optimal_weight: 80, executions: 45, indices_skipped: 36
```

Minimum weight capacity is 80

Project github is at https://github.com/RanaAlsaadi/Project4