Branch and Bound

- Travel Salesman Problem
- Knapsack Problem

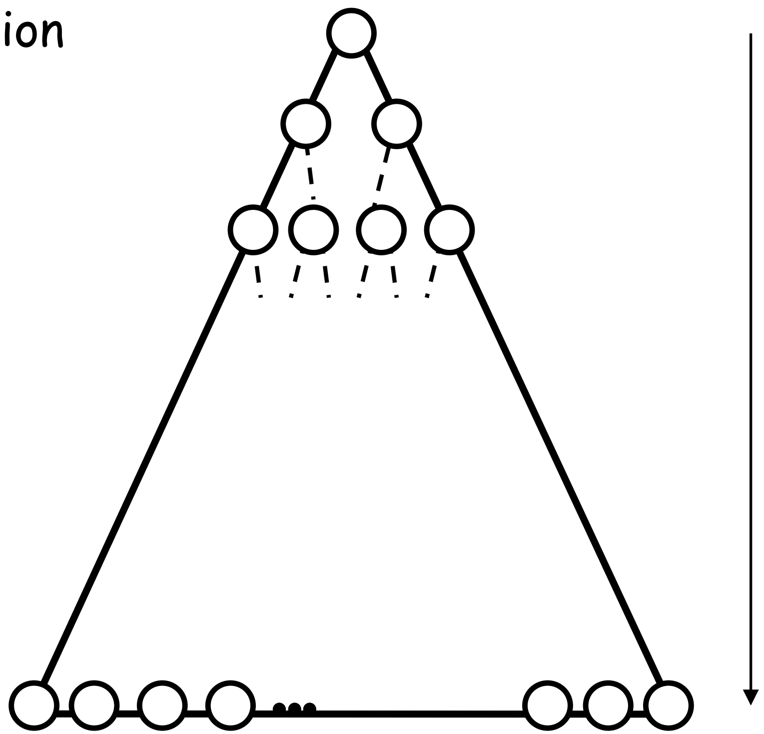
Branch and Bound

- B&B strategy similar to backtracking searches a tree (the recursion tree)
- B&B strategy can be used to solve optimization problems and includes two mechanisms.
  - A mechanism to generate branches (possible choices) when searching the solution space
  - A mechanism to generate a bound so that many branches can be terminated
- It is efficient in the average case because many branches can be terminated very early.
- Although it is usually very efficient, a very large tree may be generated in the worst case.
- Many NP-hard problem can be solved by B&B efficiently in the average case; however, the worst case time complexity is still exponential.

Branch and Bound

Explore all alternatives

- Solution constructed by stepwise choices
- Decision tree (recursion tree)
- Guarantees optimal solution
- Exponential time (slow)



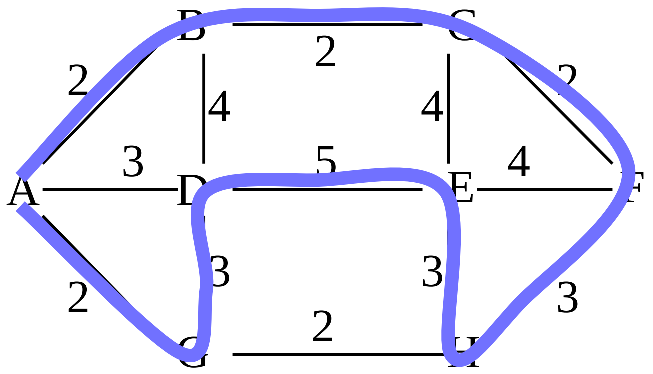
Travel Salesman Problem

Travel Salesman Problem

Input. A complete weighted graph G(V,E)

Output. Shortest tour visiting every node exactly once.

Example. Assume every edge that is not shown, has weight equal to infinity.

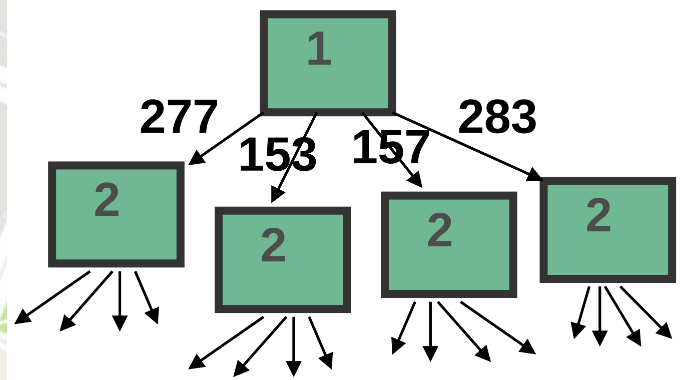


Optimal = A-B-C-F-H-E-D-G-A
Length = 22

Brute Force TSP

Brute force:
search full tree

O(N!)

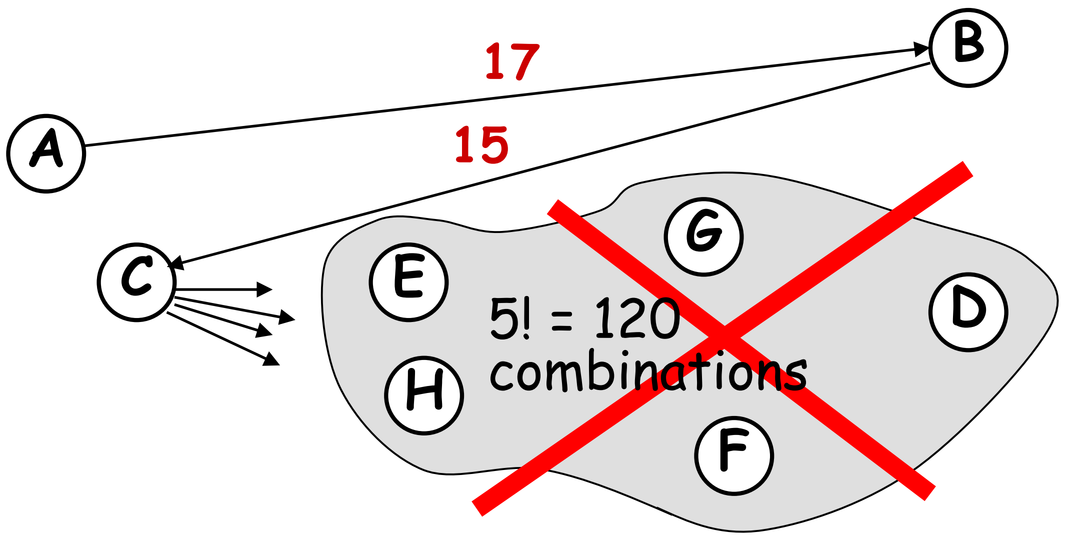


Bounding Criterion 1

Maintain the best found so far.

- you can stop searching when the length of the sub-tour travelled so far is greater than the best found so far.

Example. Assume the best found so far is 32 and your search selects the sub-tour ABC.

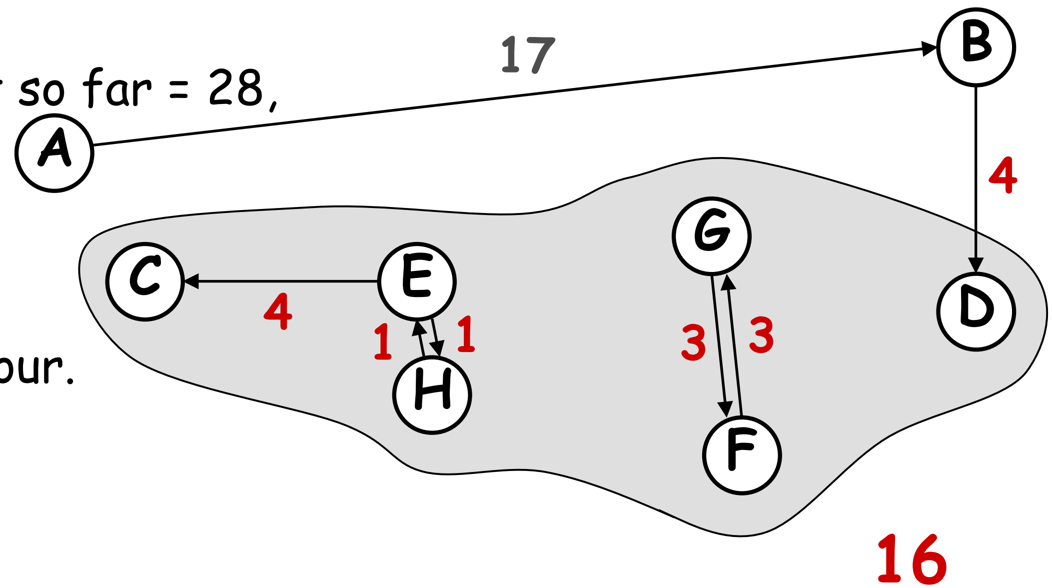


Bounding Criterion 2

Try to find a lower bound for the remaining tour.

- All nodes must be visited.
- So, find the smallest possible incoming link

Example. Assume the best so far = 28, and you start from A and then you choose B. Try to find a good lower bound for the remaining tour.



A Lower bound on the whole path: 17+16=33. Since it is greater than 28, we can terminate branching from B and backtrack to A, and try other paths.

B&B Solution

T: an array maintaining the sub-tour found so far.

```
bestsofar = +infinity

TSP(T,r)
  if r = n+1 then
     if the length of tour T is less than bestsofar then
        bestsofar = the length of tour T
  else
     for any vertex v not appear on sub-tour T[1..r-1] do
         T[r] = v
         LB = a lower bound on any tour whose prefix is T[1..r-1]
         if LB < bestsofar then
            TSP(T, r+1)
```

- There may exist different algorithms to compute LB.
- Larger LB, less search.
- Indeed, if LB > bestsofar, we terminate branching at v which may take exponential time.
- It is better to consume polynomial time at v to find a good lower bound instead of branching at v which may take exponential time.

Knapsack Problem

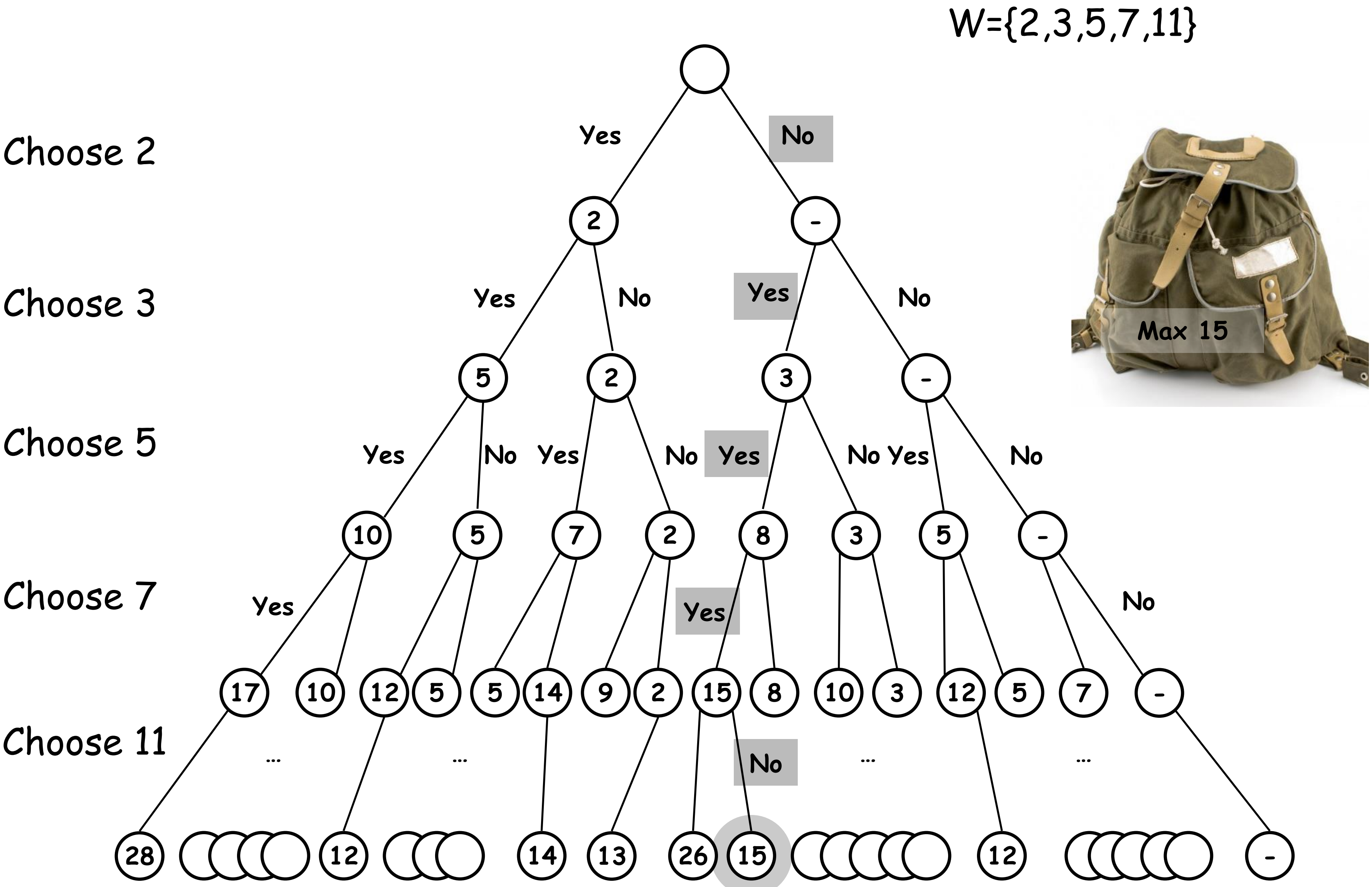
Knapsack Problem.

- Input. Weight of n items $W=\{w_1, w_2, ..., w_n\}$ and knapsack limit S
- Output. Selection for knapsack $\{x_1, x_2, ..., x_n\}$ where $x_i \in \{0,1\}$ that maximizes the whole weight.

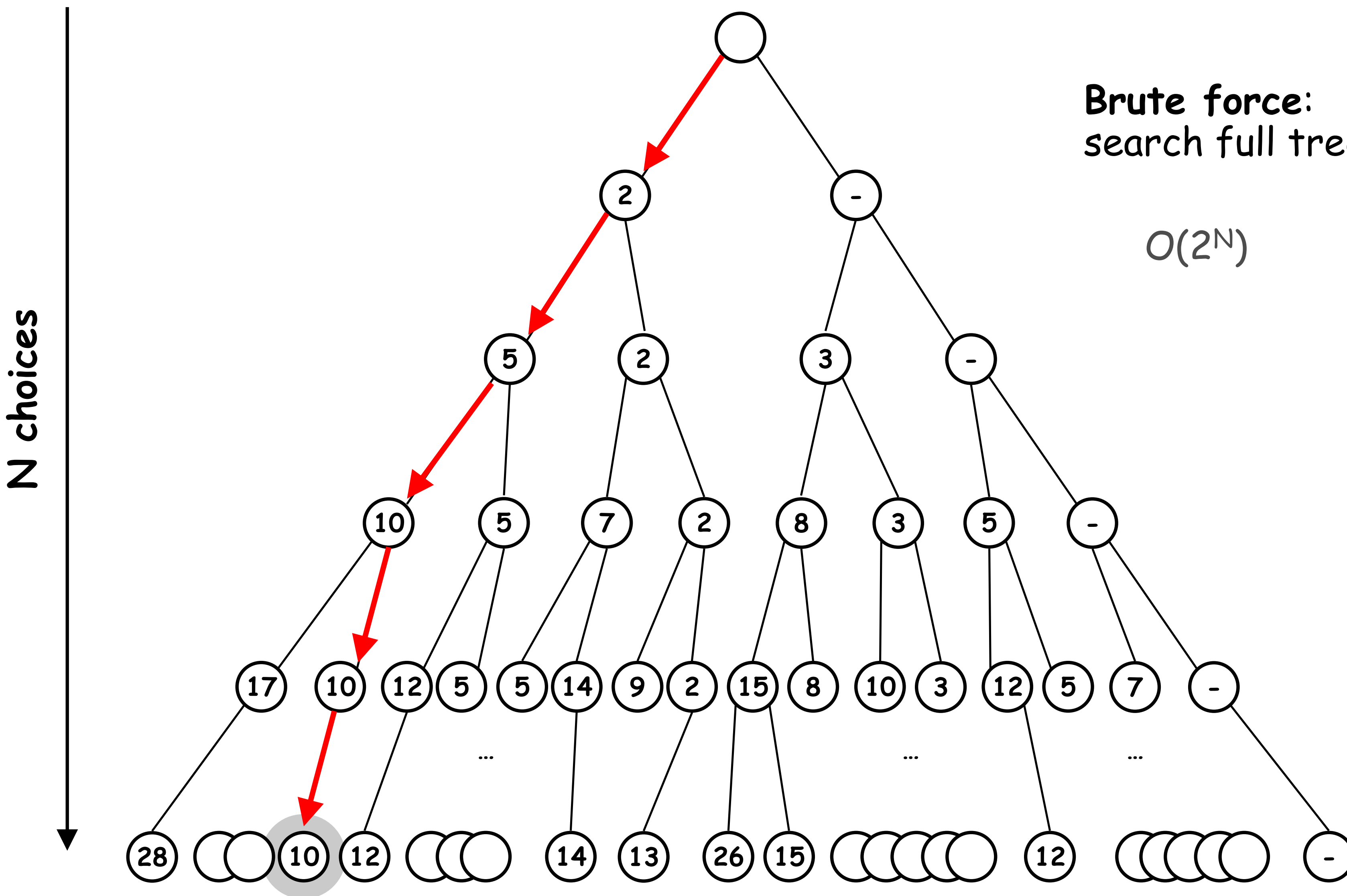
Example.

- $W = \{2, 3, 5, 7, 11\}$ and $S = 15$

Decision Tree (Recursion Tree)



Decision Tree (Recursion Tree)



Brute force:
search full tree

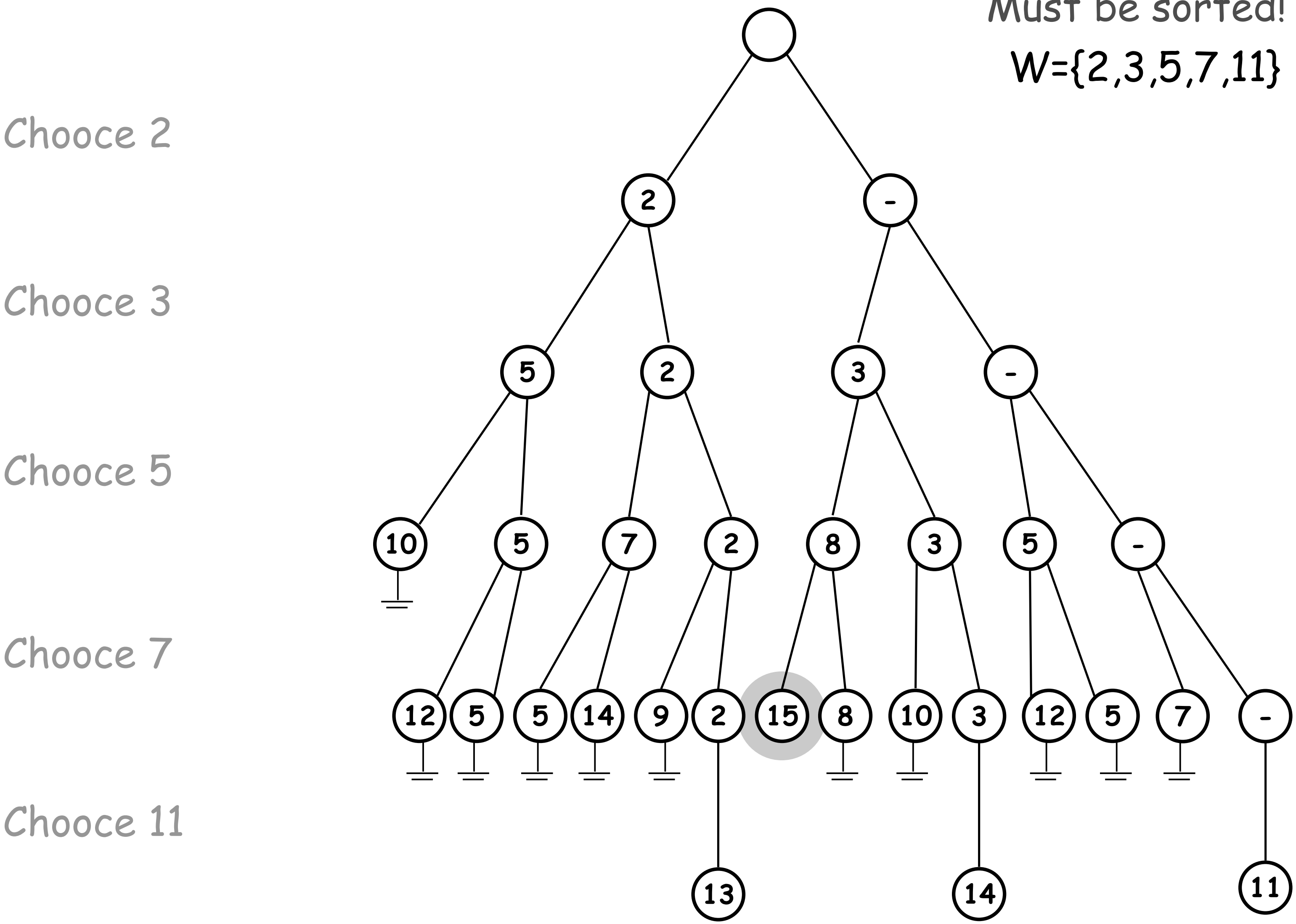
$O(2^N)$

Bounding Criterion

$\Sigma + w_{next} > S$

Must be sorted!

W={2,3,5,7,11}



General Pattern

General Pattern

```
Bestsofar = +infinity in the minimization problem

Process at a node u of the recursion tree

if (u is a leaf) and (a solution is found) then
   if the solution is better than bestsofar then
      update bestsofar
else
   for any child v of u (i.e. any possible choice) do
      compute a lower bound (in polynomial time)
      if the lower bound is less than bestsofar then
         process v
backtrack to the parent of u
```

By finding a good lower bound at node v in polynomial time, we terminate branching at v which may take exponential time if the optimal solution is not inside the subtree rooted at v.

References

References

- The original slides were prepared by Pasi Fränti