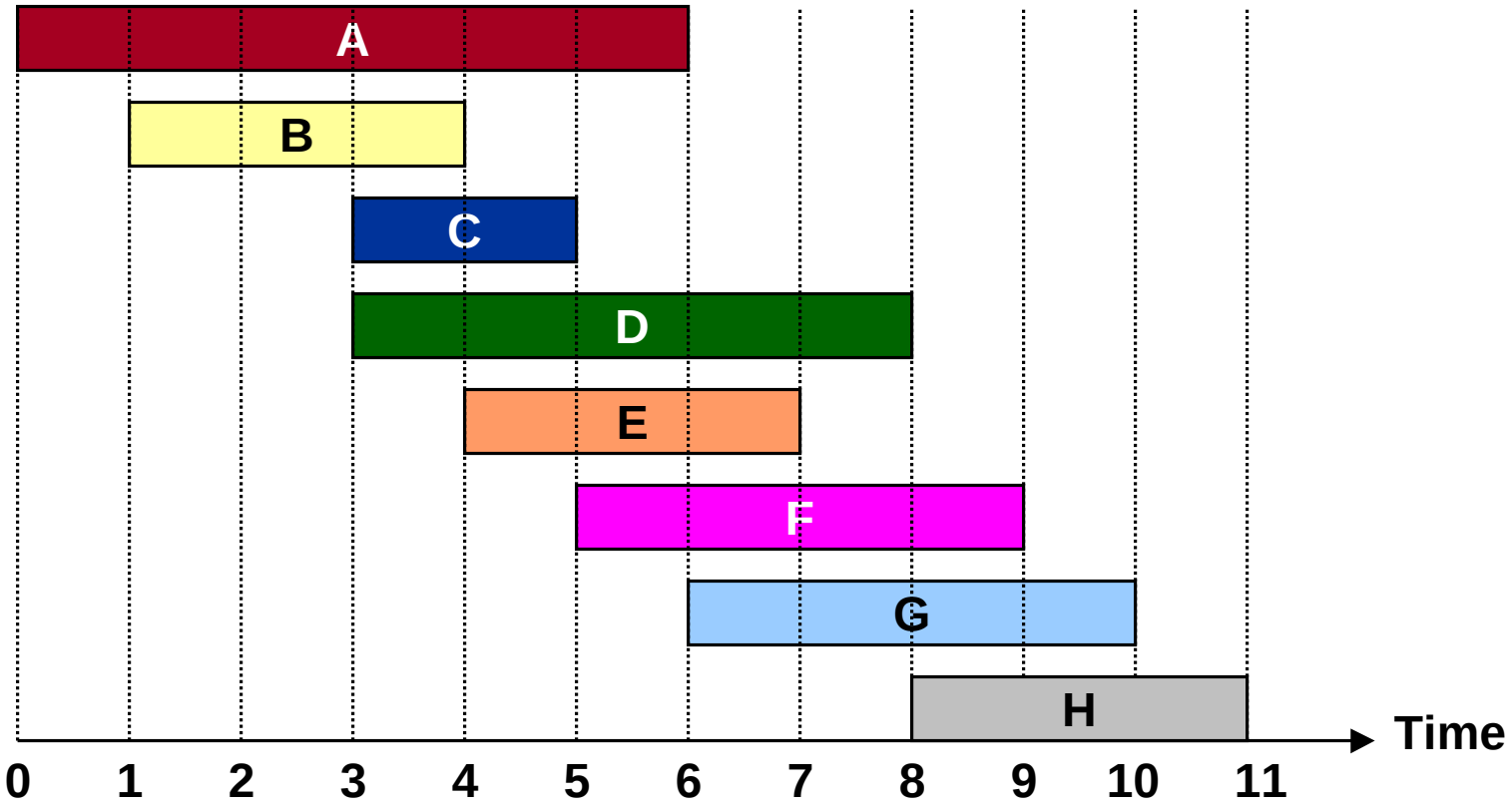
Weighted Activity Selection

Weighted activity selection problem (generalization of CLR 17.1).

- Job requests 1, 2, … , N.
- Job j starts at s_j, finishes at f_j , and has weight w_j .
- Two jobs compatible if they don't overlap.
- Goal: find maximum weight subset of mutually compatible jobs.



Activity Selection: Greedy Algorithm

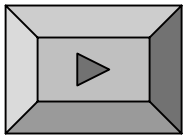
Recall greedy algorithm works if all weights are 1.

Greedy Activity Selection Algorithm

Sort jobs by increasing finish times so that $f_1 \leq f_2 \leq \ldots \leq f_N$.

$S = \phi$ ← S = jobs selected.

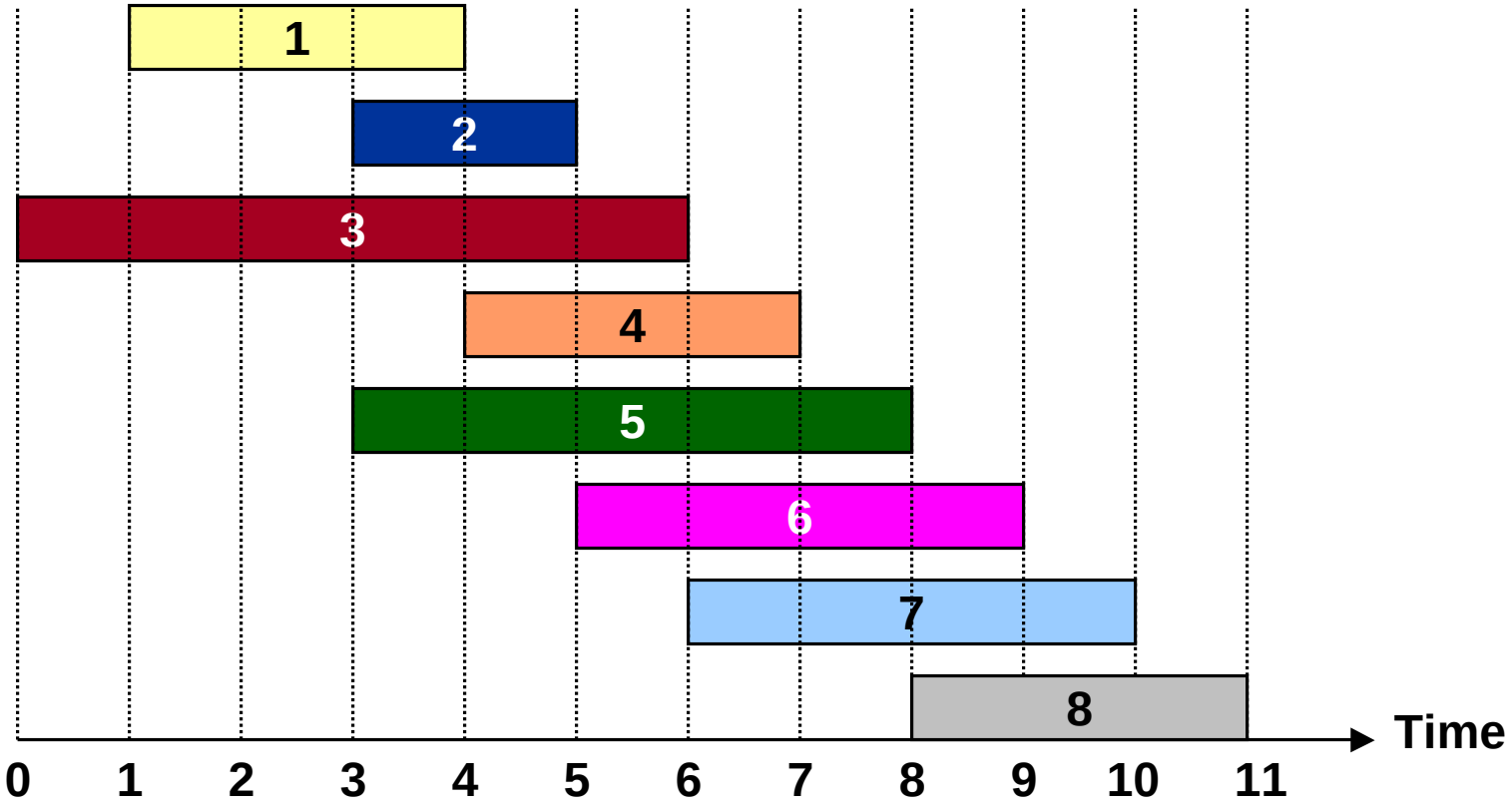
```
FOR j = 1 to N
   IF (job j compatible with A)
      S ← S ∪ {j}
RETURN S
```



Weighted Activity Selection

Notation.

- Label jobs by finishing time: $f_1 \leq f_2 \leq \ldots \leq f_N$.
- Define q_j = largest index i < j such that job i is compatible with j.
  - $q_7 = 3$, $q_2 = 0$



Weighted Activity Selection: Structure

Let OPT(j) = value of optimal solution to the problem consisting of job requests {1, 2, . . . , j }.

- **Case 1: OPT selects job j.**
  - **can't use incompatible jobs { $q_j + 1, q_j + 2, \ldots, j-1$ }**
  - **must include optimal solution to problem consisting of remaining compatible jobs { $1, 2, \ldots, q_j$ }**
- **Case 2: OPT does not select job j.**
  - **must include optimal solution to problem consisting of remaining compatible jobs { $1, 2, \ldots, j - 1$ }**

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{w_j + OPT(q_j),\ OPT(j-1)\} & \text{otherwise} \end{cases}$$

Weighted Activity Selection: Brute Force

Recursive Activity Selection

INPUT: N, $s_1,\ldots,s_N$, $f_1,\ldots,f_N$, $w_1,\ldots,w_N$

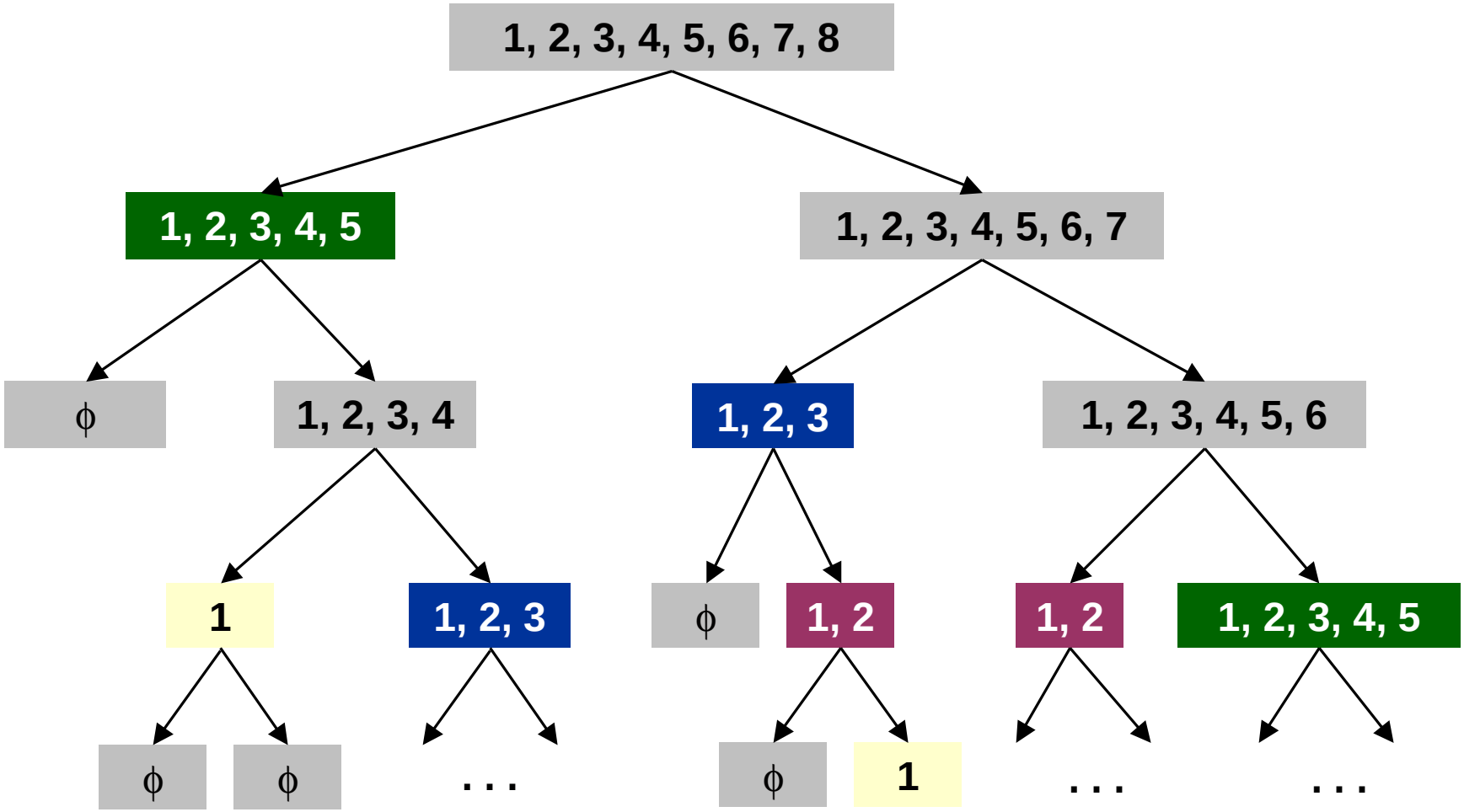
Sort jobs by increasing finish times so that $f_1 \leq f_2 \leq \ldots \leq f_N$.

Compute $q_1, q_2, \ldots, q_N$

```
r-compute(j) {
   IF (j = 0)
      RETURN 0
   ELSE
      return max(w_j + r-compute(q_j), r-compute(j-1))
}
```

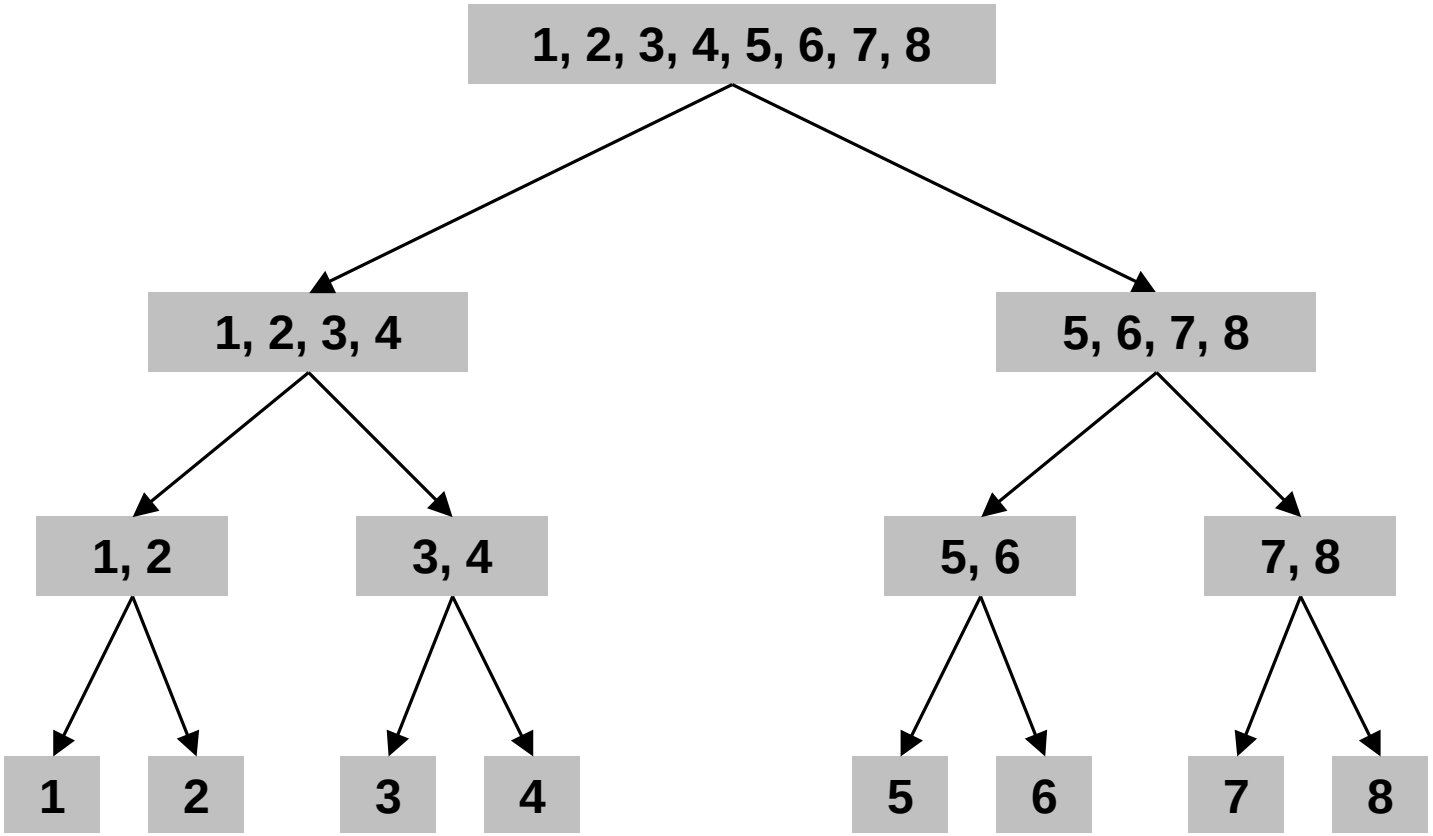
Dynamic Programming Subproblems

Spectacularly redundant subproblems $\Rightarrow$ exponential algorithms.



Divide-and-Conquer Subproblems

Independent subproblems $\Rightarrow$ efficient algorithms.



Weighted Activity Selection: Memoization

Memoized Activity Selection

INPUT: N, $s_1,\ldots,s_N$, $f_1,\ldots,f_N$, $w_1,\ldots,w_N$

Sort jobs by increasing finish times so that $f_1 \leq f_2 \leq \ldots \leq f_N$.

Compute $q_1, q_2 , \ldots , q_N$

```
Global array OPT[0..N]
FOR j = 0 to N
   OPT[j] = "empty"

m-compute(j) {
   IF (j = 0)
      OPT[0] = 0
   ELSE IF (OPT[j] = "empty")
      OPT[j] = max(w_j + m-compute(q_j), m-compute(j-1))
   RETURN OPT[j]
}
```

Weighted Activity Selection: Running Time

Claim: memoized version of algorithm takes O(N log N) time.

- Ordering by finish time: O(N log N).
- Computing q_j : O(N log N) via binary search.

- `m-compute(j)`: each invocation takes O(1) time and either
  - (i) returns an existing value of `OPT[]`
  - (ii) fills in one new entry of `OPT[]` and makes two recursive calls

- Progress measure Φ = # nonempty entries of `OPT[]`.
  - Initially $\Phi = 0$, throughout $\Phi \leq N$.
  - (ii) increases Φ by 1 $\Rightarrow$ at most 2N recursive calls.

- Overall running time of `m-compute(N)` is O(N).

Weighted Activity Selection: Finding a Solution

`m-compute(N)` determines value of optimal solution.

- **Modify to obtain optimal solution itself.**

Finding an Optimal Set of Activities

```
ARRAY: OPT[0..N]
Run m-compute(N)

find-sol(j) {
   IF (j = 0)
      output nothing
   ELSE IF (w_j + OPT[q_j] > OPT[j-1])
      print j
      find-sol(q_j)
   ELSE
      find-sol(j-1)
}
```

- **# of recursive calls $\leq$ N $\Rightarrow$ O(N).**

Weighted Activity Selection: Bottom-Up

Unwind recursion in memoized algorithm.

Bottom-Up Activity Selection

INPUT: N, $s_1, \ldots, s_N$, $f_1, \ldots, f_N$, $w_1, \ldots, w_N$

Sort jobs by increasing finish times so that $f_1 \leq f_2 \leq \ldots \leq f_N$.

Compute q_1, q_2 , ... , q_N

ARRAY: OPT[0..N]
OPT[0] = 0

FOR j = 1 to N
   OPT[j] = max(w_j + OPT[q_j], OPT[j-1])