Using Decision Procedures Efficiently for Optimization

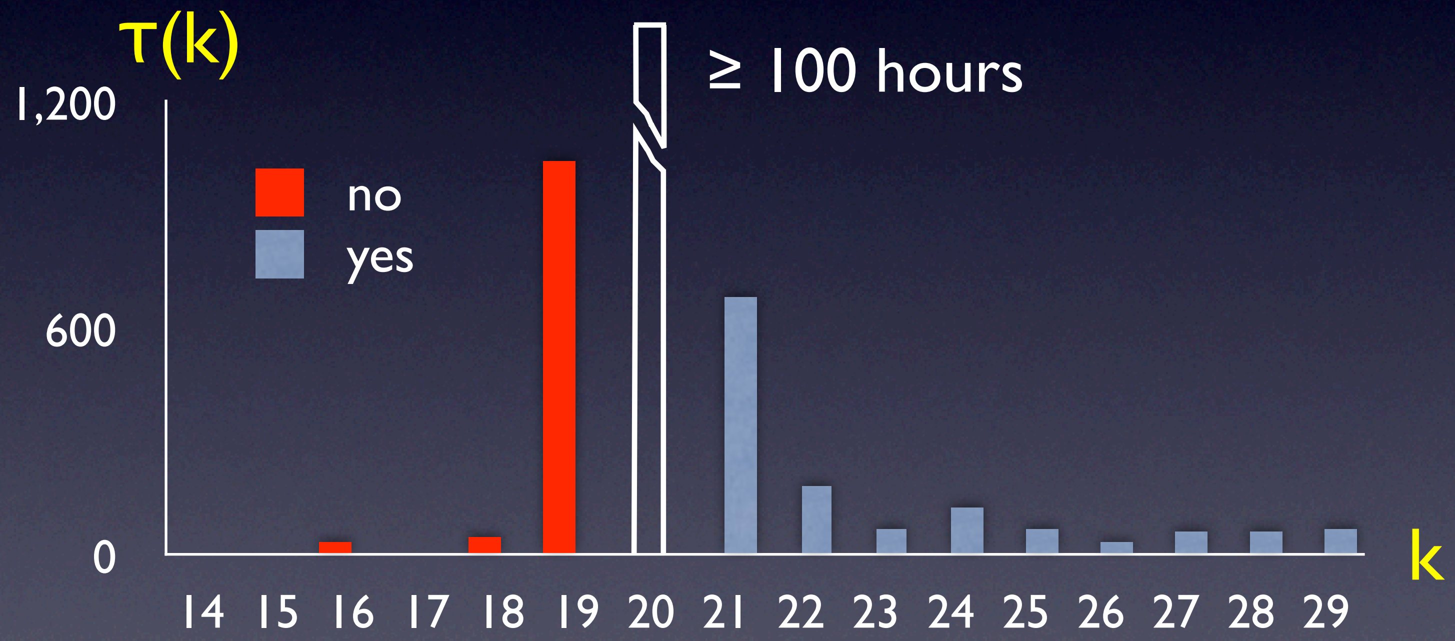
Matthew Streeter and Stephen F. Smith
Carnegie Mellon University

Introduction

- Optimization problems can be solved by asking a decision procedure questions of the form "is there a solution of cost $\leq$ k?" (e.g., SATPLAN, MAXPLAN)
- Many possible strategies for determining what question to ask next:
  - *ramp-up* (SATPLAN)
  - *ramp-down* (MAXPLAN)
  - *geometric* (Rintanen'04)
- Which is best?

Motivations

- Query strategy can dramatically affect the time needed to find an approximately optimal solution



Time required by siege (SAT solver used by SATPLAN) to determine if there exists a plan of length ≤ k

Query Strategies

- A *query* (k,t) runs the decision procedure with time limit t, and asks it "is there a solution of cost ≤ k?" Result can be "yes", "no", or "timeout".
- A *query strategy* determines the next query to execute, as a function of the results of previous queries

Query Strategies

- A *query* (k,t) runs the decision procedure with time limit t, and asks it "is there a solution of cost ≤ k?" Result can be "yes", "no", or "timeout".
- A *query strategy* determines the next query to execute, as a function of the results of previous queries

- Notation:
  - $\tau(k)$ = time required by decision proc. on input k
  - OPT = minimum solution cost

Metrics & Assumptions

Metrics & Assumptions

- Performance metric: worst-case competitive ratio. Equals max, over all k, of

$$\frac{\text{time required to prove } k \leq \text{OPT or } k \geq \text{OPT}}{\tau(k)}$$

Metrics & Assumptions

- Performance metric: worst-case competitive ratio. Equals max, over all k, of

$$\frac{\text{time required to prove } k \leq \text{OPT or } k \geq \text{OPT}}{\tau(k)}$$

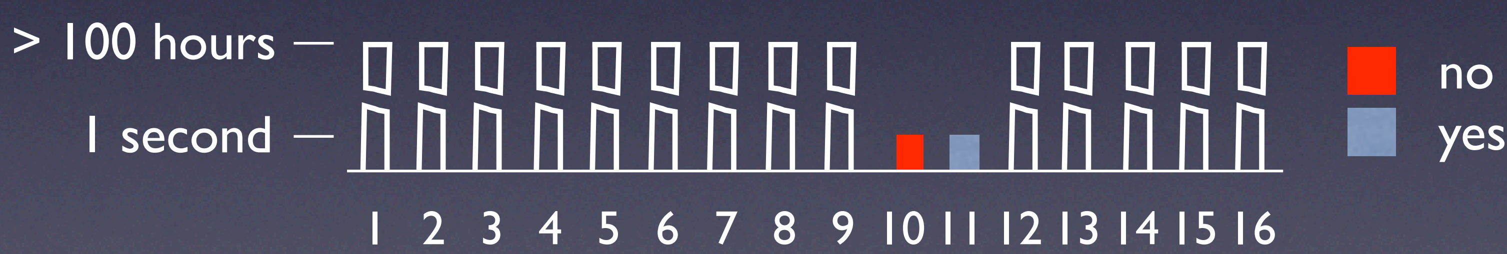
- Without any assumptions about $\tau(k)$, can't do better than trying all k-values in parallel. Competitive ratio = #(possible k-values)

Metrics & Assumptions

- Performance metric: worst-case competitive ratio. Equals max, over all k, of

$$\frac{\text{time required to prove } k \leq \text{OPT or } k \geq \text{OPT}}{\tau(k)}$$

- Without any assumptions about τ(k), can't do better than trying all k-values in parallel. Competitive ratio = #(possible k-values)



- We'll assume τ(k) is (approximately) increasing-then-decreasing

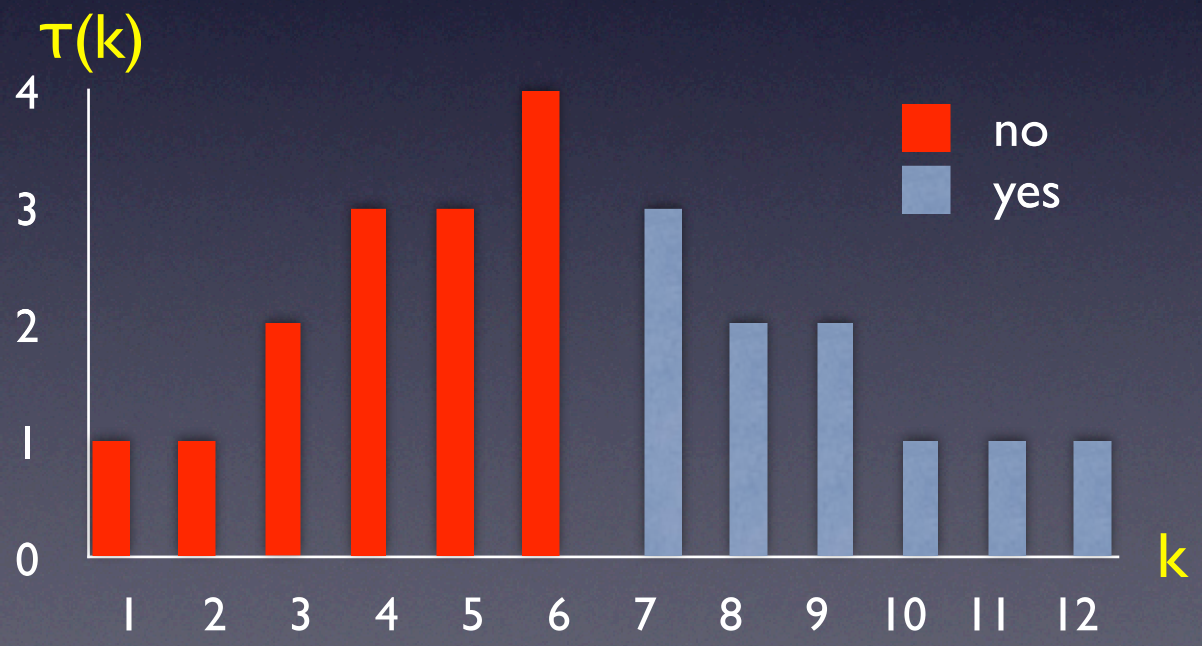
Query strategy S_2

Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat

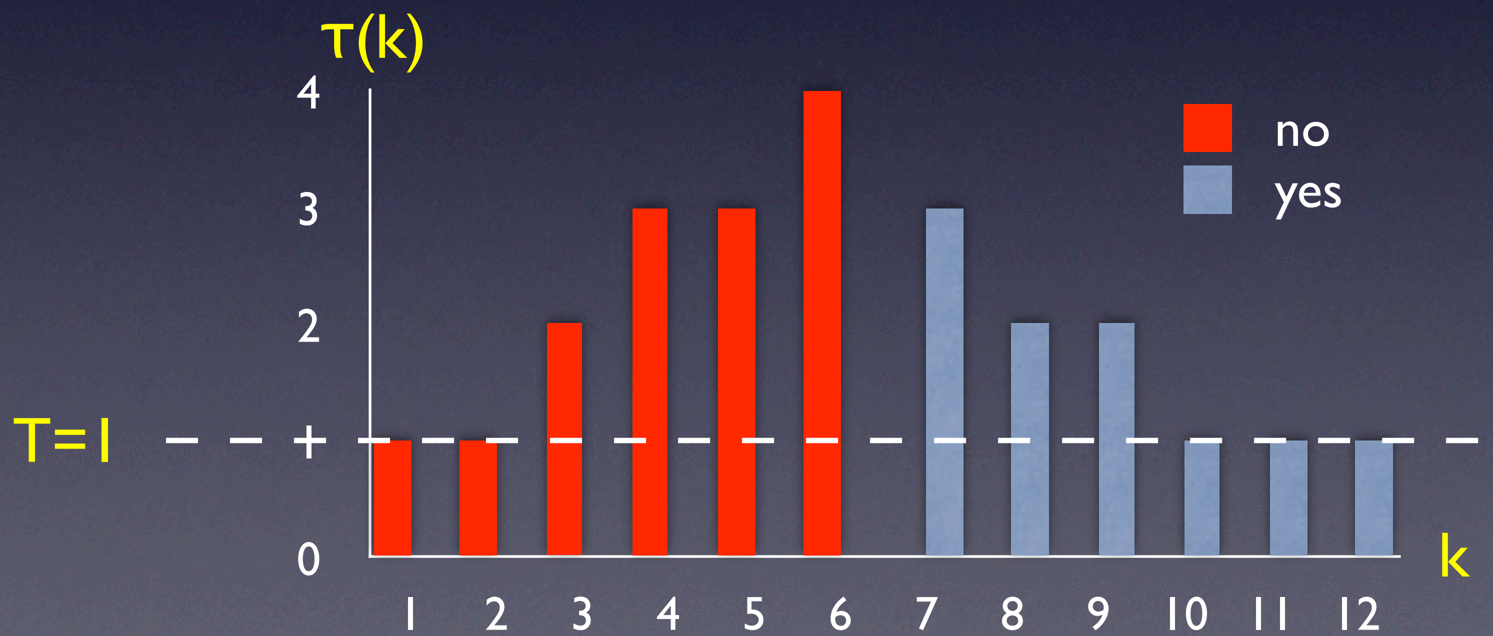
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



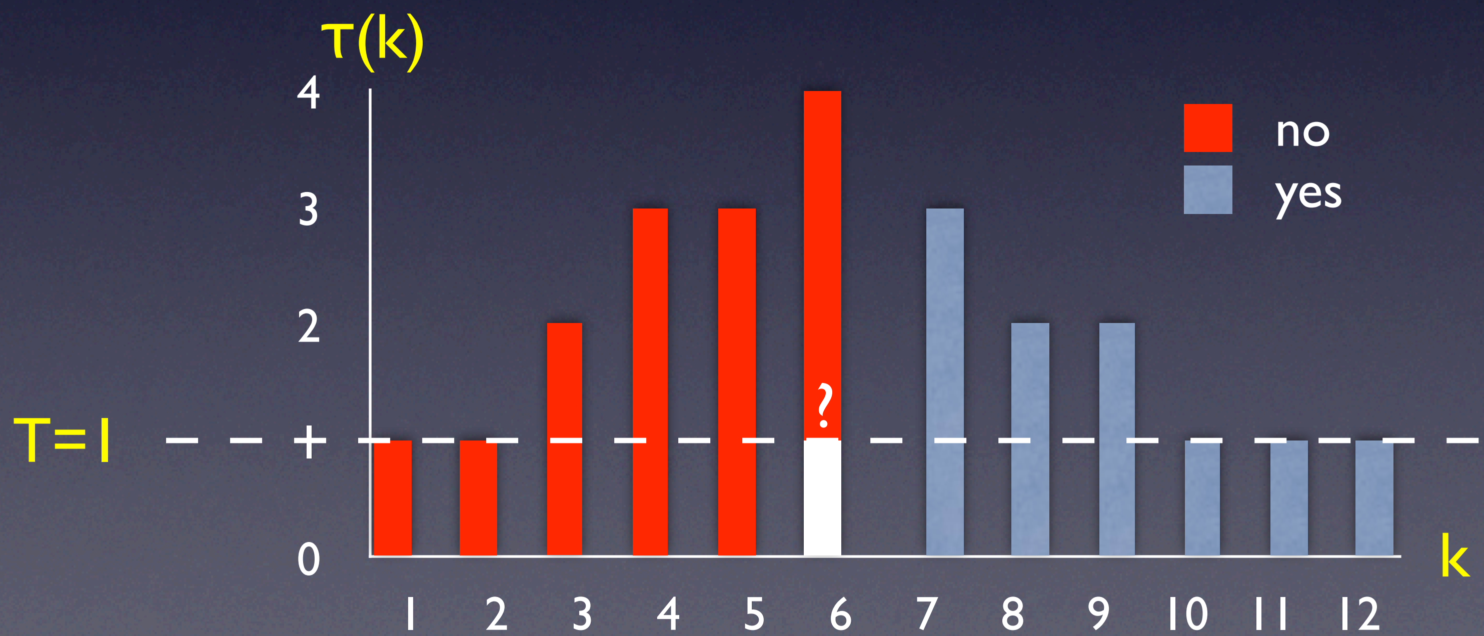
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



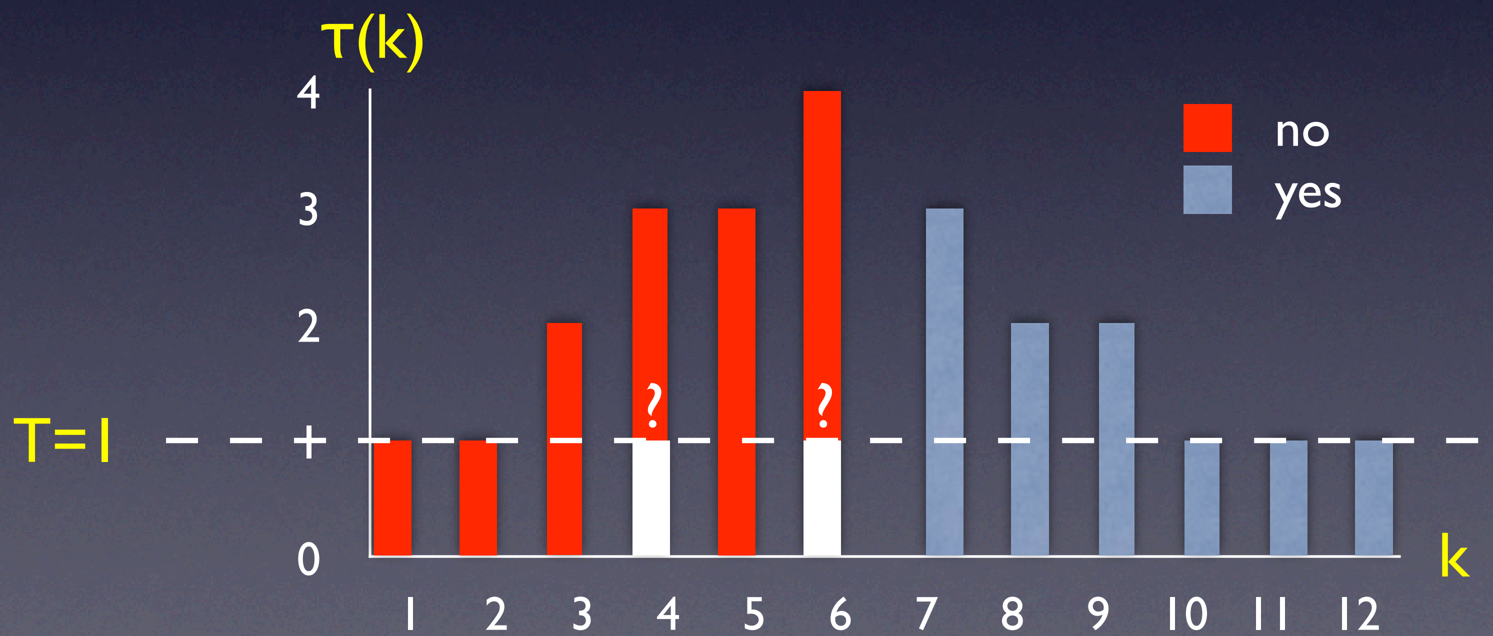
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



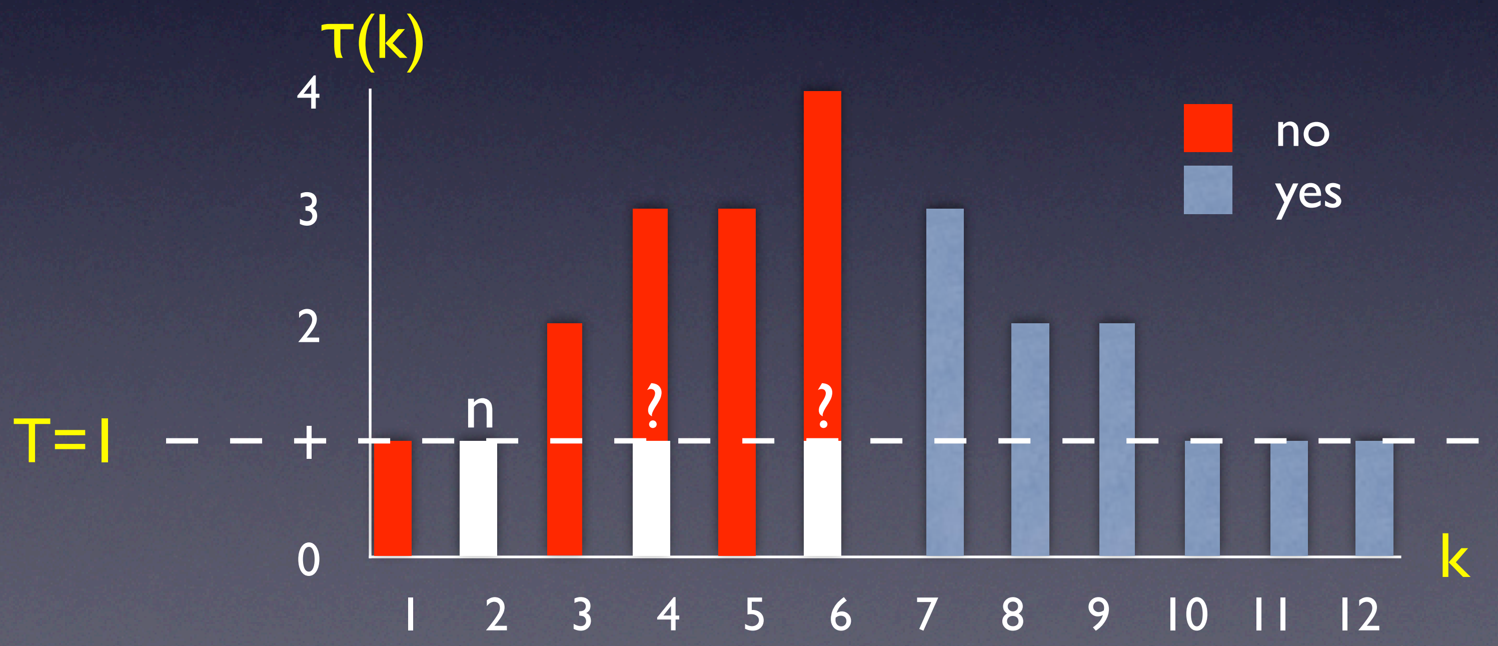
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



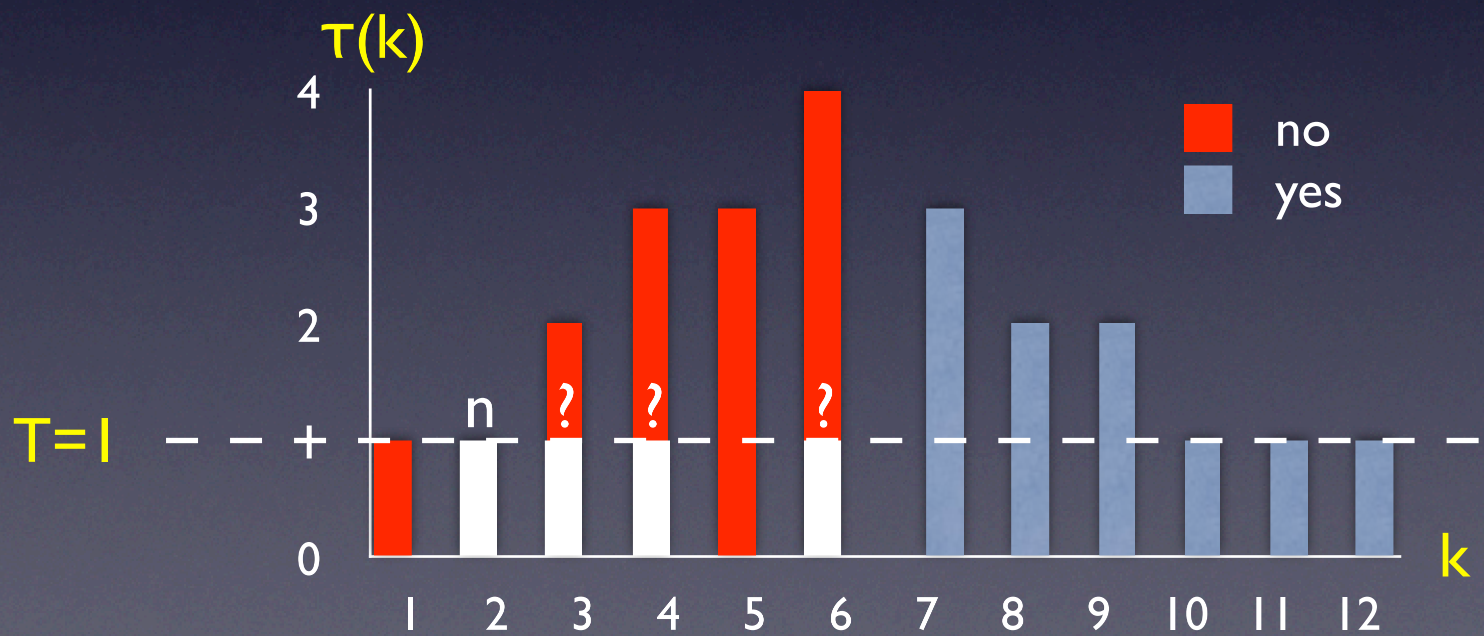
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



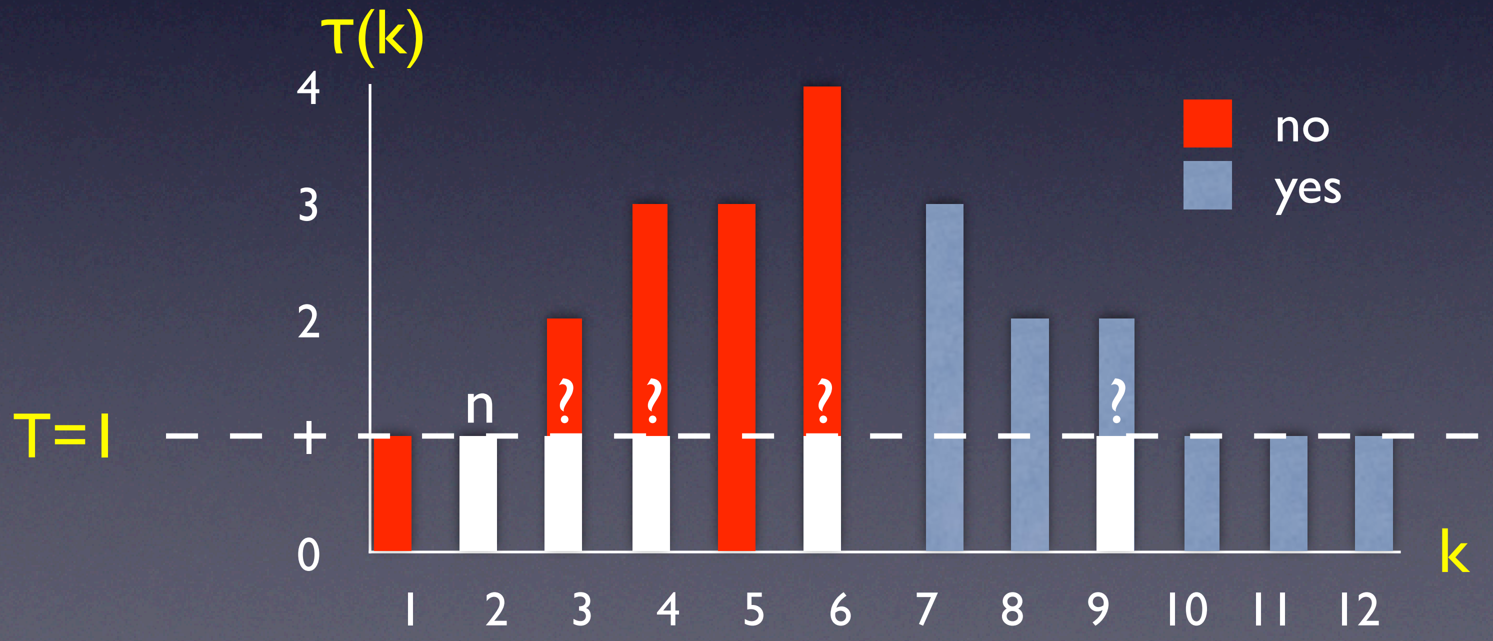
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



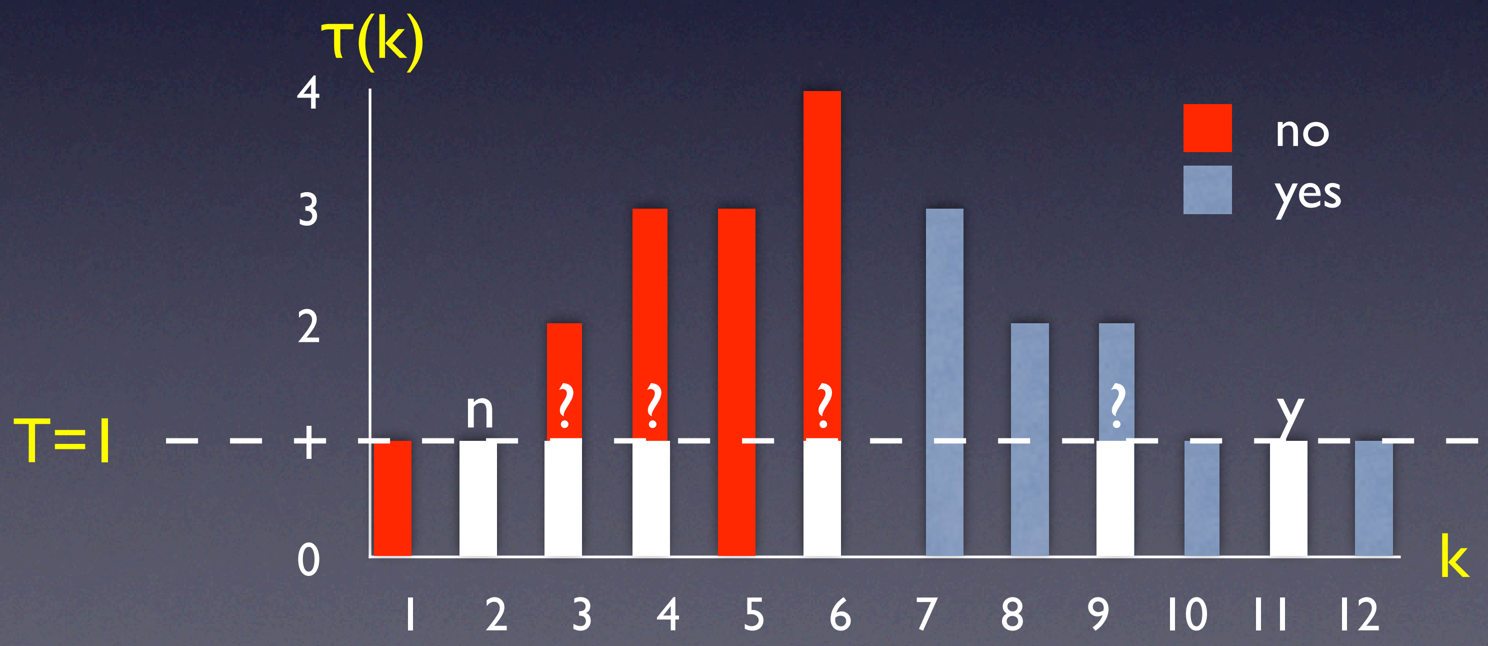
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



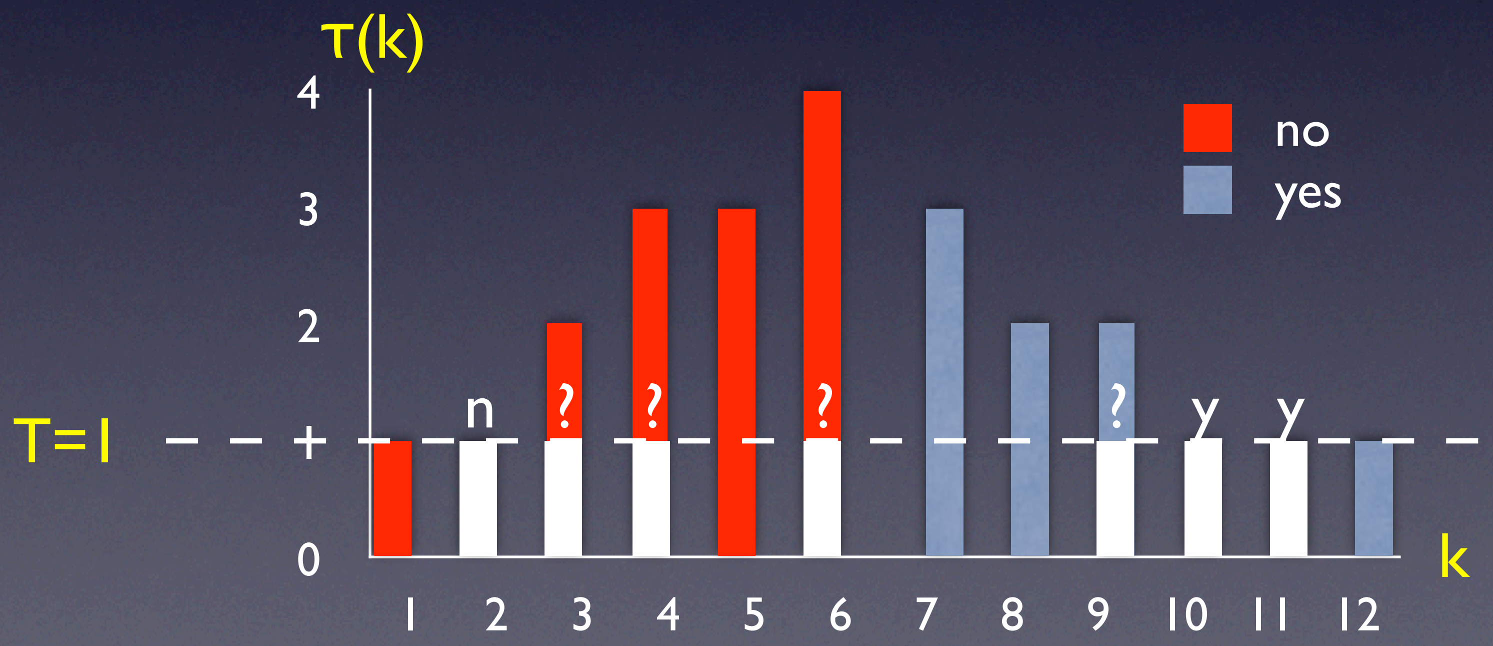
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



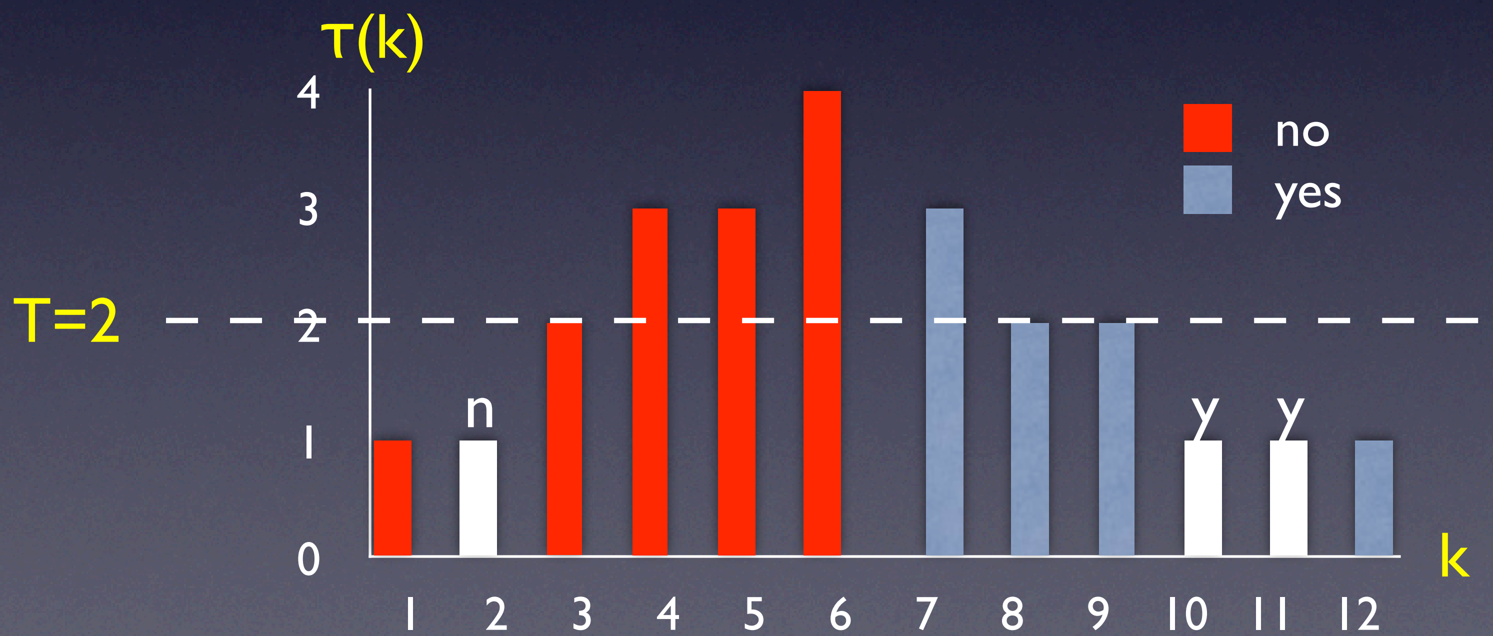
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



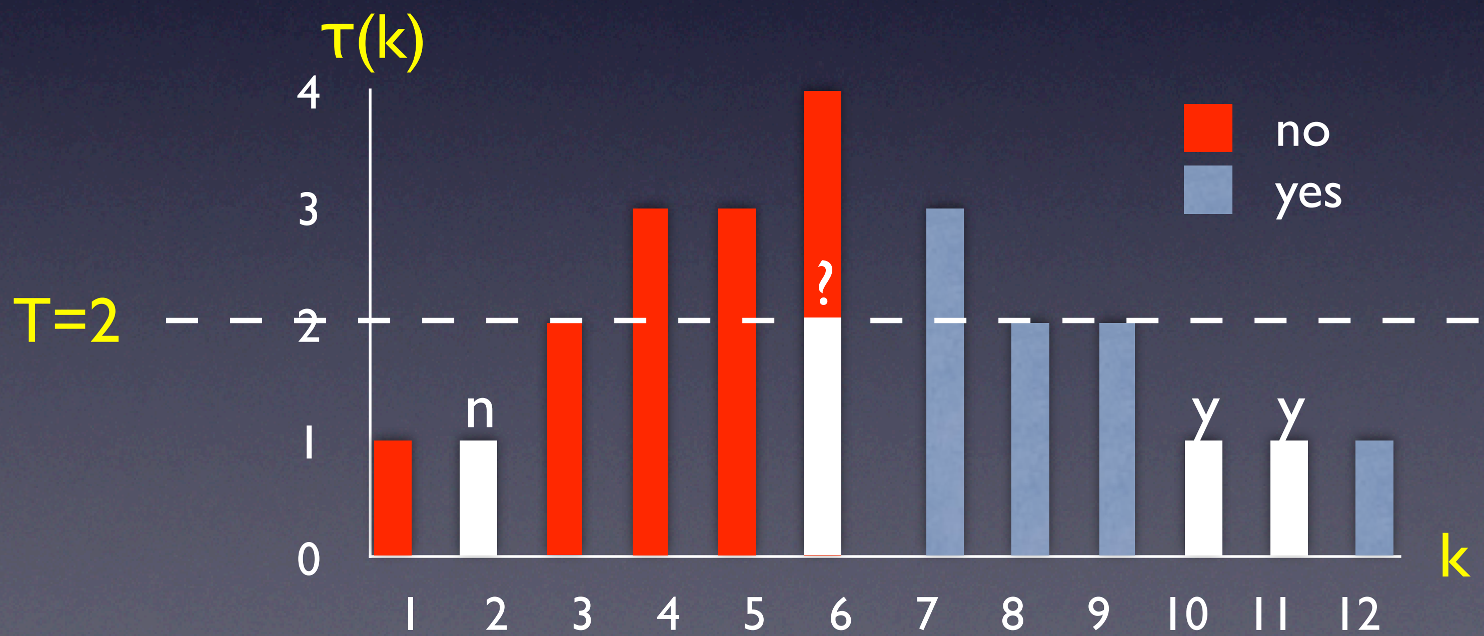
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



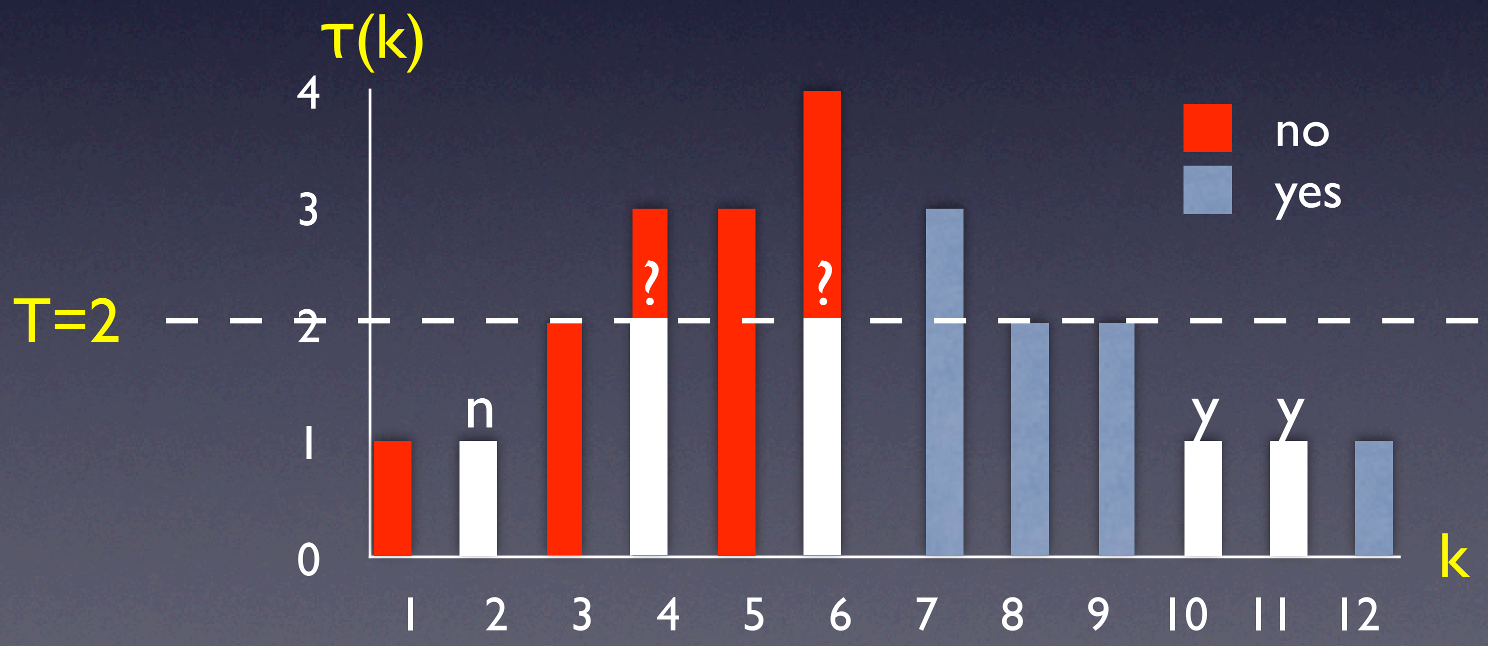
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



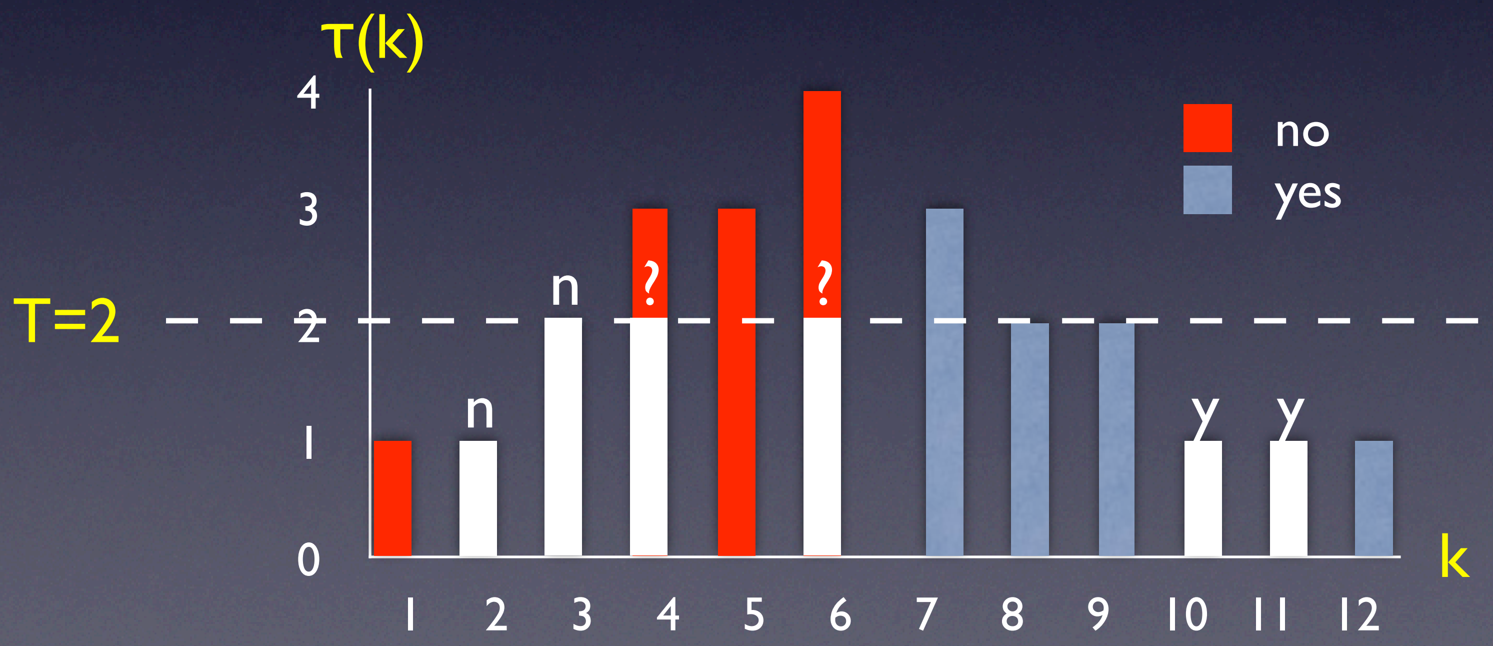
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



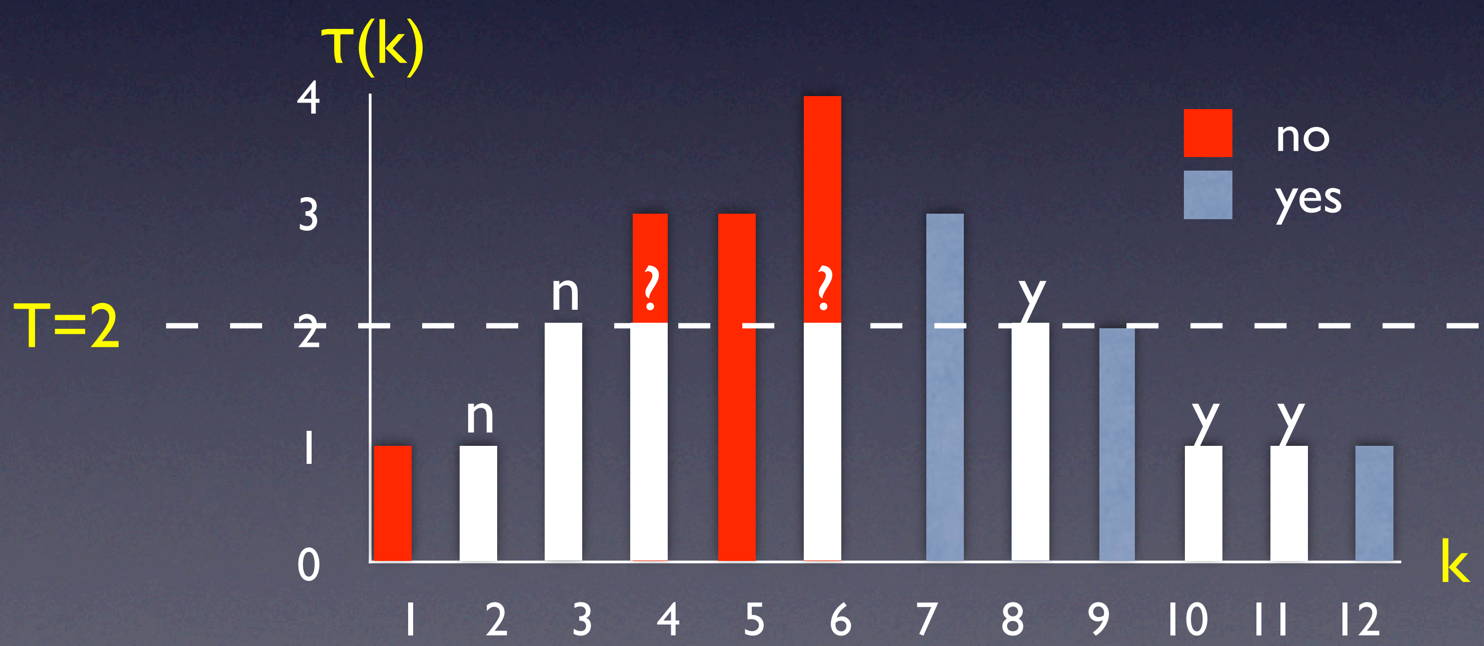
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



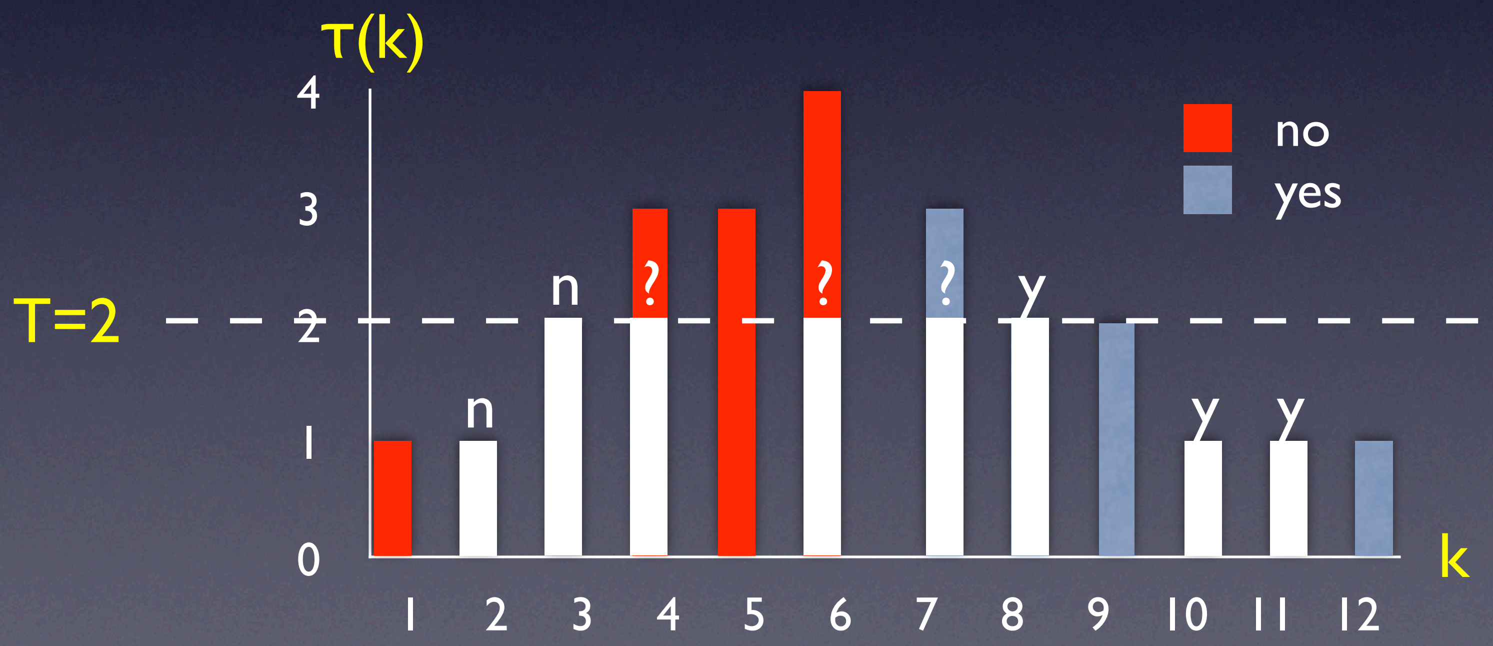
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



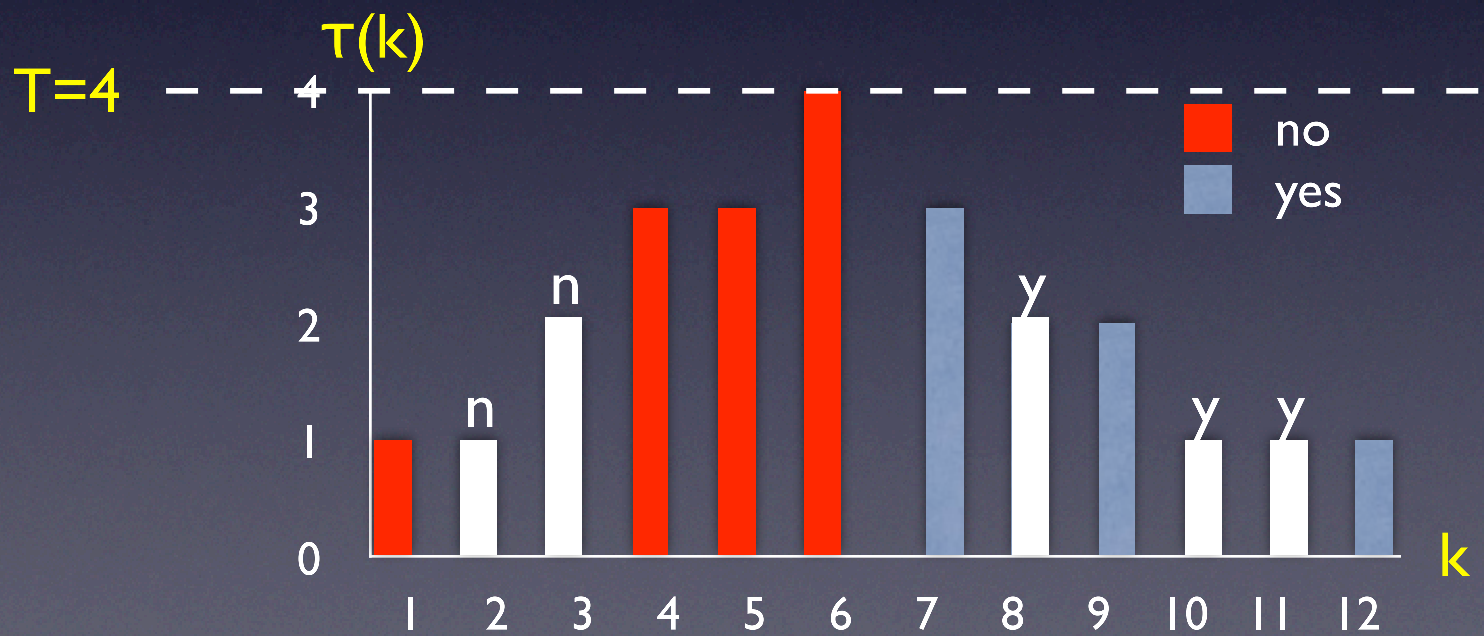
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



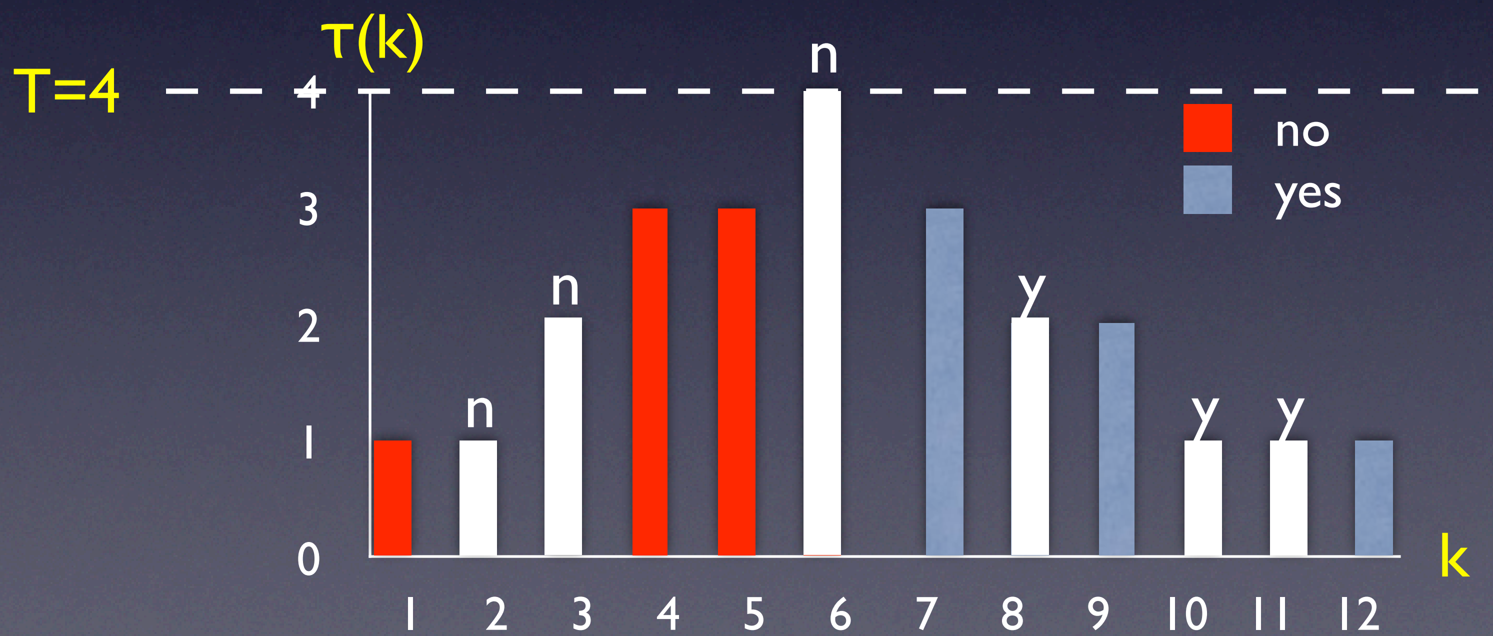
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



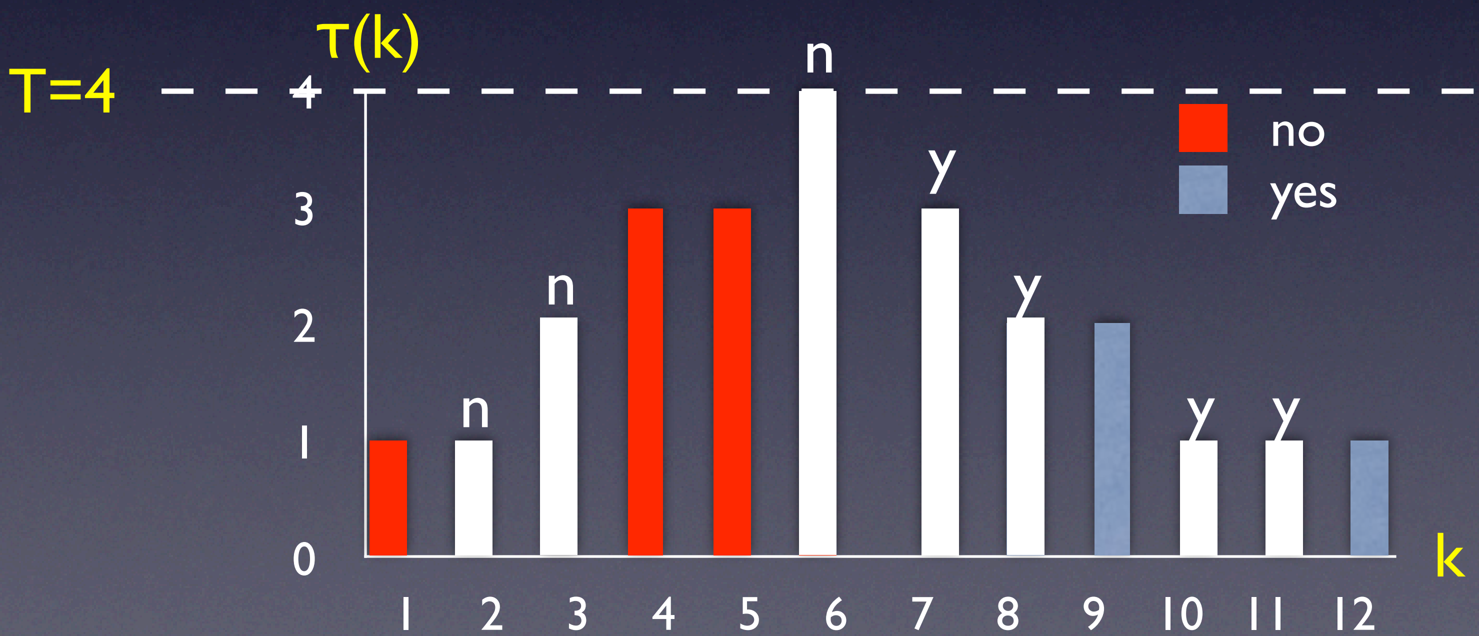
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



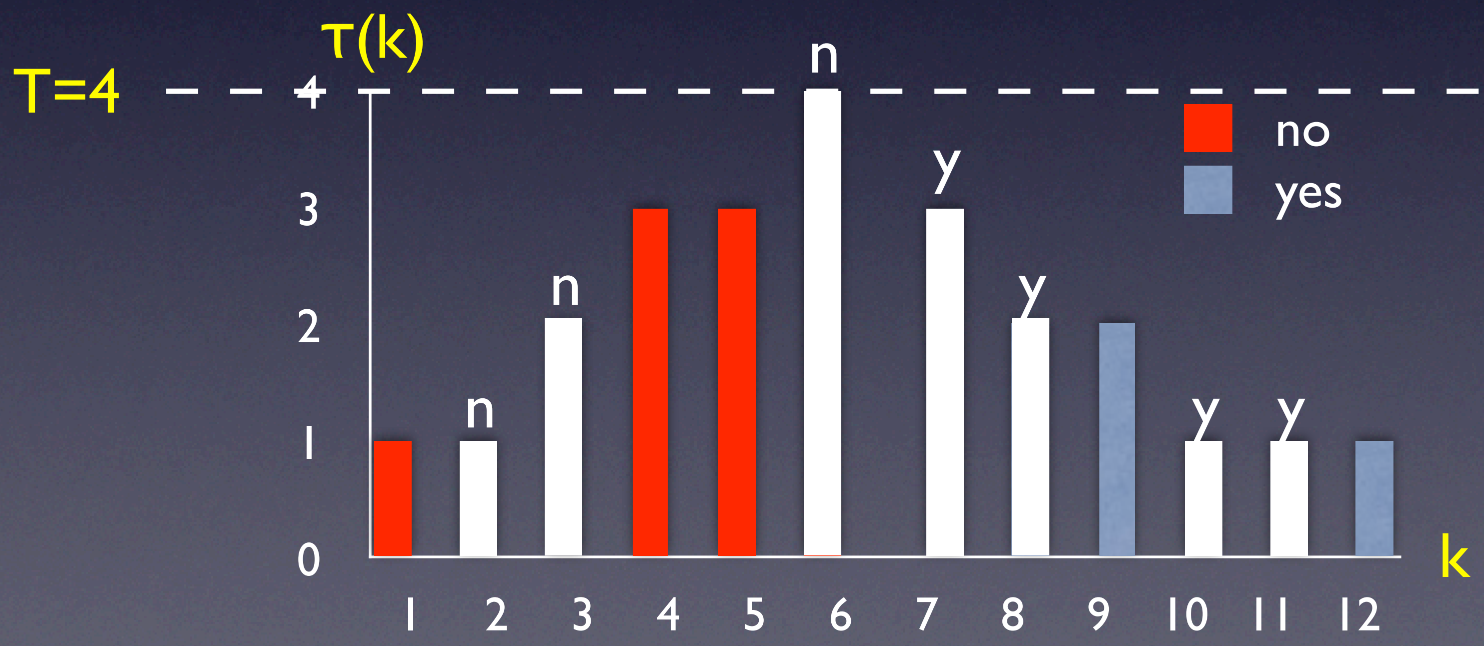
Query strategy S_2

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of k-values such that $\tau(k) > T$
- Double T and repeat



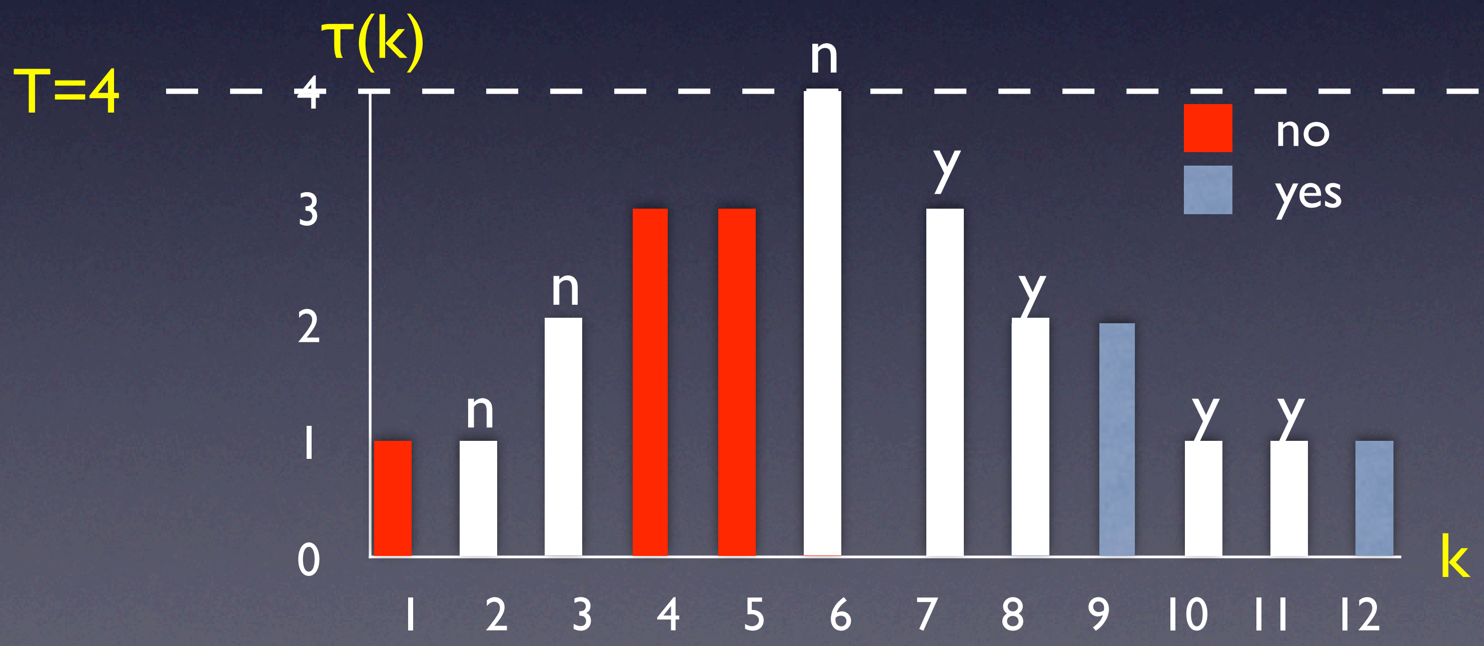
Query strategy S_2

- **Theorem:** if $\tau(k)$ is increasing-then-decreasing, then S_2 has competitive ratio $O(\log \#(\text{possible } k\text{-values}))$



Query strategy S_2

- **Theorem:** if $\tau(k)$ is increasing-then-decreasing, then S_2 has competitive ratio $O(\log \#(\text{possible k-values}))$
- If $\tau(k)$ becomes increasing-then-decreasing after multiplying each $\tau(k)$ by a factor $\leq \Delta$, ratio goes up by factor $\leq \Delta$



Experiments: planning

- Created modified version of SATPLAN that uses S_2.
- Ran both versions on benchmarks from ICAPS’06 planning competition, one hour time limit per benchmark
- Also tried geometric strategy S_g based on Rintanen (2004)

Experiments: planning

Results on instances from *pathways* domain

Instance	SATPLAN (S_2) [lower,upper]	SATPLAN (geom.) [lower,upper]	SATPLAN (orig.) [lower,upper]
p1	[5,5]	[5,5]	[5,5]
p2	[7,7]	[7,7]	[7,7]
p3	[8,8]	[8,8]	[8,8]
. . .	. . .	. . .	. . .
p27	[19,34]	[20,31]	[20,∞]
p28	[19,27]	[20,∞]	[21,∞]
p29	[19,29]	[18,29]	[18,∞]
p30	[20,60]	[21,∞]	[21,∞]

Experiments: scheduling

- We next used S_2 in a branch and bound algorithm for job shop scheduling (Brucker et al. 1994).
- Here we execute query (k,t) by setting upper bound to $k+1$ and seeing if problem is feasible.

Experiments: scheduling

Results on instances from OR Library

Instance	Brucker (S_2) [lower,upper]	Brucker (orig.) [lower,upper]
abz7	[650,712]	[650,726]
abz8	[622,725]	[597,767]
abz9	[644,728]	[616,820]
...	...	...
yn1	[813,987]	[763,992]
yn2	[835,1004]	[795,1037]
yn3	[812,982]	[793,1013]
yn4	[899,1158]	[871,1178]

Questions?