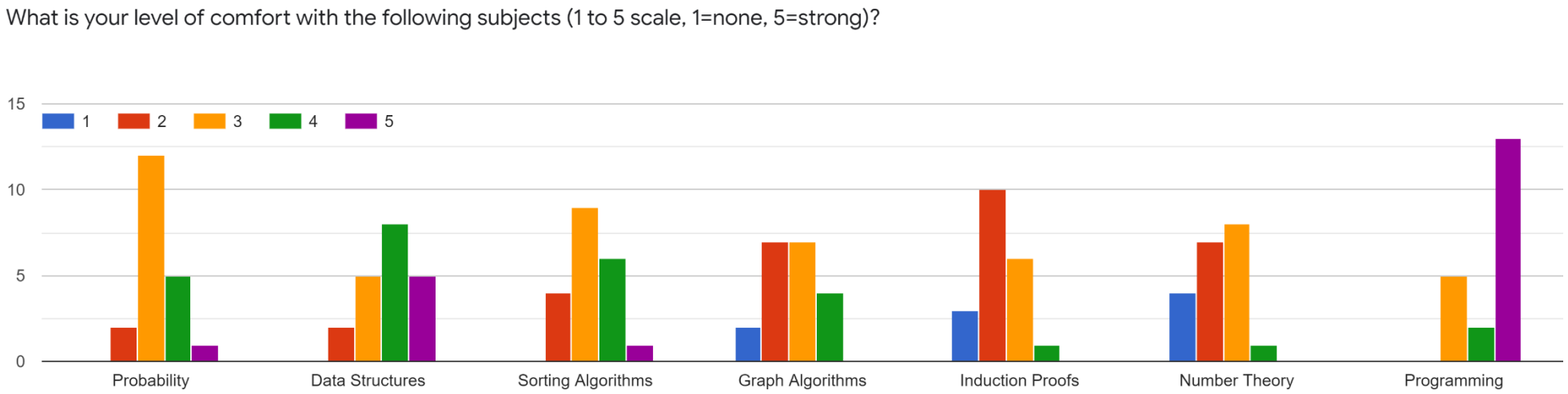
CSEP 521: Applied Algorithms
Lecture 4 Randomized Algorithms

Richard Anderson

January 14, 2021

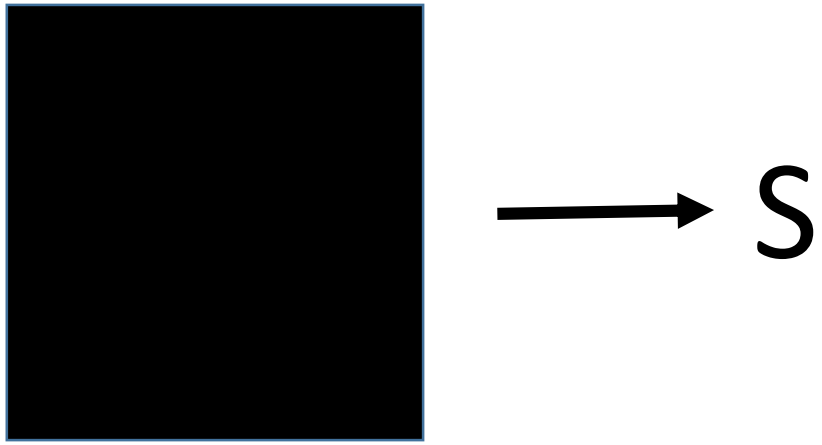
Announcements

- Homework deadline shifted to Thursday (including HW 1)
  - But distribution will remain Tuesdays (HW 2 has been posted)
- Sign up for Gradescope (?)
- Shift in office hours – Oscar: 5-6 pm, Monday and Friday



Randomized Algorithms

- How can random choices be used to make algorithms efficient?
- Now suppose you have an algorithm, which returns a specific optimal solutions with probability p.



- Mincut result – success with probability $1/n^2$
- Amplification: Probability of failing k times is $(1-p)^k$

Using randomization to remove structure

- Many problems perform well on random data, but specific well structured data can cause poor behavior
  - Example, binary search trees
- In practice, bad case data can be natural
  - Sorting nearly ordered lists
  - Planar partition for parallel line segments
- Randomization can very likely hide the bad cases
- Prefer to deal with random data than a deterministic adversary

Binary planar partition

- S = {s_1, s_2, . . . s_n}: non-intersecting line segments in the plane

- Binary tree, each node v of the tree is a region r(v) of the plane and has a line l(v) which separates r(v) into regions r_1(v) and r_2(v) which are associated with it children
- The line segments are then cut up and assigned to the associated leaf regions

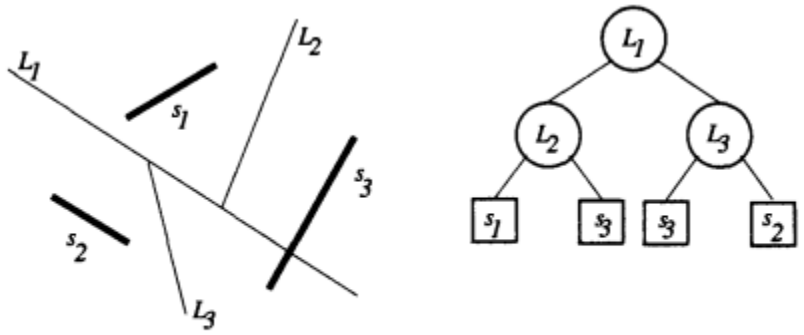
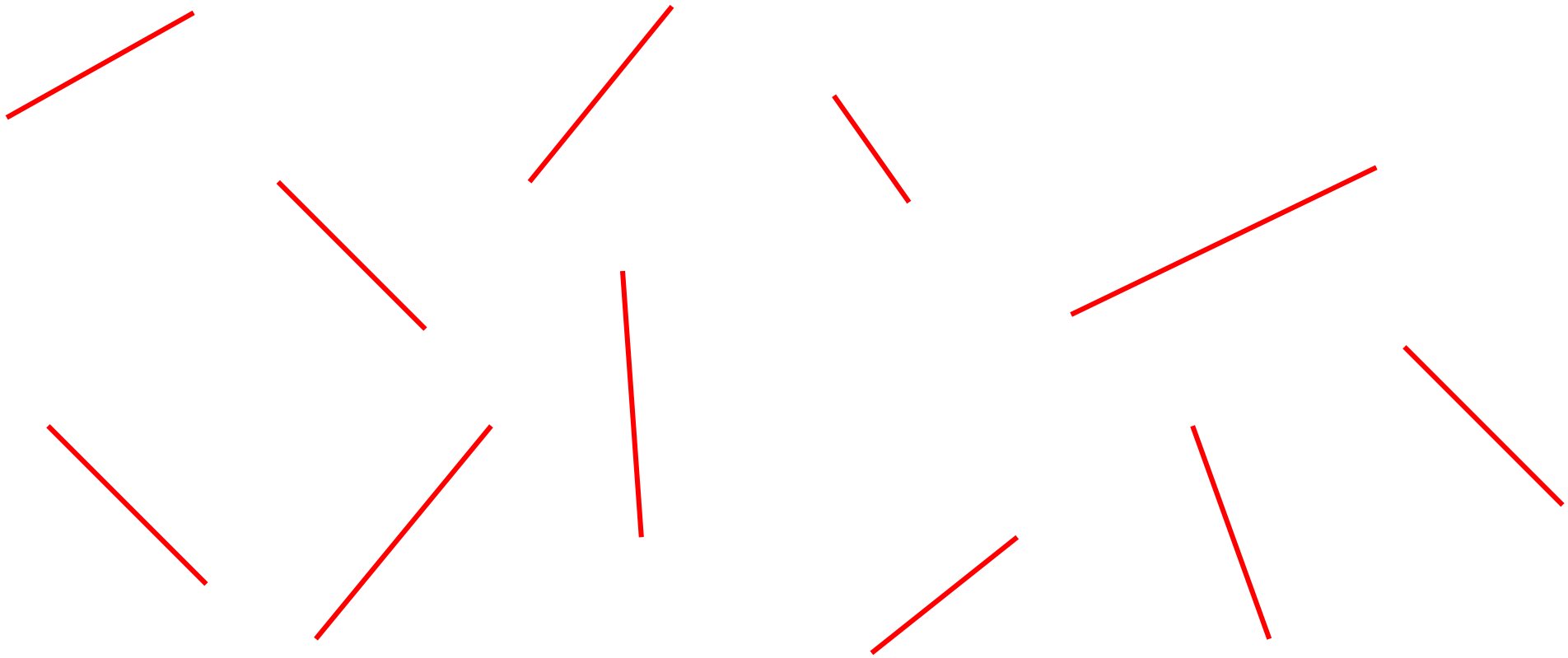
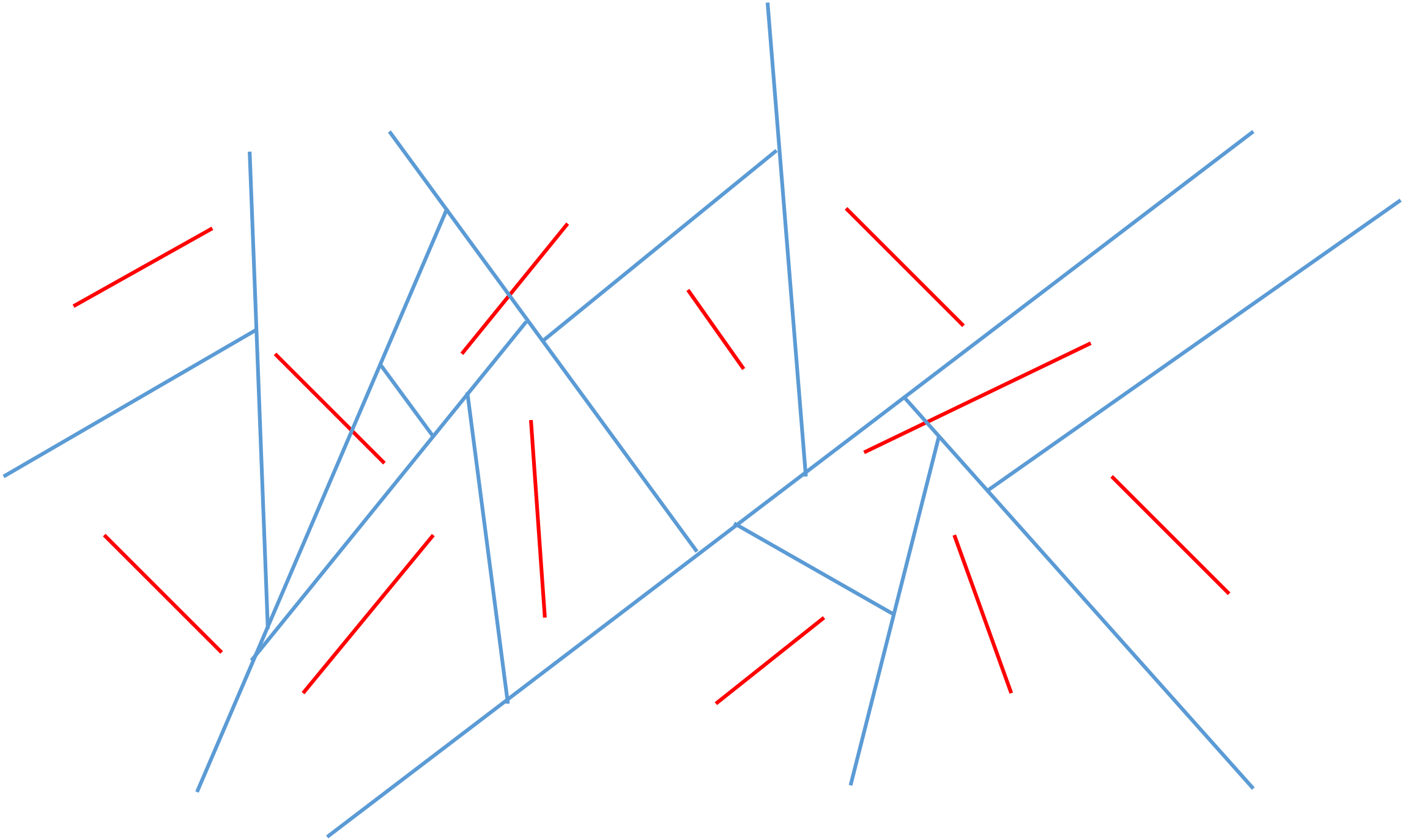
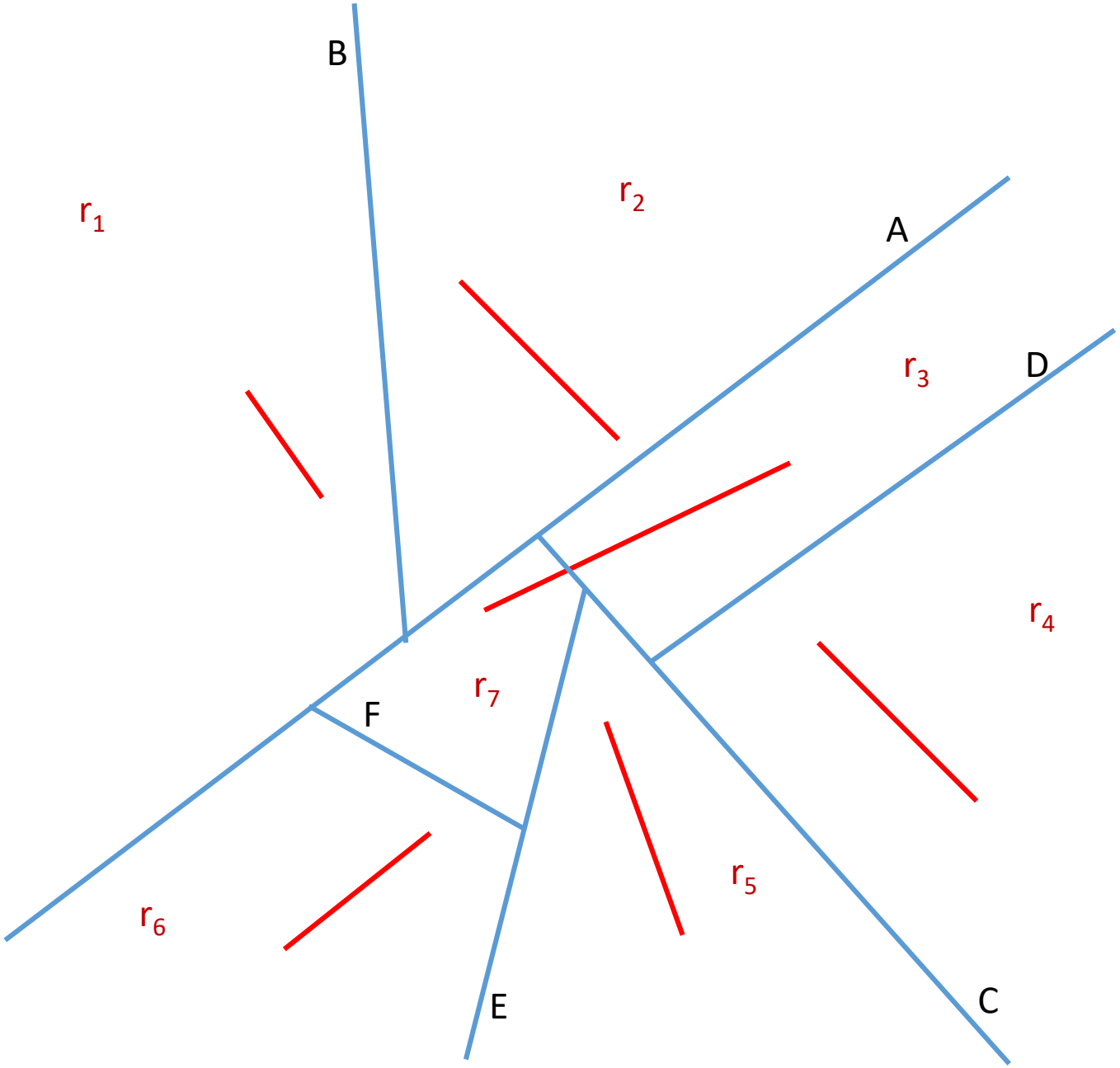


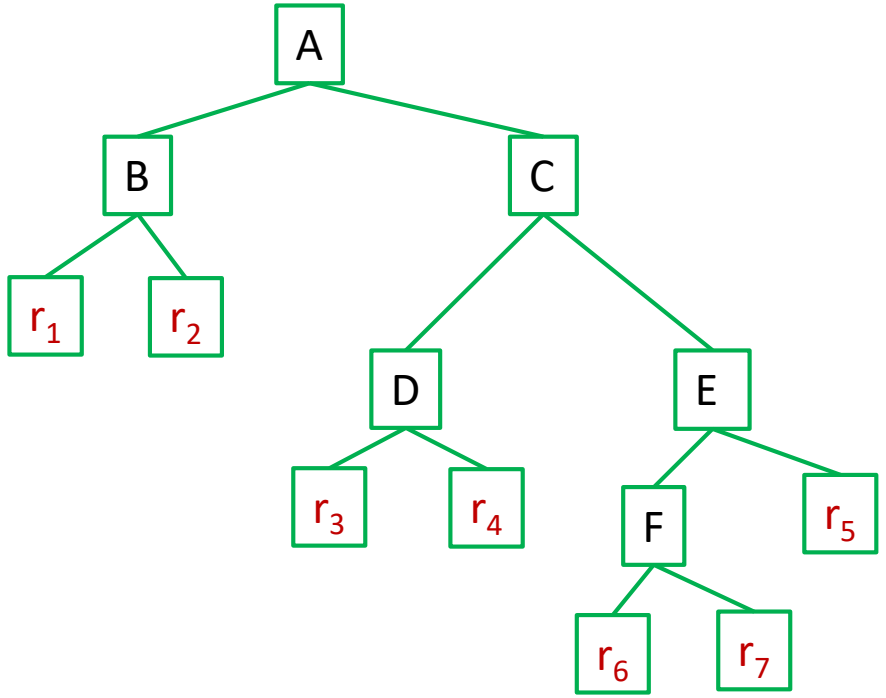
Figure 1.2: An example of a binary planar partition for a set of segments (dark lines). Each leaf is labeled by the line segment it contains. The labels $r(v)$ are omitted for clarity.

Binary planar partition tree

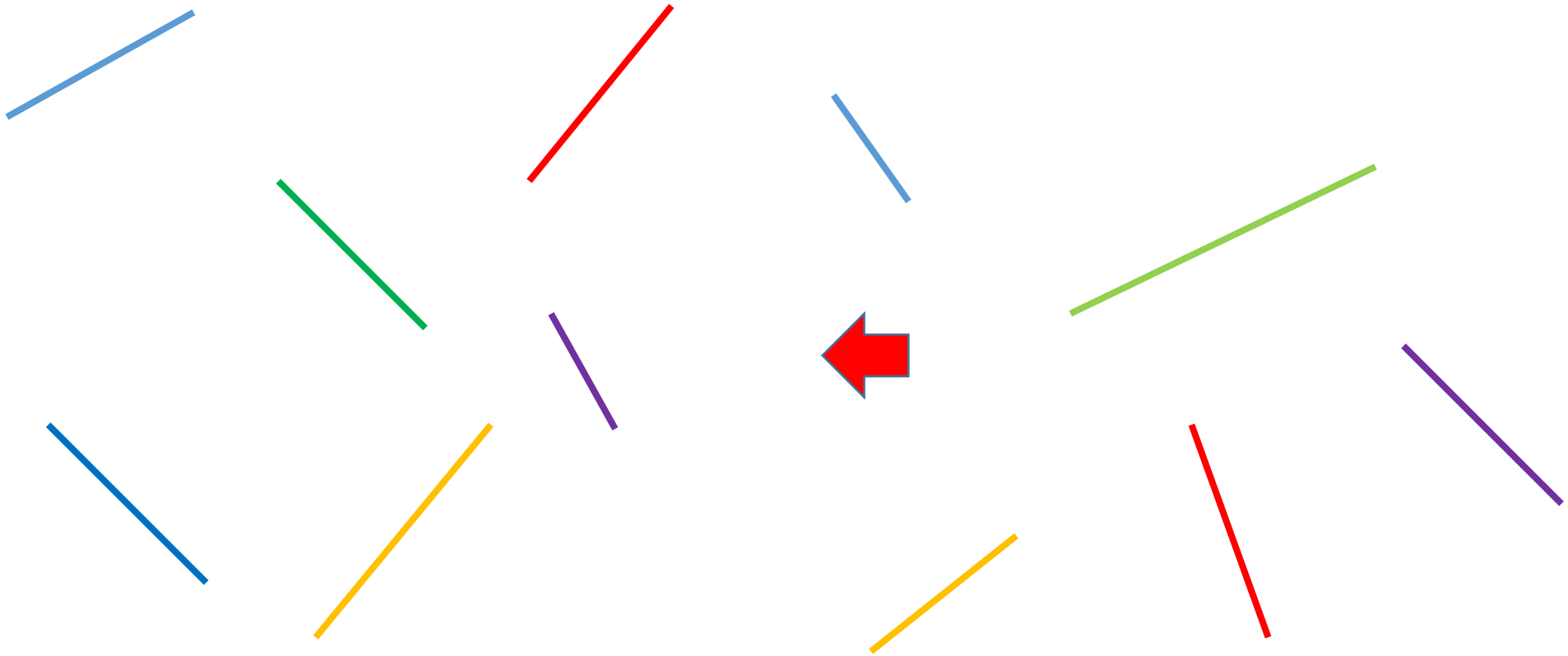




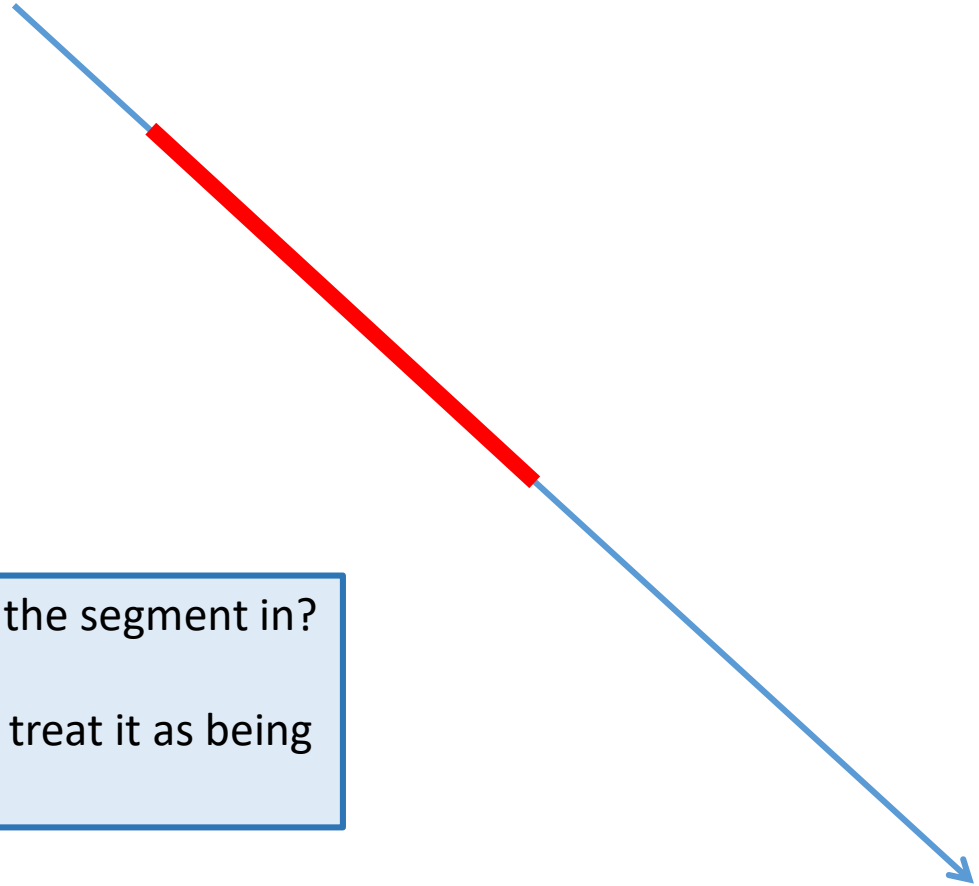

B
A
D
F
E
C
r1
r2
r3
r4
r5
r6
r7


A
B
C
D
E
F
r1
r2
r3
r4
r5
r6
r7

Hidden line removal

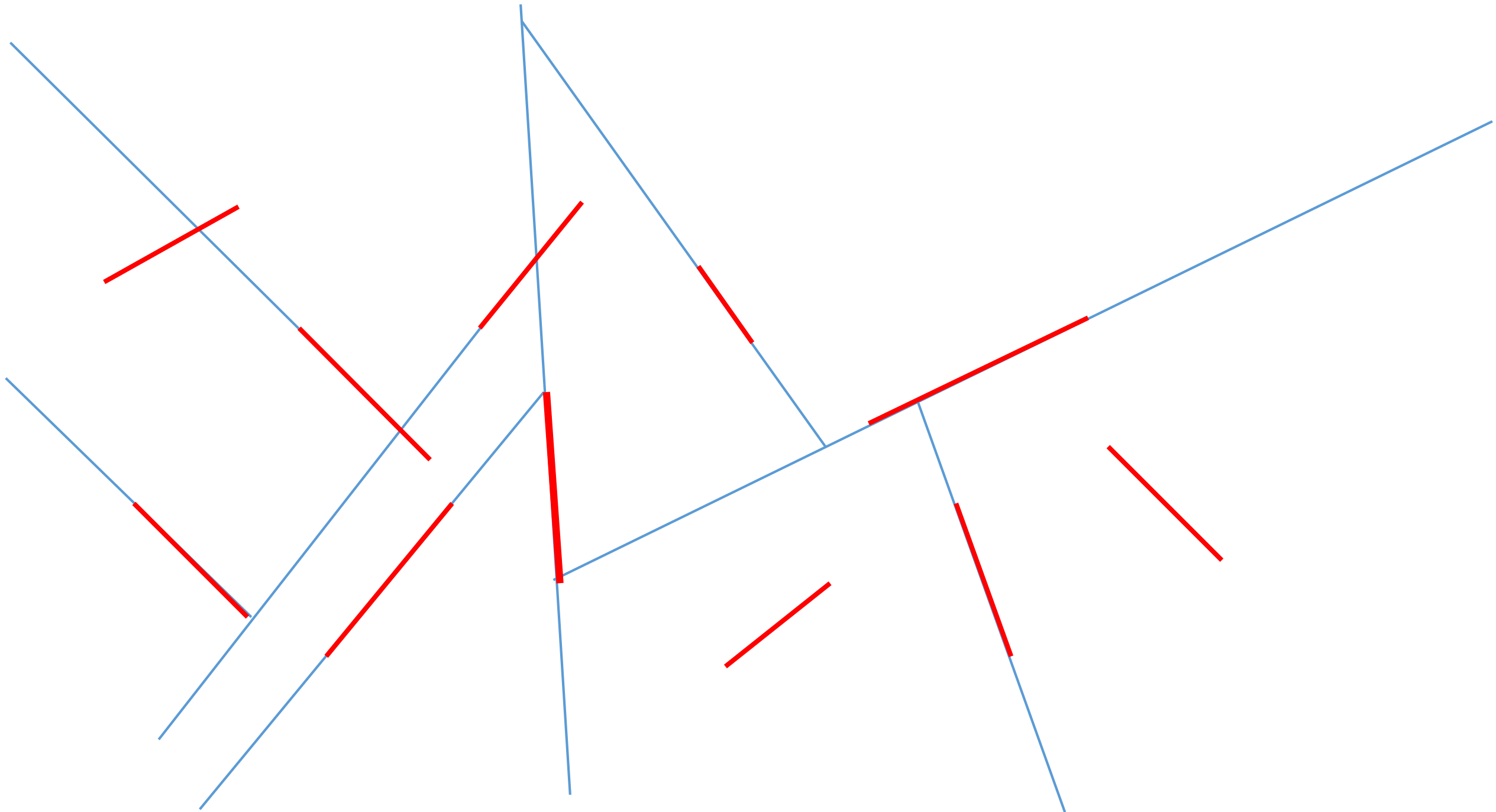


Use lines which are extension of line segments



Detail: Which region do we put the segment in?

Place it in an interior node, and treat it as being between regions



Motivation: hidden surface elimination

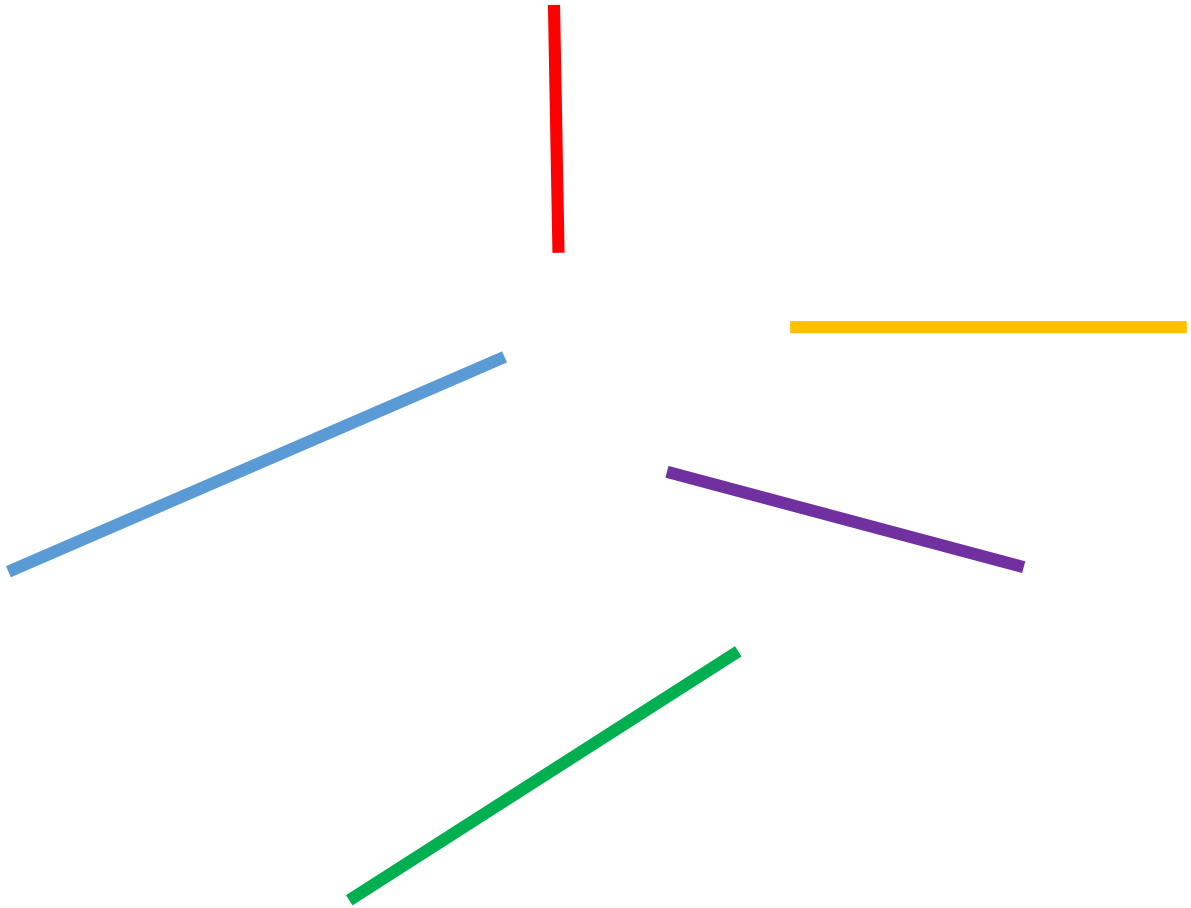
- 2-D Version, draw the line segments visible from any point in space
  - Painters algorithm: draw the lines from back to front in viewing direction

- Tree traversal algorithm. Determine which side of the line you are on for each node. Traverse the subtree on the other side of the line, then the subtree on your side of the line

- Run time is proportional to the size of the tree
  - Tree size O(n)? O(n log n)? $O(n^2)$?

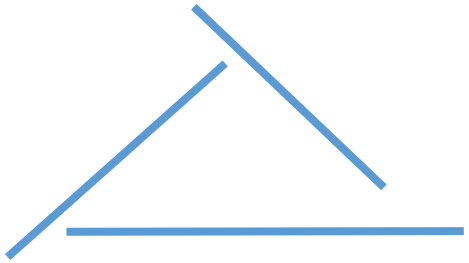
Greedy Algorithm

- Permutation of line segments
- While a region has more than one segment
  - Choose the next segment, and use its line to split segments

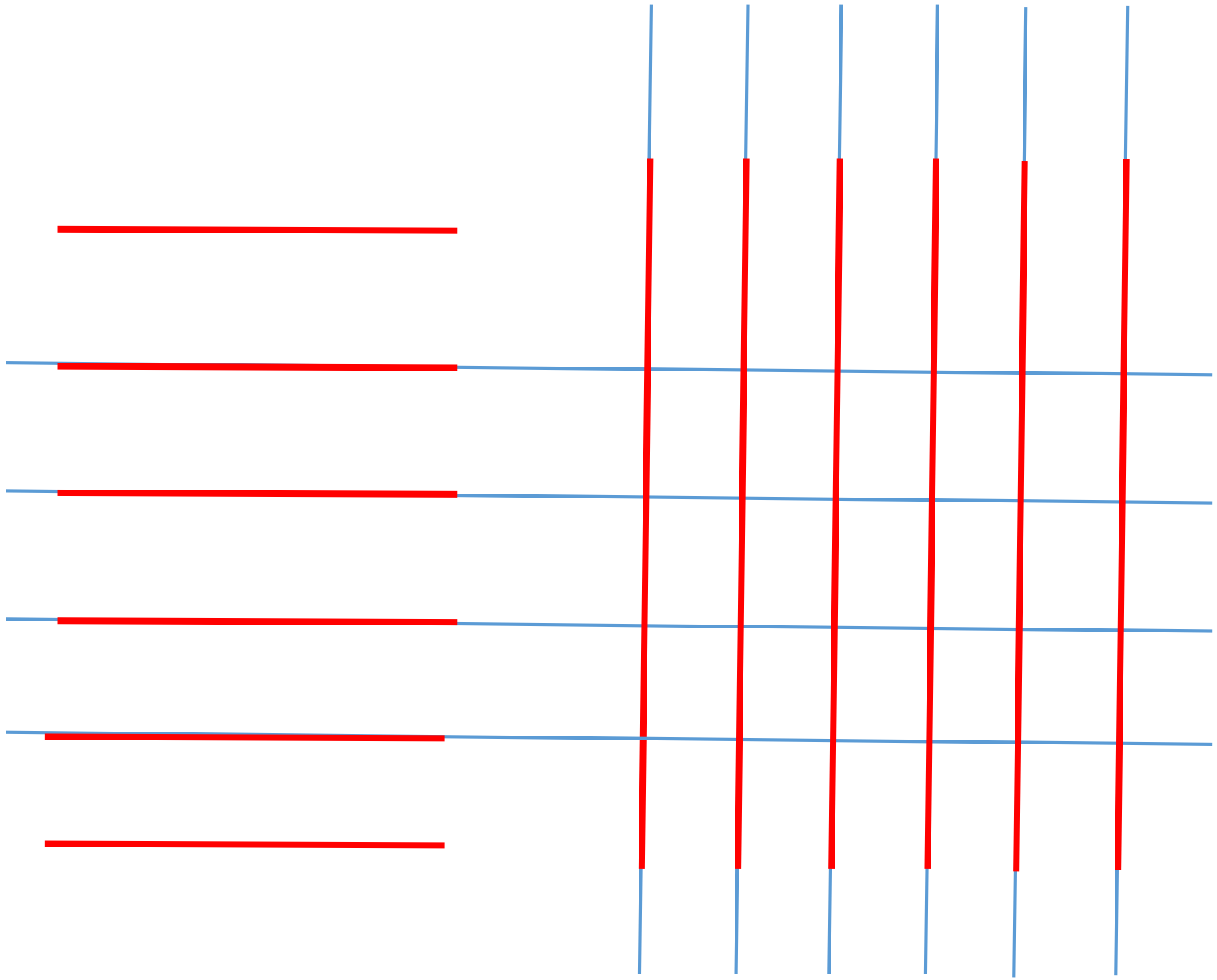


Warmup exercises (breakout)

- Show that there exists a set of line segments for which no binary planar partition can avoid breaking up some of the segments into pieces, if each segment is to lie in a different region of the partition



- What is a bad case of partitioning

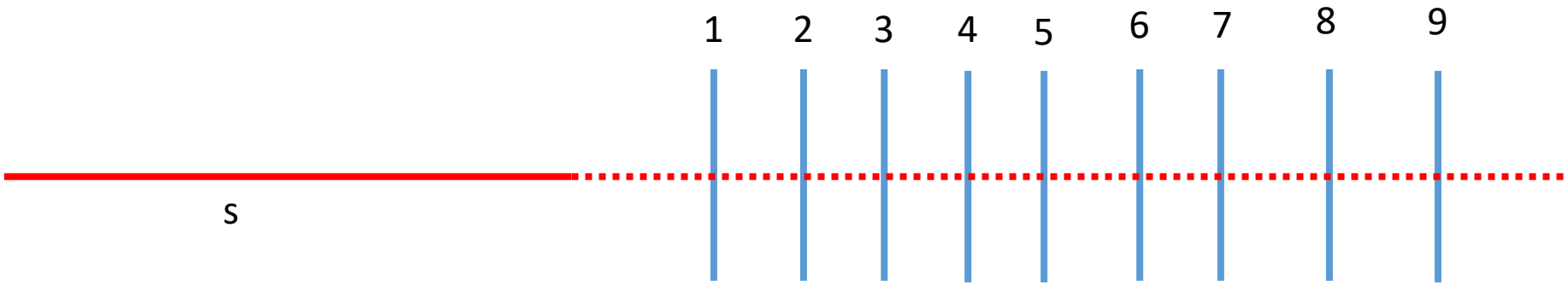


Random algorithm

- Choose the segments in a random order
- Result: Expected number of partitions is O(n log n)

Computing the expected size

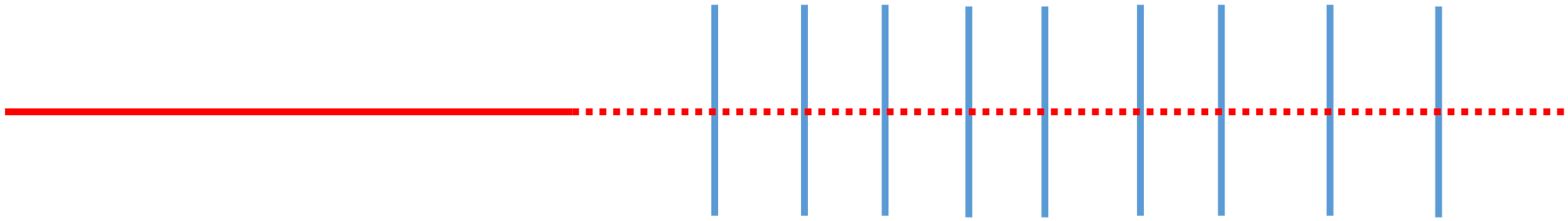
- Expected size is given by the number of expected pieces of segments
- How many intersections are generated by extensions of the segment s



Summary of result

- Binary Space Partition trees are small with random ordering
- This result generalizes to higher dimensions where it is useful for hidden surface elimination
  - Leads to simple algorithms that perform well

- Use of randomization to defeat the bad case



Breakout question

- Suppose that you have an application that requires unique IDs for data elements, and you generate 128 bit keys by a random
- Does this make your application a randomized/probabilistic process?
- What are the failure modes?
- How would you analyze the probability of failure?
- What is an acceptable probability of failure?

Primality Testing

- Miller-Rabin test demonstrated importance of randomized algorithm
  - Break through result in 1980
- Depends on number theory (maybe a senior ugrad class)
  - But much of the algorithm can be appreciated without the theory
- Key concept is that of a witness
  - If something is true, a witness always says TRUE
  - If something is false, a witness says TRUE with probability less than ½

Primality testing

Is the number:
2380954130912091008943057093740928439084380298439034832098439302843092383409843043984309384092843890342938420948092983249898432902309423843290342893248903484320894328320497173009908098082389032481098473871209889192984589393000949591000499585901209438732409428390324988324923939
Prime?

- A number p is prime if its only proper divisors are 1 and p, and is composite otherwise
- Small primes {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, . . .}
- Simple primality testing algorithms
  - Trial division
  - Sieve of Eratosthenes

Bignum computation

- Arithmetic computation on very large numbers – thousands of digits
- Run time expressed as a function of the number of digits
- Addition of two N-bit numbers: O(n)
- Multiplication of two n-bit numbers: $O(n^2)$ or $O(n^{3/2})$ or O(n log n loglog n)
- Primality testing algorithms (testing if n is prime), generally perform operations mod n
- Computing a^x mod n can be done with O(log x) multiplications

Proving numbers are prime

- Important to do this for cryptography
- Central component for RSA cryptography – number N = pq where p and q are secret primes
- Need a way to generate “random primes”

Witnesses and Certificates

- Certificate C that can be used to prove a property
  - To show N is composite, find a number A such that $1 < GCD(A, N) < N$
  - 178 is a Certificate that 11481 is composite
- Is there a certificate for primality?
- Witness
  - A property that always holds for primes
  - A property that only sometimes holds for composites

Miller-Rabin test

- Determine if n is prime
- Given an integer a, $1 < a < n$,
  - Miller(n, a) returns either “maybe prime” or “definitely composite”
  - For n prime, Miller(n, a) always says “maybe prime”
  - For n composite, Miller(n, a) says “maybe prime” with probability at most ¼ for a random a
- By running the Miller test repeatedly, we can make it arbitrary high probability

Fermat Test

- Fermat's little theorem
  - For prime n, $a^{(n-1)} = 0 \pmod{n}$ for all a

- For most composite numbers, this fails most of the time

- Unfortunately, there are set of composite numbers (Carmichael numbers) that satisfy this
  - {561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973, 75361, 101101, 115921, 126217, 162401, 172081, 188461, 252601, 278545, 294409, 314821, 334153, ...}

Miller-Rabin test

- For a prime number n, the only square roots of 1 modulo n, are 1 and -1
- For $n = 2^s d + 1$, $a^d = 1 \pmod n$ or $a^{(2^r)d} = -1 \pmod n$ for some $0 <= r < s$

- For a composite number at most ¼ of values a satisfy the conditions

Pseudo-code

Input #1: n > 3, an odd integer to be tested for primality
Input #2: k, the number of rounds of testing to perform
Output: “composite” if n is found to be composite, “probably prime” otherwise

write n as $2^r \cdot d + 1$ with d odd (by factoring out powers of 2 from n − 1)
WitnessLoop: repeat k times:
    pick a random integer a in the range [2, n − 2]
    x ← ad mod n
    if x = 1 or x = n − 1 then
        continue WitnessLoop
    repeat r − 1 times:
        x ← x2 mod n
        if x = n − 1 then
            continue WitnessLoop
    return “composite”
return “probably prime”

Other facts on Prime Testing

- Miller-Rabin test is deterministic if Extended Riemann Hypothesis is true
- 2002 a deterministic polynomial time test based on Cyclotomic Polynomials was discovered
  - Agrawal-Kayal-Saxena, IIT Kanpur
  - Not practical (termed galactic algorithm – see Wikipedia)
- Factoring is thought to be harder then primality testing
  - In practice, numbers of about 100 decimal digits are factorable in a few hours on a PC
  - 250 decimal digit (829 bit) RSA keys have been factored (2700 CPU Years)
  - Recommendation for RSA is 2048 bit keys

RSA

- RSA key is a number n=pq, where p and q are prime
- How do you generate random primes of 300 digits?
- Generate random number of 300 digits and test if they are prime
  - Of course, there are simple tricks to avoid small divisors
- Prime number theorem: Probability of a random number less than N is prime is about 1/log N (Natural logarithm)
- For 300 digits, this is about 690