UBCPC 2020 Solution Slides

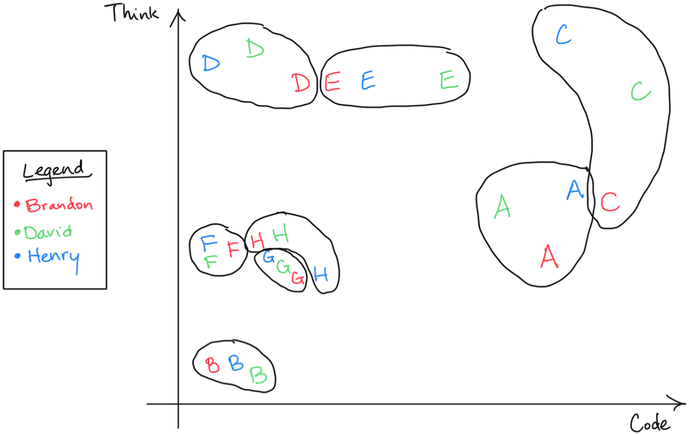
UBC Programming Contest 2020

Problem Writers: Henry Xia and David Zheng
Testers: Brandon Zhang, Andrew Lin, Paul Liu, Angus Lim, Nick Wu, and Sam Reinehr

2020/10/24

University of British Columbia

Think-Code Chart



Problem B: Big Boxes

Input: An array of n numbers $[w_1, w_2, \ldots, w_n]$, and an integer k.

Goal: Partition the n numbers into k consecutive groups. Let the of a group be the sum of the numbers in it. We want to minimize the maximum size of the groups.

Limits:

- $n \leq 10^5$

107 solves / 217 attempts (49%)

Problem B: Big Boxes

Solution: Binary search on the answer.

If we know the maximum allowed size of each group, then we can greedily determine whether it is achievable.

Problem B: Big Boxes

Solution: Binary search on the answer.

If we know the maximum allowed size of each group, then we can greedily determine whether it is achievable.

Time complexity: $O(n \log(\sum_{i=1}^{n} w_i))$

Problem B: Big Boxes

Solution: Binary search on the answer.

If we know the maximum allowed size of each group, then we can greedily determine whether it is achievable.

Time complexity: $O(n \log (\sum_{i=1}^{n} w_i))$

— *you don't need a quote for every problem. You're not going to get a quote for B*

Problem F: Find the Graph

Input: An oracle which lets us query the size of a cut in a graph with n vertices.

Goal: Recover the graph with less than 3000 calls to the oracle.

Limits:

- $n \leq 50$

90 solves / 214 attempts (42%)

Problem F: Find the Graph

Denote the cut of a set S by $d(S)$. Consider each potential edge one at a time.

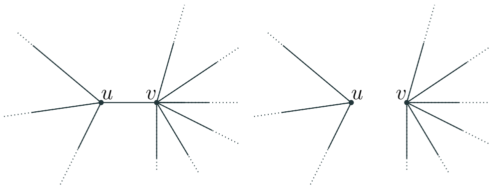
Solution:

Problem F: Find the Graph

Denote the cut of a set S by $d(S)$. Consider each potential edge one at a time.

Solution:

- Compute $d(u)$, $d(v)$ and $d(u, v)$.
  This is enough to figure out if there is an edge between u and v.

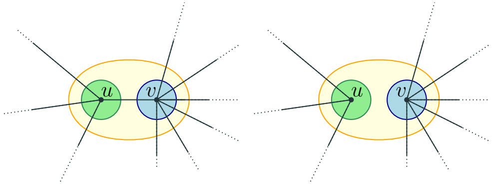


Problem F: Find the Graph

Denote the cut of a set S by $d(S)$. Consider each potential edge one at a time.

Solution:

- Compute $d(u)$, $d(v)$ and $d(u, v)$.
  This is enough to figure out if there is an edge between u and v.
- If there's an edge $d(u) + d(v) = d(u, v) + 2$. Otherwise $d(u) + d(v) = d(u, v)$.

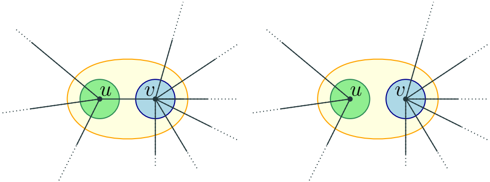


Problem F: Find the Graph

Denote the cut of a set S by $d(S)$. Consider each potential edge one at a time.

Solution:

- Compute $d(u)$, $d(v)$ and $d(u, v)$.
  This is enough to figure out if there is an edge between u and v.
- If there's an edge $d(u) + d(v) = d(u, v) + 2$. Otherwise $d(u) + d(v) = d(u, v)$.



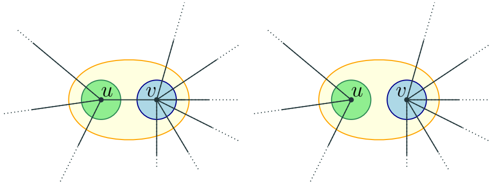
Complexity: $O(n^2)$, specifically $n + \frac{n(n-1)}{2}$ queries. Lower bound $\Omega\left(\frac{n^2}{\log n}\right)$.

Problem F: Find the Graph

Denote the cut of a set S by $d(S)$. Consider each potential edge one at a time.

Solution:

- Compute $d(u)$, $d(v)$ and $d(u, v)$.
  This is enough to figure out if there is an edge between u and v.
- If there's an edge $d(u) + d(v) = d(u, v) + 2$. Otherwise $d(u) + d(v) = d(u, v)$.



Complexity: $O(n^2)$, specifically $n + \frac{n(n-1)}{2}$ queries. Lower bound $\Omega\left(\frac{n^2}{\log n}\right)$.

— *understanding my code is probably harder than solving the question lmao*

Problem G: Grid Magic

Input: The dimensions of an n by m grid.

Goal: Count the number of super-prime grids with given the dimensions.

Limits:

- $n, m \leq 8$

86 solves / 130 attempts (66%)

Problem G: Grid Magic

Input: The dimensions of an n by m grid.

Goal: Count the number of super-prime grids with given the dimensions.

Limits:

- $n, m \leq 8$

86 solves / 130 attempts (66%)

— literally my plan for G was, write the brute force, run it while I go buy dinner. So I ran it and it printed everything instantly

Problem G: Grid Magic

Python solution:

```python
ans = [
    [4,  4,  3,  3,  2,  0,  0,  0],
    [4,  9,  5,  0,  0,  0,  0,  0],
    [3,  5, 16,  0,  0,  0,  0,  0],
    [3,  0,  0,  0,  0,  0,  0,  0],
    [2,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0],
    [0,  0,  0,  0,  0,  0,  0,  0],
]
n, m = [int(x) for x in input().split()]
print(ans[n-1][m-1])
```

Problem H: Huge Campus

Input: There are n days, k buildings, and two umbrellas.

On day i, a person travels from his home to building a_i in the morning, b_i at noon, and back home in the evening.

The weather in the morning, at noon, and in the evening are known to be either sunny or rainy on each of the days.

Goal: Compute the minimum number of trips with at least one umbrella taken.

Limits:

- $n \leq 10^4$
- $k \leq 30$

63 solves / 101 attempts (62%)

Problem H: Huge Campus

Input: There are n days, k buildings, and two umbrellas.

On day i, a person travels from his home to building a_i in the morning, b_i at noon, and back home in the evening.

The weather in the morning, at noon, and in the evening are known to be either sunny or rainy on each of the days.

Goal: Compute the minimum number of trips with at least one umbrella taken.

Limits:

- $n \leq 10^4$
- $k \leq 30$

63 solves / 101 attempts (62%)

— *statement needs some clarifying, eg. you must carry an umbrella when it is raining*

Problem H: Huge Campus

Observation: The problem only depends on the location of the two umbrellas and the day.

Problem H: Huge Campus

Observation: The problem only depends on the location of the two umbrellas and the day.

Solution:

- Dynamic program on time, day, and locations of the two umbrellas.
- On each day if at the location of one or more umbrella, try taking them or not.
- Value of ∞ if it is raining and there is no umbrella!

Problem H: Huge Campus

Observation: The problem only depends on the location of the two umbrellas and the day.

Solution:

- Dynamic program on time, day, and locations of the two umbrellas.
- On each day if at the location of one or more umbrella, try taking them or not.
- Value of ∞ if it is raining and there is no umbrella!

Time complexity: $O(nk^2)$.

Problem D: Drowning Combinatorist

Input: Integers n and k.

Goal: Count the number of permutations of length n with "runs" of size at most k.

Limits:

- $n \leq 2\,000$
- $k \leq 7$

25 solves / 46 attempts (54%)

Problem D: Drowning Combinatorist

Input: Integers n and k.

Goal: Count the number of permutations of length n with "runs" of size at most k.

Limits:

- $n \leq 2\,000$
- $k \leq 7$

25 solves / 46 attempts (54%)

Example: $n = 3, k = 2$

[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]

~~[1, 2, 3]~~
~~[3, 2, 1]~~

Problem D: Drowning Combinatorist

We can “build” the permutation element by element using dynamic programming.

Problem D: Drowning Combinatorist

We can “build” the permutation element by element using dynamic programming.

Let $f(i, r, s, inc?)$ = the number of ways to fill the last i entries of the permutation, where

- the size of the current run on the left is r,
- the number of remaining elements smaller than the previous element is s, and
- $inc?$ is 1 if the current run is increasing. (Can get rid of this state with symmetry.)

Answer is $f(n, 0, 0, 1)$.

Problem D: Drowning Combinatorist

We can "build" the permutation element by element using dynamic programming.

Let $f(i, r, s, inc?)$ = the number of ways to fill the last i entries of the permutation, where

- the size of the current run on the left is r,
- the number of remaining elements smaller than the previous element is s, and
- $inc?$ is 1 if the current run is increasing. (Can get rid of this state with symmetry.)

Answer is $f(n, 0, 0, 1)$.

For the transitions, try to place all possible elements in the leftmost position. This either extends the current run, or starts a new one.

Problem D: Drowning Combinatorist

For example,

$$f(i,r,s,1) = \underbrace{\sum_{j=s}^{i-1} f(i-1,r+1,j,1)}_{\text{continue run upwards}} + \underbrace{\sum_{j=0}^{s-1} f(i-1,2,j,0)}_{\text{start downwards run}}$$

(and similarly for $f(i,r,s,0)$).

Problem D: Drowning Combinatorist

For example,

$$f(i, r, s, 1) = \underbrace{\sum_{j=s}^{i-1} f(i-1, r+1, j, 1)}_{\text{continue run upwards}} + \underbrace{\sum_{j=0}^{s-1} f(i-1, 2, j, 0)}_{\text{start downwards run}}$$

(and similarly for $f(i, r, s, 0)$).

Speed up the transition to $O(1)$ by computing the prefix sum of $f(i, r, \cdot, inc?)$.

Time complexity: $O(n^2 k)$.

Problem D: Drowning Combinatorist

Alternate Solution:

Consider placing the largest element in the permutation. This partitions the permutation into two independent parts.

Problem D: Drowning Combinatorist

Alternate Solution:

Consider placing the largest element in the permutation. This partitions the permutation into two independent parts.

Let $f(n, l, r)$ be the number of valid permutations of length n, where the decreasing run to the left already has length l and the increasing run to the right already has length r.

Problem D: Drowning Combinatorist

Alternate Solution:

Consider placing the largest element in the permutation. This partitions the permutation into two independent parts.

Let $f(n, l, r)$ be the number of valid permutations of length n, where the decreasing run to the left already has length l and the increasing run to the right already has length r.

The transitions are roughly

$$f(n, l, r) = f(n-1, l, r+1) + f(n-1, l+1, r) + \sum_{j=2}^{n-1} f(j-1, l, 1) \cdot f(n-j, 1, r) \cdot \binom{n-1}{j-1}.$$

Problem D: Drowning Combinatorist

Alternate Solution:

Consider placing the largest element in the permutation. This partitions the permutation into two independent parts.

Let $f(n, l, r)$ be the number of valid permutations of length n, where the decreasing run to the left already has length l and the increasing run to the right already has length r.

The transitions are roughly

$$f(n, l, r) = f(n-1, l, r+1) + f(n-1, l+1, r) + \sum_{j=2}^{n-1} f(j-1, l, 1) \cdot f(n-j, 1, r) \cdot \binom{n-1}{j-1}.$$

Time complexity: $O(n^2 k^2)$.

Problem D: Drowning Combinatorist

Alternate Solution:

Consider placing the largest element in the permutation. This partitions the permutation into two independent parts.

Let $f(n, l, r)$ be the number of valid permutations of length n, where the decreasing run to the left already has length l and the increasing run to the right already has length r.

The transitions are roughly

$$f(n, l, r) = f(n-1, l, r+1) + f(n-1, l+1, r) + \sum_{j=2}^{n-1} f(j-1, l, 1) \cdot f(n-j, 1, r) \cdot \binom{n-1}{j-1}.$$

Time complexity: $O(n^2 k^2)$.

— *you made me think about alternating permutations non-stop for a week*

Problem E: Eerie Subarrays

Input: An array A of length n with distinct elements.

Goal: Count the number of subarrays whose leftmost element is equal to its median.

Limits:

- $n \leq 2 \cdot 10^5$

13 solves / 86 attempts (15%)

Problem E: Eerie Subarrays

Idea: For each index i, we count the number of scary subarrays that start at i.

Idea: For each index i, we count the number of scary subarrays that start at i.

Attempt 1: For each index i, iterate over the rest of the array and keep track of the number of elements less than or greater than a_i.

Too Slow!

Problem E: Eerie Subarrays

Observation 1: If we define

$$b_{i,j} = \begin{cases} -1 & \text{if } a_j < a_i \\ 1 & \text{otherwise,} \end{cases}$$

then the number of scary subarrays starting at i is equal to the number of indices $k > i$ such that

$$\sum_{j=i+1}^{k} b_{i,j} = 0.$$

Problem E: Eerie Subarrays

Observation 1: If we define

$$b_{i,j} = \begin{cases} -1 & \text{if } a_j < a_i \\ 1 & \text{otherwise,} \end{cases}$$

then the number of scary subarrays starting at i is equal to the number of indices $k > i$ such that

$$\sum_{j=i+1}^{k} b_{i,j} = 0.$$

Observation 2: Let $s_{i,k} = \sum_{j=1}^{k} b_{i,j}$, then the above condition is equivalent to $s_{i,k} = s_{i,i}$.

Problem E: Eerie Subarrays

Observation 3: Let B_i be the array $[b_{i,1}, b_{i,2}, \ldots, b_{i,n}]$. Let x_i be the index of i-th largest element in the original array A. Then B_{x_i} only differs from $B_{x_{i+1}}$ at index x_i.

Problem E: Eerie Subarrays

Observation 3: Let B_i be the array $[b_{i,1}, b_{i,2}, \ldots, b_{i,n}]$. Let x_i be the index of i-th largest element in the original array A. Then B_{x_i} only differs from $B_{x_{i+1}}$ at index x_i.

Observation 4: Let S_i be the array $[s_{i,1}, s_{i,2}, \ldots, s_{i,n}]$. Then we can convert S_{x_i} to $S_{x_{i+1}}$ by decreasing each $s_{x_i,j}$ by 2 for each $j \geq x_i$.

Problem E: Eerie Subarrays

Observation 3: Let B_i be the array $[b_{i,1}, b_{i,2}, \ldots, b_{i,n}]$. Let x_i be the index of i-th largest element in the original array A. Then B_{x_i} only differs from $B_{x_{i+1}}$ at index x_i.

Observation 4: Let S_i be the array $[s_{i,1}, s_{i,2}, \ldots, s_{i,n}]$. Then we can convert S_{x_i} to $S_{x_{i+1}}$ by decreasing each $s_{x_i,j}$ by 2 for each $j \geq x_i$.

Solution: Use square root decomposition to handle these range updates and range queries.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.
2. Divide S into $\sqrt{n}$ blocks of similar size, for each block, maintain the number of times each value appears with an offset.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.
2. Divide S into $\sqrt{n}$ blocks of similar size, for each block, maintain the number of times each value appears with an offset.
   Observe that the number of values that can appear in each block is bounded by the size of the block, because adjacent elements in S differ by exactly 1.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.
2. Divide S into $\sqrt{n}$ blocks of similar size, for each block, maintain the number of times each value appears with an offset.
   Observe that the number of values that can appear in each block is bounded by the size of the block, because adjacent elements in S differ by exactly 1.
3. Updates and queries can be done in $O(\sqrt{n})$ each.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.
2. Divide S into $\sqrt{n}$ blocks of similar size, for each block, maintain the number of times each value appears with an offset.
   Observe that the number of values that can appear in each block is bounded by the size of the block, because adjacent elements in S differ by exactly 1.
3. Updates and queries can be done in $O(\sqrt{n})$ each.

Time complexity: $O(n\sqrt{n})$.

Problem E: Eerie Subarrays

Solution: Use square root decomposition to handle these range updates and range queries.

1. Initialize the array S to $[1, 2, ..., n]$.
2. Divide S into $\sqrt{n}$ blocks of similar size, for each block, maintain the number of times each value appears with an offset.
   Observe that the number of values that can appear in each block is bounded by the size of the block, because adjacent elements in S differ by exactly 1.
3. Updates and queries can be done in $O(\sqrt{n})$ each.

Time complexity: $O(n\sqrt{n})$.

— *E is very cute, I approve*

Problem C: Chromium Shipping

Input: a graph with n vertices and m edges, along with s employees, t clients, and 2 warehouses located at some of the vertices.

Goal: Find the cheapest way to route each client to a warehouse and then to a unique employee.

Limits:

- $n, m, s, t \leq 2 \cdot 10^5$

11 solves / 49 attempts (22%)

Problem C: Chromium Shipping

Observation 1: The graph does not matter once we have the distance from each employee/client to each warehouse.

Problem C: Chromium Shipping

Observation 1: The graph does not matter once we have the distance from each employee/client to each warehouse.

Observation 2: We may match employees and clients to warehouses separately, as long as the number of employees matched to each warehouse is the same as the number of clients matched to that warehouse.

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \ldots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \ldots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

1. assign each employee to warehouse a
2. greedily move employees from warehouse a to warehouse b (one by one).

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \dots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

1. assign each employee to warehouse a
2. greedily move employees from warehouse a to warehouse b (one by one).
   - move an employee e directly from warehouse a to warehouse b.
     The cost changes by $\min_e\{dist(b, e) - dist(a, e)\}$.

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \ldots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

1. assign each employee to warehouse a
2. greedily move employees from warehouse a to warehouse b (one by one).
   - move an employee e directly from warehouse a to warehouse b.
     The cost changes by $\min_e\{dist(b, e) - dist(a, e)\}$.
   - remove an employee e from warehouse a, and assign an unassigned employee f to warehouse b.
     The cost changes by $\min_f\{dist(b, f)\} - \max_e\{dist(a, e)\}$.

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \ldots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

1. assign each employee to warehouse a
2. greedily move employees from warehouse a to warehouse b (one by one).
   - move an employee e directly from warehouse a to warehouse b.
     The cost changes by $\min_e\{dist(b, e) - dist(a, e)\}$.
   - remove an employee e from warehouse a, and assign an unassigned employee f to warehouse b.
     The cost changes by $\min_f\{dist(b, f)\} - \max_e\{dist(a, e)\}$.

Time complexity: $O(m \log m + (s + t) \log(s + t))$.

Problem C: Chromium Shipping

Solution: For each $k \in \{0, 1, \ldots, t\}$, find the cheapest way to match k employees/clients to warehouse a, and $t - k$ employees/clients to warehouse b.

1. assign each employee to warehouse a
2. greedily move employees from warehouse a to warehouse b (one by one).
   - move an employee e directly from warehouse a to warehouse b.
     The cost changes by $\min_e\{dist(b, e) - dist(a, e)\}$.
   - remove an employee e from warehouse a, and assign an unassigned employee f to warehouse b.
     The cost changes by $\min_f\{dist(b, f)\} - \max_e\{dist(a, e)\}$.

Time complexity: $O(m \log m + (s + t) \log(s + t))$.

— *I literally don't know why it works or how it works or why I coded it*

Problem A: Art Appreciation

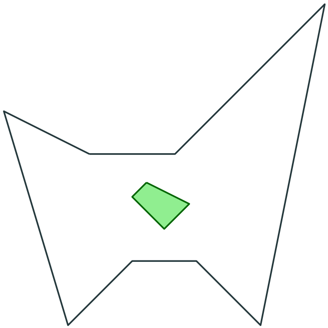
Input: A simple polygon on n vertices.

Goal: Find the area of the set of points from where one could see the entire polygon.

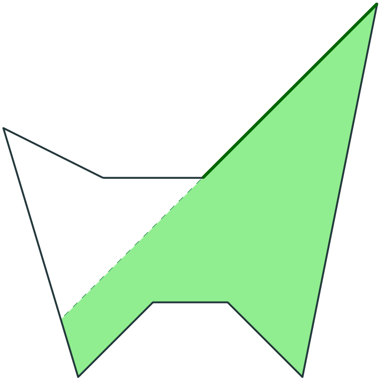
Limits:

- $n \leq 2 \cdot 10^5$

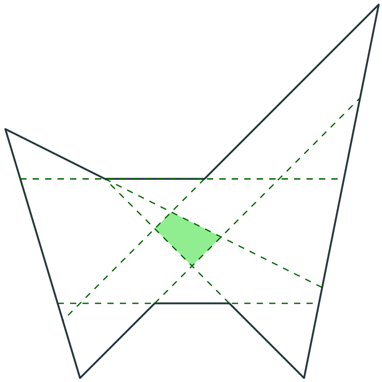
7 solves
/ 57 attempts (12%)



Observation: We must be on the correct side of each edge.



Observation: We must be on the correct side of ALL edges.



Problem A: Art Appreciation

Solution:

1. Each edge of the polygon induces a half-plane
2. Find the area of the intersection of all these half-planes

Problem A: Art Appreciation

Solution:

1. Each edge of the polygon induces a half-plane
2. Find the area of the intersection of all these half-planes

How to intersect half-planes:

- Sort lines by angle, and cut a convex shape (illustrated)
- Sort lines by angle, and build the convex envelope (not illustrated)
- Build upper and lower envelopes, and merge them (standard)

Problem A: Art Appreciation

Solution:

1. Each edge of the polygon induces a half-plane
2. Find the area of the intersection of all these half-planes

How to intersect half-planes:

- Sort lines by angle, and cut a convex shape (illustrated)
- Sort lines by angle, and build the convex envelope (not illustrated)
- Build upper and lower envelopes, and merge them (standard)

How to find the area: shoelace

Problem A: Art Appreciation

Solution:

1. Each edge of the polygon induces a half-plane
2. Find the area of the intersection of all these half-planes

How to intersect half-planes:

- Sort lines by angle, and cut a convex shape (illustrated)
- Sort lines by angle, and build the convex envelope (not illustrated)
- Build upper and lower envelopes, and merge them (standard)

How to find the area: shoelace

Time complexity: $O(n \log n)$.

Problem A: Art Appreciation

Solution:

1. Each edge of the polygon induces a half-plane
2. Find the area of the intersection of all these half-planes

How to intersect half-planes:

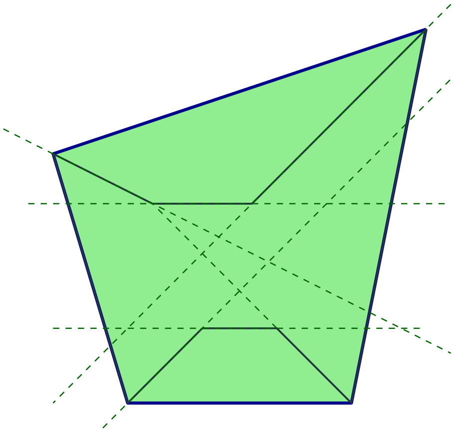
- Sort lines by angle, and cut a convex shape (illustrated)
- Sort lines by angle, and build the convex envelope (not illustrated)
- Build upper and lower envelopes, and merge them (standard)

How to find the area: shoelace

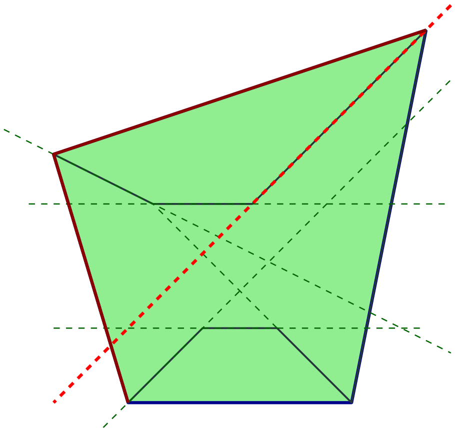
Time complexity: $O(n \log n)$.

— *numerical integration? I should have tried that*

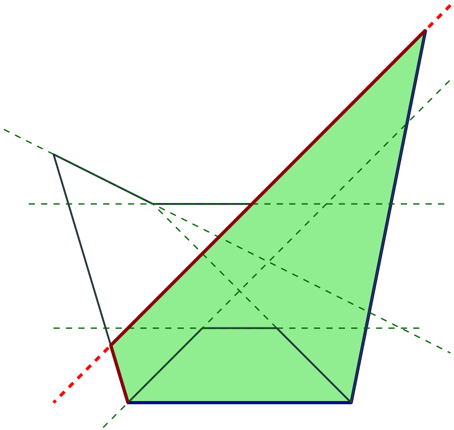
Problem A: Art Appreciation



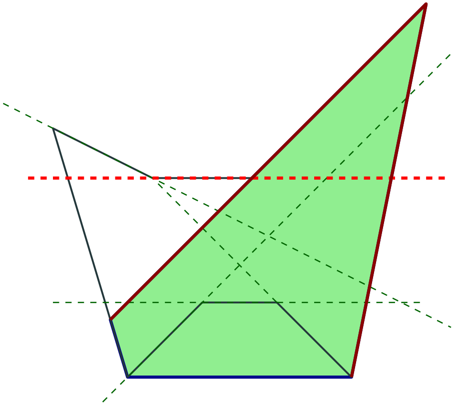
Problem A: Art Appreciation



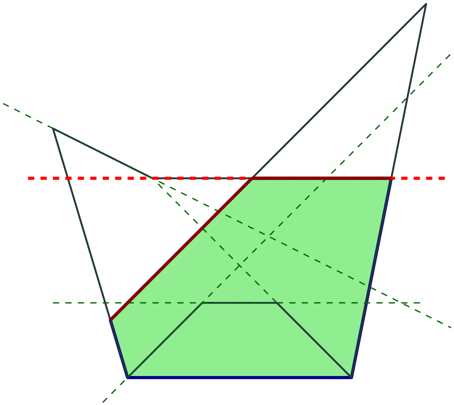
Problem A: Art Appreciation



Problem A: Art Appreciation



Problem A: Art Appreciation



Problem A: Art Appreciation