3-SUM

The problem on Assignment 1

3-SUM PROBLEM

- Classic problem from Gajentaan and Overmars (1995)

THE PROBLEM

- Given 3 arrays A, B, and C
- Each consists of n integers
- Problem: give i, j, k such that A[i] + B[j] = C[k]

Can someone give me a simple algorithm to solve this problem in $O(n^3)$ time?

How about $O(n^2 \log n)$ time?

IS THIS ACTUALLY WORTH SOLVING?

- Yes, surprisingly!
- Important subroutine for:
  - Finding 3 collinear points (important for ruling out corner cases in computational geometry)
  - Problems in graphs (finding 0-sum triangles)
  - Pattern matching (problems involving dictionaries of large strings)

BETTER ALGORITHM

- Can solve it in $O(n^2)$
- Another "walk from both sides" algorithm
- Idea: sort A and B. (can also sort C if you want)
- Fix a k
- Can find in $O(n)$ time if there is an i, j such that A[i] + B[j] = C[k]

- Invariant: if pointing at i' and j', then the correct i and j satisfy i >= i', and j <= j'

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

$68 + 3 = 71 > 53$
So we decrement B's pointer

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

44 + 3 = 47 < 53
So we increment A's pointer

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

WALK FROM BOTH SIDES

Done!

A:

3	5	27	33	46	51	67	89

B:

2	7	8	9	44	55	67	68

Target: C[k] = 53

RUNNING TIME

- How long does all this take?
- Time to sort?
  - O(n log n)
- Time to walk?
  - O(n) per value of k
- How many values of C do we need to iterate over?
  - All n
- Gives $O(n^2)$ total time

TAKING 3SUM FURTHER

- That was a cool algorithm! But it's a bit simple to implement
- We're implementing a version of 3-SUM that uses *blocking*. It has much better efficiency in terms of cache misses.

- (An aside: I believe this blocked version of 3-SUM will not be much faster, if it's faster at all. This assignment is about what you learned: taking a new algorithm, and turning it into efficient code.)

MAGIC HASH FUNCTION

```
uint64_t hash3(uint64_t value){
  return (value * 0x765a3cc864bd9779) >> (64 - SHIFT)
}
```

- Why is this magic?
- If X + Y = Z, then either:
  - `hash3(X) + hash3(Y) = hash3(Z)`
  - `hash3(X) + hash3(Y) = hash3(Z) + 1`

MAGIC HASH FUNCTION: EXPLANATION

This number is not magic (any large odd number will work)

```
uint64_t hash3(uint64_t value){
  return (value * 0x765a3cc864bd9779) >> (64 - SHIFT)
}
```

- You don’t need to know why this works. (Short version: how can lower bits of two numbers affect their sum? The two cases are if there is a carry, or there isn’t a carry)

- You DO need to know: how many values can this hash output?
  - Answer: `1 << SHIFT`

FINAL ALGORITHM

- Create `1 << SHIFT` hash buckets for A, called BucketA
  - For each item x in A, store x in bucket `BucketA[hash3(x)]`
- Create `1 << SHIFT` hash buckets for B, called BucketB
  - For each item x in B, store x in bucket `BucketB[hash3(x)]`
- Create `1 << SHIFT` hash buckets for C, called BucketC
  - For each item x in C, store x in bucket `BucketC[hash3(x)]`

FINAL ALGORITHM

For a = 1 to (1 << SHIFT)

For b = 1 to (1 << SHIFT)

Call the simple 3SUM algorithm with lists: `BucketA[a], BucketB[b], BucketC[(a + b) (modulo 1 << SHIFT)]`

Call the simple 3SUM algorithm with lists: `BucketA[a], BucketB[b], BucketC[(a + b + 1) (modulo 1 << SHIFT)]`

QUICK COMMENTS

- How to store hash buckets?
  - You don't know the size ahead of time
  - But, must be *cache-efficient* within each bucket
- Need to find *original* (unsorted) value
- The running time of this version is still $O(n^2)$

- Any questions?