Optimizing Your Code

Why optimization?

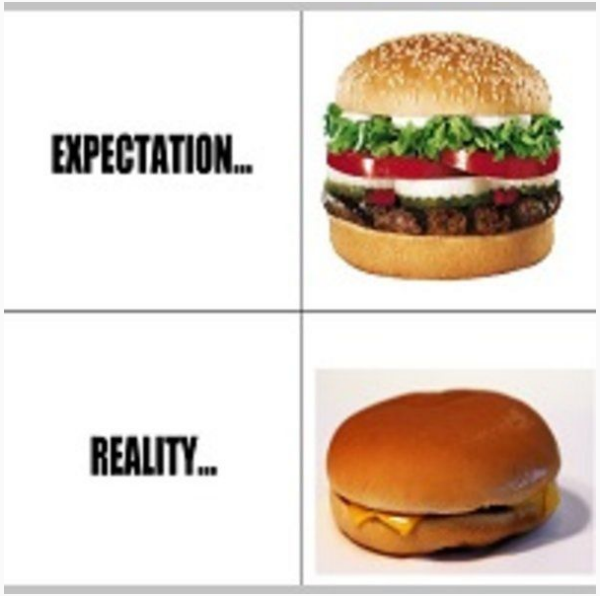
- Obviously we know we want the most efficient code possible
- Breaking out optimization into its own step allows us to take a structured approach
- Remember how important it is to have a framework

Your optimization strategy

Efficiently improve your brute force solution **as much as you can**

Interviewer Expectations

- How well do interviewers actually expect you to optimize the problem?
  - It depends
- You can still succeed without an optimal solution
- Some problems don't really have room for optimization (eg. Islands, Tree Depth)
- Focus on showing your work and backing up your assertions



Optimization step by step

1. Compute the Best Conceivable Runtime
2. Apply the BUD strategy
3. Brainstorm data structures and algorithms
4. Draw on your existing knowledge
5. *Ask for help*

Best Conceivable Runtime

Best Conceivable Runtime (BCR)

- What is the best that we could possibly do?
- Allows us to establish limits on how much we could possibly optimize
- Are there obvious limitations on our speed?
  - Printing data
  - Iterating over the input
  - Comparing items?

byte by byte

Print a linked list in reverse order

Given a singly linked list, write a function that prints out the linked list in reverse order.

eg.

```
printReverse(1 -> 2 -> 3 -> 4)
```

```
4
3
2
1
```

- At minimum we have to iterate over all of the elements in the list

Merge intervals

Given a set of intervals, merge all of the overlapping intervals.

eg.

```
merge([[1, 4], [3, 5], [7, 8]]) =

[[1, 5], [7, 8]]
```

- Bare minimum we have to look at all the data given to us

Sort an Array

- A little tricky
- Do we actually have to compare every element?

BUD Optimization

BUD Optimization

- Bottlenecks
- Unnecessary Work
- Duplicated Work

bytebybyte

Bottlenecks

- Is there some slow one-time work that you're doing?
  - Sorting?
- Are we doing some sort of slow preprocessing step?
- Is there something that could be significantly optimized by doing preprocessing?

Bottlenecks

- Find all duplicates
  - We could sort and find duplicates
  - Better to use a set

Unnecessary Work

- Can we come to the solution by skipping a step or doing that step more efficiently?
- Are we looking at information that is not relevant to finding a solution?

Unnecessary Work

- Merge Intervals
  - We don't need to look at the entire interval
  - We only need to look at endpoints
- Picking Stocks
  - We don't really need to compare every pair
  - We only need to compare local minima with local maxima
  - Any other trades are guaranteed to be worse

```
[[1, 3], [5, 6], [6, 8]]


   [1, 2, 3]
                [5, 6]
                   [6, 7, 8]
[0, 1, 1, 1, 0, 1, 1, 1, 1]
```

Duplicated Work

- Are we doing the same thing more than once?
- Are we iterating over a value multiple times?
- Could we check whether something has already been computed?

Duplicated Work

- Climbing Stairs
  - If we do this recursively, we will call the same subproblem multiple times
- Max Sum Subarray
  - We don't need to compute the sums multiple times
  - We can just keep a running maximum
- Any DP problems

Additional Optimization Strategies

Space/Time Tradeoff

- Can often optimize for space by sacrificing time or vice versa
- Eg. Any problem that takes in only integers can be solved in constant time
- The goal here is to find the best balance
- Can we improve one without sacrificing the other?

Draw on your existing knowledge

- You’re learning lots of different problems as you prepare for your interviews
- Focus on the general patterns rather than the entire problem
- In your interview, brainstorm similar problems to the one you’re currently solving
- It’s good to keep a list of patterns as you study

Consider any additional information

- Did your interviewer give you any additional information that you're not taking advantage of?
- Is the input sorted?
- Is there some limitation on the size of the input?

Brainstorm data structures

- Is there any sort of data structure that obviously works here?
- Have you considered using a hash table or a set?
- Have you considered using a heap or priority queue?
  - These are often useful and overlooked