Divide and Conquer

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Spring 2024

Objectives

- Describe the characteristics of a divide and conquer algorithm
- Apply divide and conquer to sorting
- Apply divide and conquer to binary search

Characteristics

Divide and Conquer has two common forms:

- Combining subproblems: break the problem space into parts, solve the parts, combine the parts.
  - Example: sorting, segment trees
- Pruning search space: evaluate current situation, prune half of search space, search the other half.
  - Example: binary search

Sorting

$$\underbrace{0 \quad 2 \quad 12 \quad 40 \quad 40 \cdots 40 \quad 40 \quad 30 \quad 14 \quad 9}_{\text{An array with lots of 40's}}$$

- For quicksort: you already know that you need to pick a random pivot.
- You also need to partition into 3 spaces: $<, =, >$.
- Really, just use `sort` from the STL.
- Unless you need stable sorting!
  - Use merge sort
  - Create pairs using the original index as the second component, the sort on the pairs.

Binary Search

- Algorithm: divide the search space into two and decide which of the two to explore.
- Classic examples:
  - searching for an element in a sorted array
  - searching for the zero of a function

```
1  double lo = 0
2  double hi = 10000
3  double mid = (hi + lo)/2
4
5  while (fabs(f(mid)) > EPS) {
6    if (f(mid)>0)
7      hi = mid;
8    else
9      lo = mid;
10   mid = (hi+lo)/2;
11 }
```

Ternary Search

- Suppose you want to search for the minimum of a parabola...
  - $a > b \Rightarrow f(a) > f(b)$ on the left side of the min.
  - $a > b \Rightarrow f(a) < f(b)$ on the right side of the min.
- Need three regions, each step exclude one.

```
// Stolen from CP 4
for (int i = 0; i < 50; ++i) { // similar as BSTA
  double delta = (hi-lo)/3.0; // 1/3rd of the range
  double m1 = lo+delta; // 1/3rd away from lo
  double m2 = hi-delta; // 1/3rd away from hi
  (f(m1) > f(m2)) ? lo = m1 : hi = m2; // f is unimodal
}
```

Binary Search the Answer

- Suppose you want to launch a rocket to a distant asteroid (or do some other physics simulation)
  - no closed form solution exists
  - want the minimum amount of fuel / initial velocity / whatever to get there.

```
#define EPS 1e-9  // Code from Competitive Programming 3 t
bool can(double f) {
    // Can you do the task with starting fuel f?
}
int main() {
    double lo = 0.0, hi = 10000.0, mid = 0.0, ans = 0.0;
    while (fabs(hi - lo) > EPS) {  // answer not found yet
        mid = (lo + hi) / 2.0;
        if (can(mid)) {
            ans = mid; hi = mid;  // We can do it, try a low
        } else lo = mid; // couldn't do it, go higher
    }
```