Time Complexity

Neo Wang

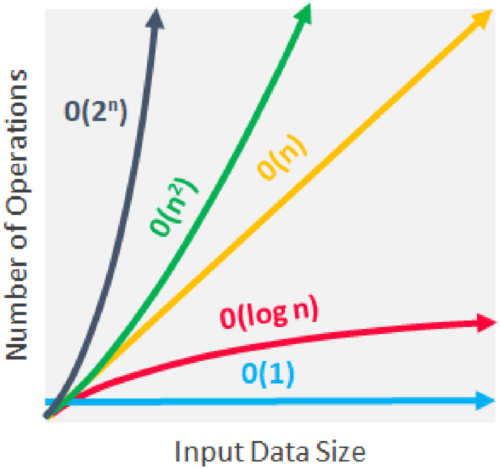
Westlake High School

September 1, 2021

Time Complexity

Time complexity describes the amount of time it takes for a computer to run an algorithm.

Time Complexity - Examples



- Notice how the number of operations of the algorithm grows as the input size grows.

Common Time Complexities

- $\mathcal{O}(1)$ - Constant Time. Example: swap two numbers in an array, arithmetic.
- $\mathcal{O}(N)$ - Linear Time. Example: looping through numbers $[0, \ldots, n]$
- $\mathcal{O}(N^2)$ - Quadratic Time. Example: looping through an $N \times N$ matrix.
- $\mathcal{O}(\log N)$ - Logarithmic Time. Example: binary search.
- $\mathcal{O}(N \log N)$ - Logarithmic Time. Example: merge sort.
- Bonus: $\mathcal{O}(\alpha(N))$

Notes on Time Complexity

- Time complexity ignores lower forms of computation, because we are only trying to describe how the computation scales with input size.
- Example: $\mathcal{O}(2N)$ is the same as $\mathcal{O}(N)$, because we ignore the constant factor of 2.
- Example 2

$$\mathcal{O}(N^2 \log N + N \log N) = \mathcal{O}(N^2 \log N)$$

Code Example

```
for(int i = 0; i < 1 << n; i++) {
  int b = i - 5;
}

for(int i = 0; i < m * m; i++) {
  int c = i + m;
  int b = m;
  swap(c, b);
}
```

- Assume n and m are already defined.

Code Example

```
for(int i = 0; i < 1 << n; i++) {
  int b = i - 5;
}

for(int i = 0; i < m * m; i++) {
  int c = i + m;
  int b = m;
  swap(c, b);
}
```

- Assume n and m are already defined.
- Solution:

Code Example

```
for(int i = 0; i < 1 << n; i++) {
  int b = i - 5;
}

for(int i = 0; i < m * m; i++) {
  int c = i + m;
  int b = m;
  swap(c, b);
}
```

- Assume n and m are already defined.
- Solution:
- Solution: $\mathcal{O}(2^n + m^2)$

Why is this useful?

Problem: Given an array of (x, y) points, find the closest pair of points. There are N points. For reference, looping through each combination of points would take $O(N^2)$ time. In around 2 seconds, the computer can process in the order of around $5 \cdot 10^8$ operations.

Find out if the time complexity of $\mathcal{O}(N^2)$ would pass for the following:

Problem 1: $N \leq 3000$

Problem 2: $N \leq 10^5$

Why is this useful?

Problem 1: $N \leq 3000$
Solution:

Why is this useful?

Problem 1: $N \leq 3000$
Solution:

- Solution: Yes, because the time complexity is $O(N^2)$. We can estimate our calls by plugging the upper bound of $N (N = 3000)$ into the equation. $N^2 = 3000^2 = 9 \cdot 10^6$ which fits in the bounds of $\approx 5 \cdot 10^8$ operations.

```
int closestDistance(vpi points) {
  int n = points.size();
  int result = INT_MAX;
  for(int i = 0; i < n; i++) {
    for(int j = i + 1; j < n; j++) {
      result = min(result, distance(points[i], points[j]));
    }
  }
  return result;
}
```

Why is this useful?

Problem 1: $N \leq 10^5$
Solution:

Why is this useful?

Problem 1: $N \leq 10^5$
Solution:

- Solution: No, because the time complexity is $O(N^2)$. We can estimate our calls by plugging the upper bound of $N (N = 10^5)$ into the equation. $N^2 = (10^5)^2 = 10^10$ which does not fit in the bound of $\approx 5 \cdot 10^8$ operations.

Workaround

So what if we want to solve problem 2, since it times out?

- Devise a better algorithm, that passes under $\mathcal{O}(N^2)$ time complexity.
- Costs: Takes a bit of time to implement.
- Benefits: Runs faster.

Workaround

```
#include "../Primitives/Point.h"

pair<P,P> solve(vP v) {
    pair<ld,pair<P,P>> bes; bes.f = INF;
    set<P> S; int ind = 0;
    sort(all(v));
    FOR(i,sz(v)) {
        if (i && v[i] == v[i-1]) return {v[i],v[i]};
        for (; v[i].f-v[ind].f >= bes.f; ++ind)
            S.erase({v[ind].s,v[ind].f});
        for (auto it = S.ub({v[i].s-bes.f,INF});
            it != end(S) && it->f < v[i].s+bes.f; ++it) {
            P t = {it->s,it->f};
            ckmin(bes,{abs(t-v[i]),{t,v[i]}});
        }
        S.insert({v[i].s,v[i].f});
    }
    return bes.s;
}
```

Advanced Resources

- Stanford CS106B Lecture 11 - Good Introduction
- $\mathcal{O}(\alpha(N))$ proof for Disjoint Set Union w/ Path Compression and Rank
- $\mathcal{O}(\log(N))$ proof for Disjoint Set Union w/ Path Compression or Rank