Time Complexity

Srivaths P

Goal:

- Understand time complexity
- Understand Big-O notation for time complexity.
- Evaluate time complexity of an algorithm.
- Evaluate expected time complexity based on the given constraints of a problem.
- Evaluating space complexity of a program.
- Common verdicts of submissions.

What is an Elementary Operation?

An operation that takes constant time is called elementary operation.

Example:

- Arithmetic operations
- Comparison of primitive types
- Input and output of primitive types

10^8 operations ≈ 1 second

Quiz 1

1. Is the following an elementary operation?

```
int a, b, c, d;
cin >> a >> b >> c >> d;

cout << (a + b * c) / d << endl;
```

2. Is the following an elementary operation?

```
string s, t;
cin >> s >> t;

if (s < t)
    cout << "s is less than t" << endl;
```

What is Time Complexity?

Time complexity is a function to describe the approximate amount of operations an algorithm requires for the given input.

We can calculate approximate execution time of code using time complexity and constraints.

Big-O notation

Big-O of an algorithm is a function to calculate the worst case time complexity of the algorithm.

It is written as O(worst case time complexity)

Big-O is used to calculate the approximate *upper bound* of the algorithm. It expresses how the run time of the algorithm grows relative to the input.

More convenient and useful than other notations.

Rules for Big-O notation

- Should not have constants.
- Should not have constant factors.
- Only include the *fastest growing* function *for each variable*.
- Can never be 0. Has to be atleast O(1)

Example function: $2(N^2) + 4N + 4(M^3 + 5) + 10$

Quiz 2

1. $(N+M) / K$
2. $N(N+1)/2$
3. $N^2 + M(N^2) + M^2(N) + NM$
4. $N^3/64 + 20N + (32NM)^2$

Calculate Time Complexity of an Algorithm

Time complexity usually depends on:

- Loops
- Recursion

Time complexity of recursive algorithms will not be covered.

Note: Usage of STL counts for time complexity

Calculate Time Complexity of an Algorithm

If there are nested loops, multiply the expected number of iterations of the loops

Example:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < 4; k++) {
            // Elementary operations
        }
    }
}
```

Quiz 3

Find the time complexity of the following code snippets in Big-O notation:

1.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n/2; j++) {

    }
}
```

2.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j*j < n; j++) {

    }
}
```

Quiz 3

3.

```
for (int i = 1; i < n; i *= 2) {


}
```

4.

```
for (int i = 12; i <= n-123; i += 5) {
    for (int j = 6; j <= m*2; j += 321) {
        for (int k = 4023; k > 23; k -= 16) {


        }
    }
}
```

Time Complexity based on Constraints

Feasible Big-O Function	Maximum N	Example Algorithms
$O(N!)$	10	All permutations of a list
$O(N^3)$	400	Multiplication of two matrices
$O(N^2)$	5000	Square grid, bubble sort, insertion sort
$O(N\sqrt{N})$	10^5	Usually related to factoring
$O(NlogN)$	10^6	Merge sort, binary search for N times
$O(N)$	10^7	Linear search, reversing an array, string comparison
$O(\sqrt{N})$	10^{12}	Factors of a number
$O(logN)$, O(1)	10^{18}	Binary search, Constant time formulas

Space Complexity

Space complexity is similar to time complexity, except it measures the amount of memory.

Any datatype that has constant memory takes O(1) space.

Example: int, char, long long int, double, etc.

Space Complexity

Most problems have a memory limit of 256MB or ~2e8 bytes.

Datatype	No. of bytes
char	1
int	4
float	4
long long int	8
double	8
long double	16

Verdict of a solution

- AC: Accepted
- WA: Wrong Answer
- TLE: Time Limit Exceeded
- MLE: Memory Limit Exceeded
- RE: Runtime Error
  - NZEC: Non Zero Exit Code
  - SIGSEGV: Usually due to out of bounds
  - SIGFPE: Usually division or modulo by 0
  - SIGABRT: Due to assert statements

Points to note:

- Identify the variables that contribute to time complexity.
- Just because constraints allow slower solutions, doesn't mean there's not a fast solution.
  For example, if N <= 1000, then both $O(N^2)$ and O(N) can pass.
- Testcases matter, unless there's a limit explicitly imposed in the constraints.
- The constants and constant factors removed when calculating Big-O still matter.

Problems to test understanding

- https://codeforces.com/contest/1647/problem/A
- https://codeforces.com/problemset/problem/1538/C
- https://www.codechef.com/MARCH221D/problems/DISCUS
- https://www.codechef.com/MARCH221D/problems/WORDLE
- https://www.codechef.com/MARCH221D/problems/CHFDBT
- https://codeforces.com/contest/1651/problem/B
- https://codeforces.com/contest/1651/problem/A
- https://codeforces.com/problemset/problem/919/B

For more practice, try to figure out the time complexity for any random problem.

Further Reading:

- https://towardsdatascience.com/essential-programming-time-complexity-a95bb2608cac

- https://www.youtube.com/watch?v=9TlHvipP5yA
  https://www.youtube.com/watch?v=9SgLBjXqwd4
  https://www.youtube.com/watch?v=I0DTkS1LJ2k

- https://adrianmejia.com/most-popular-algorithms-time-complexity-every-programmer-should-know-free-online-tutorial-course/ (advanced)