Data Structures and Algorithms-I

Algorithm Complexity

About the course

- My name: Dr. Prakash Raghavendra
- Areas of Interest:
 - Compilers, HPC, Optimization for Machine Learning
- In this course:
 - Complexity, Stacks, Lists
 - Hash Tables
 - Binary Search Trees
 - Graphs, Balanced Search Trees
 - Searching, Sorting
 - Hard Problems

Dr. Prakash Raghavendra – IT202- Data Structures and Algorithms-I

Evaluation

• Quiz: 20%

• Mid Sem: 30%

• End Sem: 50%

What is an algorithm?

- An algorithm is "a finite set of precise/well defined steps/instructions for performing a computation or for solving a problem"
 - A program is an implementation of algorithm
 - Directions to somebody's house is an algorithm
 - A recipe for cooking a cake is an algorithm
 - The steps to compute the cosine of 90° is an algorithm
 - Next, you can take left/right turn not an algorithm step

Some algorithms are harder than others

- Some algorithms are easy
 - Finding the largest (or smallest) value in a list
 - Finding a specific value in a list
- Some algorithms are a bit harder
 - Sorting a list
- Some algorithms are very hard
 - Finding the shortest path between Miami and Seattle
- Some algorithms are essentially impossible
 - Factoring large composite numbers
- In this course, we'll see how to rate how "hard" algorithms are

Algorithm : Maximum element

 Given a list, how do we find the maximum element in the list?

- To express the algorithm, we'll use pseudocode
 - Pseudocode is kind of a programming language, but not really

6

Algorithm : Maximum element

Algorithm for finding the maximum element in a list:

```
procedure max (a_1, a_2, ..., a_n): integers)

max := a_1

for i := 2 to n

if (a_i > max) then max := a_i

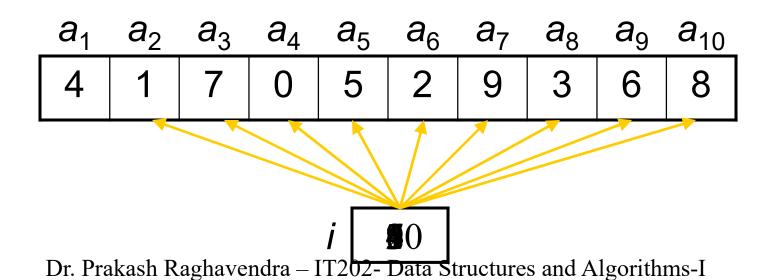
{max is the largest element}
```

Algorithm 1: Maximum element

```
procedure max (a_1, a_2, ..., a_n): integers)

max_i = a_{a_1}

for i_i = 2 to n
if \max < a_i then \max := a_i
if \max < a_i then \max := a_i
if \max < a_i then \max := a_i
\max < a_i then \max := a_i
```



Maximum element running time

- How long does this take? Rather, how many operations does it take to find the max?
- If the list has n elements, worst case scenario is that it takes n "steps"
 - Here, a step is considered a single step through the list

Properties of algorithms

- Algorithms generally share a set of properties:
 - Input: what the algorithm takes in as input
 - Output: what the algorithm produces as output
 - Definiteness: the steps are defined precisely
 - Correctness: should produce the correct output
 - Finiteness: the steps required should be finite
 - Effectiveness: each step must be able to be performed in a finite amount of time
 - Generality: the algorithm should be applicable to all problems of a similar form

• Informally, Time to solve a problem of size, n, T(n) is $O(\log n)$

$$\Box T(n) = c \log_2 n$$

- Formally:
 - O(g(n)) is the set of functions, f, such that f(n) < c g(n)

for some constant, c > 0, and n > N

ie for sufficientlylarge n

• Alternatively, we may write $\lim_{n\to\infty}\frac{f(n)}{g(n)}\leq c$

g is an upper bound for f

Dr. Prakash Raghavendra – IT202- Data Structures and Algorithms-I

- O(g)
 - the set of functions that grow no faster than g.
- g(n) describes the worst case behaviour of an algorithm that is O(g)

- O(g)
 - the set of functions that grow no faster than g.
- g(n) describes the worst case behaviour of an algorithm that is O(g)
- Two additional notations
- $\Omega(g)$
 - the set of functions, f, such that

for some constant, c, and n > N

g is a lower bound for f

- O(g)
 - the set of functions that grow no faster than g.
- g(n) describes the worst case behaviour of an algorithm that is O(g)
- Two additional notations
- $\Omega(g)$
 - the set of functions, f, such that

$$f(n) > c g(n)$$

for some constant, c, and n > N

g is a lower bound for f

• $\Theta(g) = O(g) \cap \Omega(g)$ Set of functions growing

Set of functions growing at the same rate as $oldsymbol{g}$

- Constant factors may be ignored
 - $\forall k > 0, kf is O(f)$

- Constant factors may be ignored
 - $\forall k > 0, kf \text{ is } O(f)$
- Higher powers grow faster
 - n^r is $O(n^s)$ if $0 \le r \le s$

- Constant factors may be ignored
 - $\forall k > 0, kf \text{ is } O(f)$
- Higher powers grow faster
 - n^r is $O(n^s)$ if $0 \le r \le s$
- □ Fastest growing term dominates a sum
 - If f is O(g), then f + g is O(g) $eg \quad an^4 + bn^3 \quad \text{is} \quad O(n^4)$

- Constant factors may be ignored
 - $\forall k > 0, kf \text{ is } O(f)$
- Higher powers grow faster
 - n^r is $O(n^s)$ if $0 \le r \le s$
- □ Fastest growing term dominates a sum
 - If f is O(g), then f + g is O(g) $eg \quad an^4 + bn^3 \quad \text{is} \quad O(n^4)$
- Polynomial's growth rate is determined by leading term
 - If f is a polynomial of degree d, then f is $O(n^d)$

Dr. Prakash Raghavendra – IT202- Data Structures and Algorithms-I

- f is O(g) is transitive
 - If f is O(g) and g is O(h) then f is O(h)

- f is O(g) is transitive
 - If f is O(g) and g is O(h) then f is O(h)
- Product of upper bounds is upper bound for the product
 - If f is O(g) and h is O(r) then fh is O(gr)

- f is O(g) is transitive
 - If f is O(g) and g is O(h) then f is O(h)
- Product of upper bounds is upper bound for the product
 - If f is O(g) and h is O(r) then fh is O(gr)
- Exponential functions grow faster than powers
 - n^k is $O(b^n) \forall b > 1$ and $k \ge 0$ eg n^{20} is $O(1.05^n)$

- f is O(g) is transitive
 - If f is O(g) and g is O(h) then f is O(h)
- Product of upper bounds is upper bound for the product
 - If f is O(g) and h is O(r) then fh is O(gr)
- Exponential functions grow faster than powers
 - n^k is $O(b^n) \forall b > 1$ and $k \ge 0$ eg n^{20} is $O(1.05^n)$
- Logarithms grow more slowly than powers
 - $\log_b n$ is $O(n^k) \forall b > 1$ and k > 0eg $\log_2 n$ is $O(n^{0.5})$

- f is O(g) is transitive
 - If f is O(g) and g is O(h) then f is O(h)
- Product of upper bounds is upper bound for the product
 - If f is O(g) and h is O(r) then fh is O(gr)
- Exponential functions grow faster than powers
 - n^k is $O(b^n) \forall b > 1$ and $k \ge 0$ eg n^{20} is $O(1.05^n)$
- Logarithms grow more slowly than powers
 - $\log_b n$ is $O(n^k) \forall b > 1$ and k > 0eg $\log_2 n$ is $O(n^{0.5})$ [mportant]

- All logarithms grow at the same rate
 - $\log_b n$ is $O(\log_d n) \forall b, d > 1$

- All logarithms grow at the same rate
 - $\log_b n$ is $O(\log_d n) \forall b, d > 1$
- Sum of first $n r^{th}$ powers grows as the $(r+1)^{th}$ power

•
$$\sum_{k=1}^{n} k^{r}$$
 is $\Theta(n^{r+1})$

$$eg \quad \sum_{k=1}^{n} i = \frac{n(n+1)}{2} \quad \text{is } \Theta(n^2)$$

Polynomial and Intractable Algorithms

- Polynomial Time complexity
 - An algorithm is said to be polynomial if it is $O(n^d)$ for some integer d
 - Polynomial algorithms are said to be efficient
 - They solve problems in reasonable times!
- Intractable algorithms
 - Algorithms for which there is no known polynomial time algorithm
 - We will come back to this important class later in the course

Analysing an Algorithm

Simple statement sequence

```
s_1; s_2; .... ; s_k
```

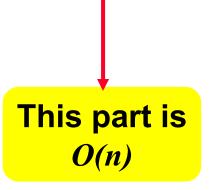
- O(1) as long as k is constant
- Simple loops

```
for(i=0;i<n;i++) { s; } where s is O(1)
```

- Time complexity is n O(1) or O(n)
- Nested loops

```
for(i=0;i<n;i++)
for(j=0;j<n;j++) { s; }</pre>
```

• Complexity is n O(n) or $O(n^2)$



Dr. Prakash Raghavendra – IT202- Data Structures and Algorithms-I

Analysing an Algorithm

Loop index doesn't vary linearly

```
h = 1;
while ( h <= n ) {
    s;
    h = 2 * h;
}</pre>
```

- h takes values 1, 2, 4, ... until it exceeds n
- There are $1 + \log_2 n$ iterations
- Complexity $O(\log n)$

Analysing an Algorithm

Loop index depends on outer loop index

```
for (j=0;j<n;j++)
for (k=0;k<j;k++) {
    s;
}</pre>
```

- Inner loop executed
 - 1, 2, 3,, n times

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

 \therefore Complexity $O(n^2)$

Distinguish this case - where the iteration count increases (decreases) by a constant $\zeta O(n^k)$ from the previous one - where it changes by a factor $\zeta O(\log n)$

Dr. Prakash Raghavendra – IT202- Data Structures and Algorithms-I