(CS2009)-DESIGN AND ANALYSIS OF ALGORITHMS

Presented By: Sandesh Kumar
Email: sandesh.kumar@nu.edu.pk

LECTURE 1 & 2 – INTRODUCTION & COURSE OVERVIEW

GRADING POLICY (CS 2009)

Assessment Type	**Weight**
Assignment	10
Mid Terms (Week 6 & Week 11)	30 (12.5 and 17.5)
Project	10
Final	50

TEXT AND REFERENCE BOOKS

- Required Textbook
  - Thomas H. Cormen “Introduction to Algorithms” 4th Edition
- Reference Books
  - Anany Levitin “Introduction to the Design and Analysis of Algorithms” 3rd Edition
  - John Kleinberg and Eva Tardos “Algorithm Design”
  - Sanjoy Dasgupta et al. “Algorithms”
  - Steven S. Skiena “The Algorithm Design Manual”

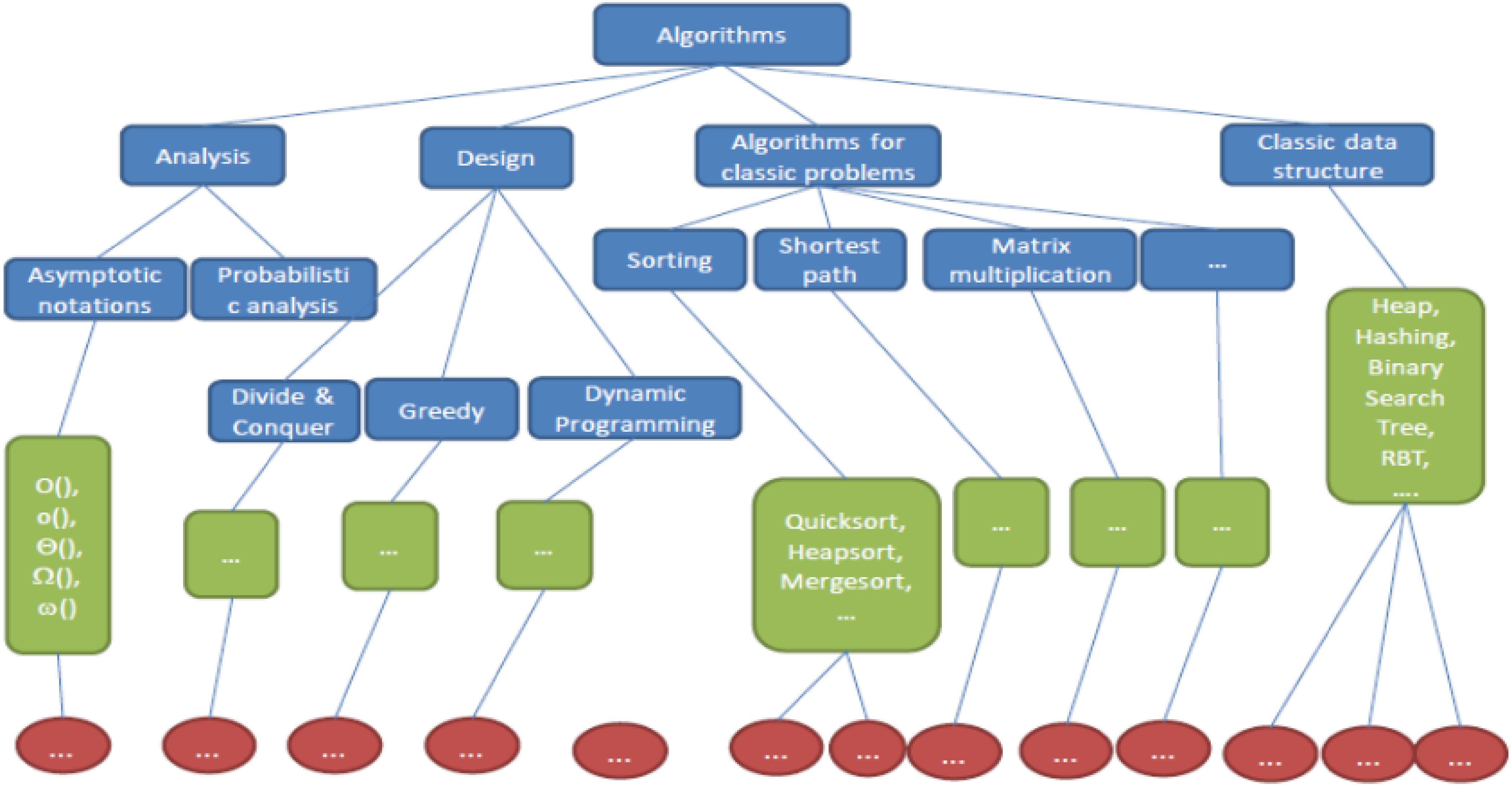
CONTENTS & TENTATIVE SCHEDULE

WEEK	TOPICS
Week 1 & 2	Basics of Algorithms, Mathematical Foundation, Growth of Function, Asymptotic Notations. Data Structure Review (Stack, Queue, Linked List, Hash Table, Binary Tree).
Week 3 & 4	Divide and Conquer, Substitution Method, Recurrence – Tree Method, Master’s Method
Week 5	Sorting (Merge, Insertion, Quick, Heap, Counting, Radix, Bucket)
Week 6	Mid Term 1 Exam
Week 7	Dynamic Programming
Week 8	Dynamic Programming and Greedy Algorithms

CONTENTS & TENTATIVE SCHEDULE

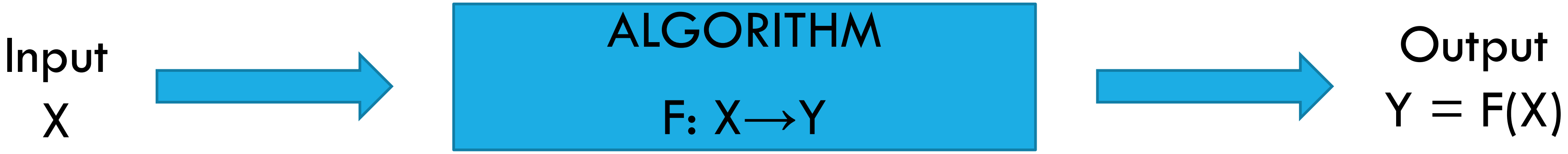
WEEK	TOPICS
Week 9, 10 & 12	Graph Theory (Graph Categorization, Graph Terminology, Representation of Graphs, BFS & DFS, Strongly Connected Components, Greedy Algorithms: Kruskal's Algorithm, Prim's Algorithm, Bellman-Ford Algorithm, Djikstra's Algorithm)
Week 11	Mid Term 2 Exam
Week 13 & 14	Geometric Algorithms (Introduction, Graham Scan, Close Points). String Matching
Week 15 & 16	NP Complete Problems and Solutions using Approximation Algorithm, Amortized Algorithms
Week 17	Review and Project Presentations

KNOWLEDGE TREE



WHAT IS AN ALGORITHM?

- An algorithm is any well-defined computational procedure that takes some value as input and produces some value as output. (Thomas H. Cormen)



Input

X

ALGORITHM

F: X→Y

Output

Y = F(X)

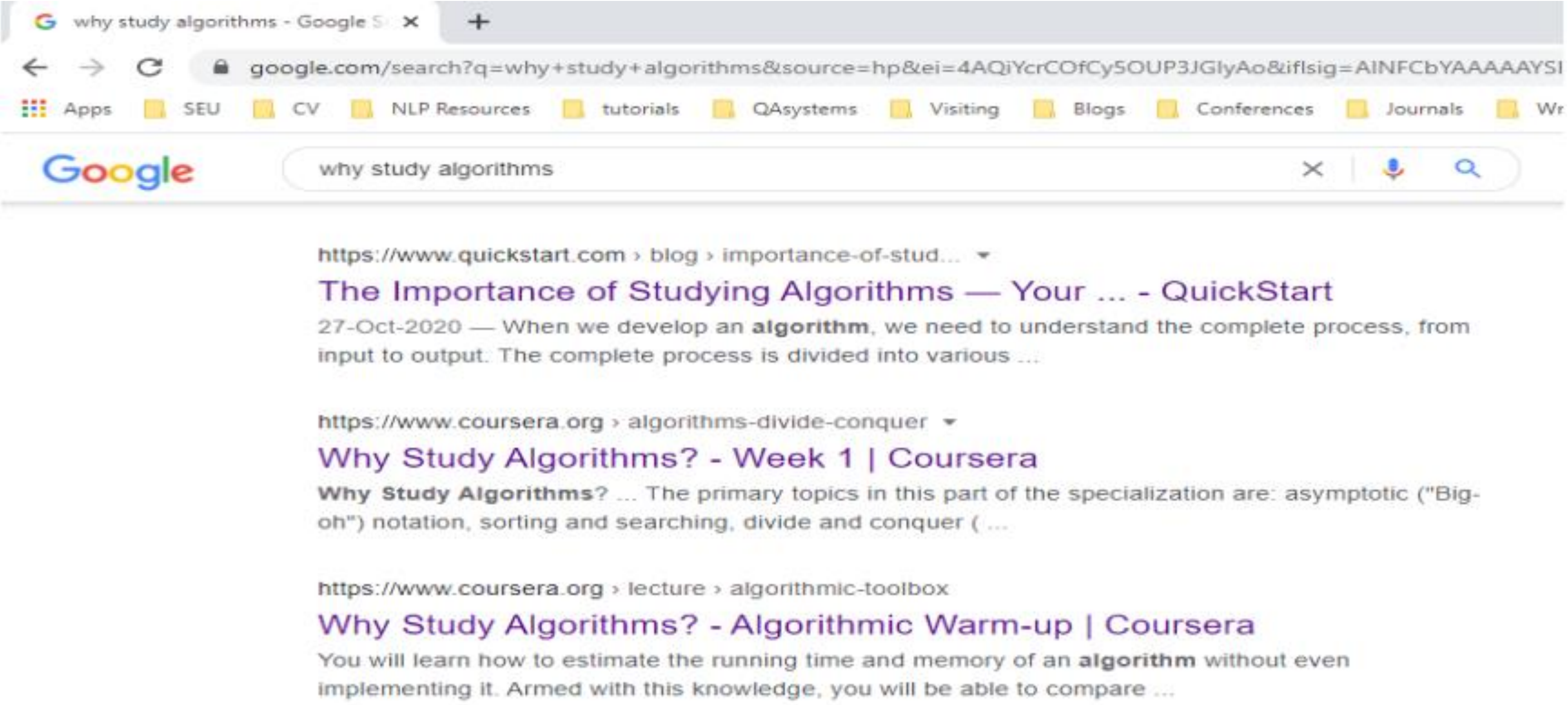
- An algorithm is a sequence of computational steps for solving a problem.
- Examples
  - Multiply Two Numbers
  - A Cooking Recipe
  - Finding a library book in the library
  - Directions for driving from A to B
  - Internet and Communication Links (Graph)

WHAT IS AN ALGORITHM?

- Algorithm: Cook a cup of instant noodles
  1. Pull back lid to the dotted line.
  2. Fill the cup to the inside line with boiling water from a kettle or from the microwave
  3. Close lid and let stand for 3 minutes.
  4. Stir well and add a pinch of salt and pepper to taste.

WHY STUDY ALGORITHMS?

WEB SEARCH


why study algorithms - Google S
google.com/search?q=why+study+algorithms&source=hp&ei=4AQiYcrCOfCy5OUP3JGlyAo&iflsig=AINFCbYAAAAAYSI
Apps
SEU
CV
NLP Resources
tutorials
QAsystems
Visiting
Blogs
Conferences
Journals
Wr
Google

why study algorithms

https://www.quickstart.com › blog › importance-of-stud...

The Importance of Studying Algorithms — Your ... - QuickStart

27-Oct-2020 — When we develop an **algorithm**, we need to understand the complete process, from input to output. The complete process is divided into various ...

https://www.coursera.org › algorithms-divide-conquer

Why Study Algorithms? - Week 1 | Coursera

Why Study Algorithms? ... The primary topics in this part of the specialization are: asymptotic ("Big-oh") notation, sorting and searching, divide and conquer (...

https://www.coursera.org › lecture › algorithmic-toolbox

Why Study Algorithms? - Algorithmic Warm-up | Coursera

You will learn how to estimate the running time and memory of an **algorithm** without even implementing it. Armed with this knowledge, you will be able to compare ...

WHY STUDY ALGORITHMS?

- Personalized Recommendations

- More than 70% of what people watch on YouTube is determined by its "Recommendation Algorithm".

- News Feed, Friend Suggestions

WHY STUDY ALGORITHMS?

- To become proficient programmer.
- To solve problems that could not be solved.
- For fun and profit.

SOLVING PROBLEMS

- When we faced with a problem:
  1. First clearly **define** the problem
  2. Think of possible solutions
  3. **Select** the ones that seems **the best** under the **prevailing** circumstances
  4. Apply that solution
  5. If the solution works as desired, fine. Else **go back to step 2**
- It's **quite common** to first solve a problem for **particular case**
- Then for another and possibly another
- Watch for **patterns and trends that emerge**
- Use the knowledge from these patterns and trends in coming up with a general solution.

APPLICATIONS

- **Internet:** Web search, Packet routing, Distributed file sharing
- **Biology:** Human genome project, protein folding
- **Data Mining:** Text classification, Text Clustering, Page rank
- **Security:** E-commerce, Cell phones, Voting Machine
- **Web Programming:** Sorting algorithms, Searching algorithms
- **Graphics:** Video games, Virtual reality
- **Social Networks:** Recommendations, News feed
- **Machine Learning AI:** Linear Regression Algorithm, Neural Networks such as RNN, CNN
- **Robotics:** Planning Algorithms

WHAT WE ARE INTERESTED IN ALGORITHMS?

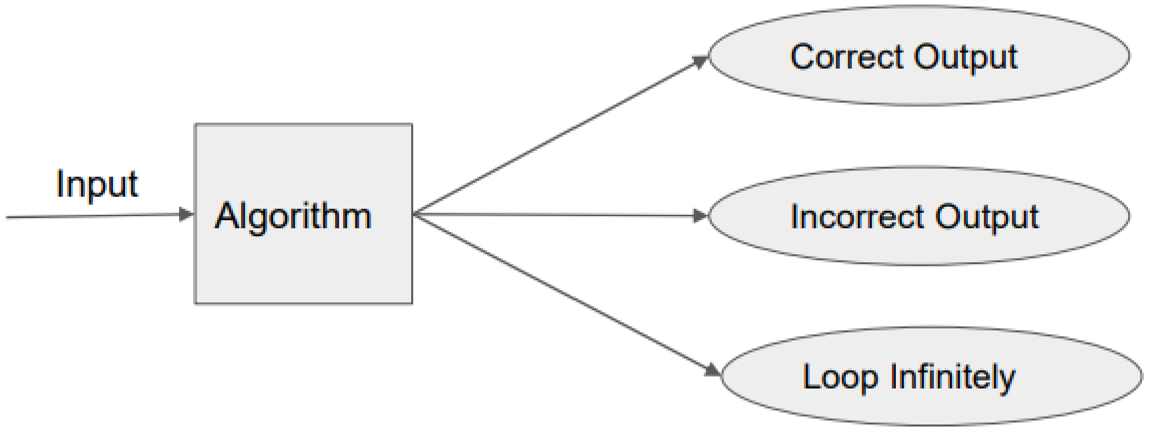
- Correctness
  - Does it work correctly?
- Performance/Efficiency
  - How much time will it take? (Time Complexity)
  - How much space will it take? (Space Complexity)
- Can we do it better?

WHAT'S MORE IMPORTANT THAN PERFORMANCE?

- Correctness
- Robustness
- User-friendliness
- Simplicity
- Extensibility
- Reliability

CORRECT ALGORITHM

- An algorithm is said to be correct if, for every input instance, it halts with the correct output.



WHICH RUNNING TIME IS BETTER?

- Computer A **(FASTER):** Run algorithm of $2n^2$ complexity. Run 10 billions instruction per second.
- Computer B **(SLOWER):** Run algorithm of $50\,nlogn$ complexity. Run 10 millions instruction per second.
- Input length **n = 10** millions
- $2 \ * \frac{(10^7)^2}{10^{10}}$ = 20,000 seconds (> 5.5 hours)
- $50 \ * \frac{(10^7)log10^7}{10^7}$ = 1163 seconds (< 20 minutes)

HOW TO WRITE AN ALGORITHM?

```
Algorithm swap(a,b)
Begin
    temp <- a;
    a <- b;
    b <- temp;
End
```

HOW TO ANALYSE AN ALGORITHM?

1. Time
2. Space
3. Network Data Consumption
4. Power
5. CPU Registers

TIME COMPLEXITY ANALYSIS – ONE LOOP

- **PROBLEM:** Does array A contain the integer t? Given A (array of length n) and t (an integer)

```
for(i=0; i<n; i++)
        if A[i] == t
                          return true
    return false
```

- **QUESTION**: What is the running time?

TIME COMPLEXITY ANALYSIS – ONE LOOP

- **The running time is:**
- 1 assignment (i = 0)
- n+1 comparisons (i<n)
- n increments (i++)
- n array offset calculations (a[i])
- n comparisons (a[i] == t)
- a+b(n+1)+cn+dn+en, where a, b, c, d and e are constants depending upon machine
- Easier just to say O(n) (constant-time) operations

```
for(i=0; i<n; i++)
        if A[i] == t
                          return true
    return false
```

TIME COMPLEXITY ANALYSIS – ONE LOOP

- **PROBLEM:** Does array A contain the integer t? Given A (array of length n) and t (an integer)

```
for(i=0; i<5; i++)
        if A[i] == t
                        return true
    return false
```

- **QUESTION**: What is the running time? O(k) where k = 5

TIME COMPLEXITY ANALYSIS – TWO NESTED LOOPS

- **PROBLEM:** Do arrays A, B have a number in common? Given arrays A, B of length n

```
        for(int i = 0; i < n; i++) {
                for(int J = 0; J < n; J++){
                        if(A[i] == B[J]);
                                return true
                        }
            return false
        }
```

- **QUESTION:** What is the running time? O(n^2)

TIME COMPLEXITY ANALYSIS

```
1). for(i=1; i<n; i=i+2)
    {
            statement; -> n/2
    }
            f(n) = O(n)
```

```
2). for(i=0; i<n; i++)
    {
            for(j=0; j<i; j++)
            {
                    statement;
            }
    }
    f(n) = O($n^2$)
```

TIME COMPLEXITY ANALYSIS

```
3). P=0;
    for(i=1; p<=n; i++)
    {
        p = p+i;
    }
```

Assume p>n -> p = k(k+1)/2

k(k+1)/2>n -> k^2>n -> k>$\sqrt{n}$

```
4).  for(i=1; i<n; i=i*2)
     {
              statement;
     }
```

Assume i>=n -> i=2^k -> 2^k >= n

2^k = n -> k = $log2n$

TIME COMPLEXITY ANALYSIS

```
5). for(i=1; i<n; i=i*2)
    {
        statement;
    }
```

Assume i>=n -> i=2^k

2^k >= n -> 2^k = n

k = $\log_2 n$

```
6). for(i=n; i>=1; i=i/2)
    {
        statement;
    }
```

Assume i<1 -> i =n/2^k -> n/2^k < 1

n/2^k = 1 -> n = 2^k -> k = $\log_2 n$

TIME COMPLEXITY ANALYSIS

```
7). for(i=1; i*i<n; i++)
    {
        statement;
    }
```

Assume i*i >= n

$i^2 = n$ -> $i = \sqrt{n}$

```
8). for(i=n; i>=1; i=i/2)
    {
        statement;
    }
```

Assume i<1 -> i =n/2^k -> n/2^k < 1

n/2^k = 1 -> n = 2^k -> k = $\log_2 n$

TIME COMPLEXITY ANALYSIS

9). P=0

```
  for(i=i; i<n; i=i*2)
      P++;
  }
  for(j=1; j<p; j=j*2)
  {
      statement;
   }
```

O(log logn)

10). for(i=0; i<n; i++)

```
    {
      for(j=1; j<n; j = j*2)
      {
              statement;
      }
    }
```

O(nlogn)

ANALYSIS OF IF AND WHILE

```
1). i=0;
    while(i<n)
    {
      statement;
      i++;
     }
```

f(n) = 3n+2 O(n)

```
2). a=1;
    while(a<b)
    {
         statement;
         a = a*2;
    }
```

Assume a>=b -> a = 2^k -> 2^k>=b

2^k = b -> k = $\log_2 b$ -> O(logn)

ANALYSIS OF IF AND WHILE

3). i=1;

```
k=1;
while(k<n)
{
 statement;
 k = k+l;
 i++;
 }
```

Assume k>=n -> $m^2 + m/2$ = k

m^2 >=n -> m^2 = n -> m = $\sqrt{n}$

4). while(m!=n)

```
{
    if(m>n)
    m = m-n;
else
    n = n-m;
}
```

f(n) = n/2 -> O(n)