Data Structure & Algorithms

Sunbeam Infotech

Nilesh Ghule

Course Introduction

- Data Structure and Algorithms
  - Data Structures: Linked list, Stack, Queue, Binary search tree, Heap.
  - Algorithms: Sorting, Searching, Stack/Queue/Linked list applications.
- Course Goals
  - Implement each DS & Algorithms from scratch.
  - Understand complexity of algorithms.
- Course Schedules
  - 10th Apr 2020 to 23rd May 2020*
  - Sat-Sun: Lecture – 8:30 AM to 12:30 PM
  - Sat-Sun: Lab – 2:00 PM to 3:30 PM
- Resource sharing
  - https://gitlab.com/nilesh-g/dsa-03
  - Recorded videos will be uploaded on http://students.sunbeamapps.org → 28 days
- Course Format
  - Participants are encouraged to code alongside (copy code from code-sharing utility in student portal).
  - Post your queries in chat box (on logical end of each topic).
  - Practice assignments will be shared. They are optional. If any doubts, share on WA group (possibly with screenshot). Faculty members or peers can help.
- Programming language
  - DS & Algorithms are language independent.
  - Classroom coding will be in Java (use IDE of your choice).
  - Will share C++/Python codes at the end of session.
  - Language pre-requisites ?

Course Pre-requisites

Java

- Language Funda
- Methods
- Class & Object
- static members
- Arrays
- Collections
- Inner classes

Python

- Language Funda
- Functions
- Class & Object
- Collections

C++

- Language Funda
- Functions
- Class & Object
- Friend class
- Arrays
- Pointers

C

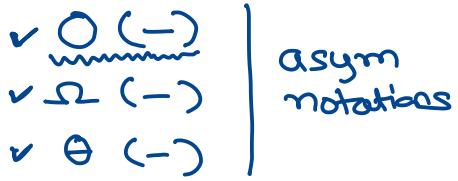
- Language Funda
- Functions
- Structures
- Arrays
- Pointers

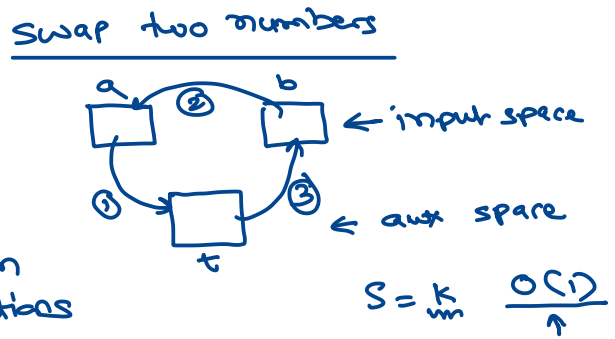
Data Structure

- Data Structure
  - Organizing data in memory
  - Processing the data
- Common data structures
  - Array
  - Linked List
  - Stack
  - Queue
  - Hash Table
- Advanced data structures
  - Tree
  - Heap
  - Graph

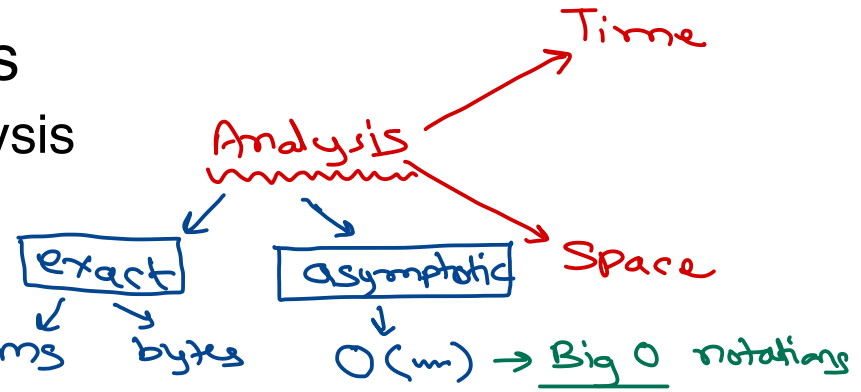
Data Structure

- Data Structure
  - Organizing data in memory
  - Processing the data
- Common data structures
  - Array
  - Linked List
  - Stack
  - Queue
- Advanced data structures
  - Tree
  - Heap
  - Graph

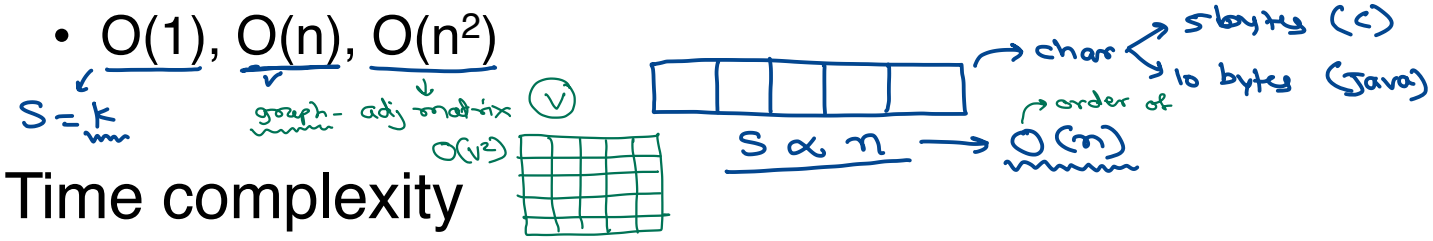




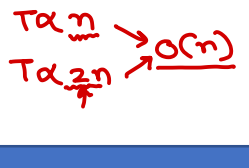
- Asymptotic analysis
  - It is not exact analysis
  - Big' O notation



- Space complexity
  - Unit space to store the data (Input space) and additional space to process the data (Auxiliary space).
  - O(1), O(n), O(n²)



- Time complexity
  - Unit time required to complete any algorithm.
  - Approximate measure of time required to complete any algorithm. T α n, T α n², T α log n, T = k
  - Depends on loops in the algorithm.
  - O(n³), O(n²), O(n log n), O(n), O(log n), O(1)



worst best

Time complexity

- Write a program to calculate factorial of given number.

```
int fact(int n) {
    int r=1;
    for(i=1; i<=n; i++)
        r = r * i;
    return r;
}
```

5! = 5 iters
50! = 50 "
500! = 500 "

$T \propto n \rightarrow O(n)$

input space → O(1) → 1 var
aux space → O(1) → 2 vars

- Print 2-D matrix of n x n.

$T \propto n^2 \rightarrow O(n^2)$

```
for(i=0; i<n; i++) {
    for(j=0; j<n; j++)
        print(arr[i][j]);
}
```

n*n
n iters

R*C iters
$T \propto RC$
O(RC)

input space = $O(n^2)$
aux space = 2 vars = O(1)

- Print given number into binary.

```
print_bin(int n) {
    while(n > 0) {
        print(n%2);
        n = n/2;
    }
}
```

n=9 → iters=4

1001

1024 → 512 → 256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1 : 10 iters

$2^i = n$ (i: iterations, n: dec num)

$i \log 2 = \log n$

$i = \frac{\log n}{\log 2}$ → O(log n)

$T \propto \frac{\log n}{\log 2} \rightarrow T \propto \log(n)$

input space = O(1) – single var
aux space = O(1) – no var.

- Print table of given number.

```
print_table(int n) {
    for(i=1; i<=10; i++)
        print(n * i);
}
```

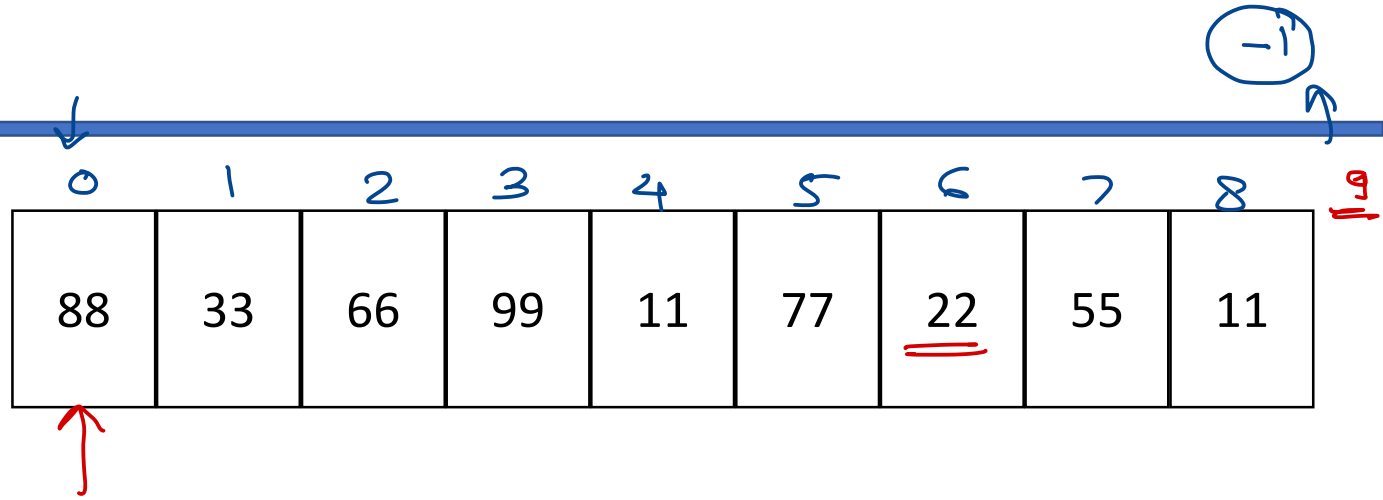
T = k → O(1)

input space = O(1) 1 var
aux space = O(1) 1 var.

Linear Search

- Find a number in a list of given numbers (random order).

88 22 11 90




```
int linearSearch(int [] arr, int key){
    for(int i=0; i<arr.length; i++){
        if(key==arr[i])
            return i; → return index of ele found.
    }
    return -1; → ele not found
}
```

- Time complexity
  - Worst case: Max num of itrs
    → find last ele or ele not found.
    $T \propto n \rightarrow O(n)$
  - Best case: Min num of itrs
    → finding first element. e.g. 88
    $T = k \rightarrow O(1)$
  - Average case: Avg num of itrs
    → finding ele in between - multiple runs.
    → itrs = $\frac{n}{2}$
    → $T \propto \frac{n}{2} \rightarrow T \propto n \rightarrow O(n)$

Binary Search

key = 77

$$2^i = n$$

$$i = \frac{\log n}{\log 2}$$

$$T \propto \frac{\log n}{\log 2}$$

$$T \propto \log n$$

$$O(\log n)$$

0	1	2	3	4	5	6	7	8
11	22	33	44	55	66	77	88	99

L, m, R

if l > r, invalid Partn.

Problem Solving Techniques

1. Divide & Conquer
2. Greedy
3. Dynamic ✓

recursion

```
while (l<=r) {
    m = (l + r)/2
    if( key == arr[m])
        return m;
    if( key < a[m])
        r = m-1;
    else // key > a[m]
        l = m+1;
}
return -1;
```

Recursion

A1 : Convert dec to bin
A2 : bin search
A3 : print nums 1 to 10.

- Function calling itself is called as recursive function.
- To write recursive function consider
  - Explain process/formula in terms of itself
  - Decide the end/terminating condition

base

- Examples:
  - $n! = n * (n-1)!$ $\quad 0! = 1$
  - $x^y = X * x^{y-1}$ $\quad x^0 = 1$ (optimal)
  - $T_n = T_{n-1} + T_{n-2}$ $\quad T_1 = T_2 = 1$
  - factors(n) = 1st prime factor of n * factors(n)

↳ (1) $\quad 24 = 2 \times 2 \times 2 \times 3$

- On each function call, function activation record or stack frame will be created on stack.

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r= n * fact(n-1);
  return r;
}
res=fact(5);
```

optional input?

0! = 1

recursion
- program ?
- execution ? (internals)

3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!
0! = 1

```
int power(int x, int y) {
  if(y==0)
    return 1;
  r = x * power(x,y-1);
  return r;
}
```

```
int fib(int n){
  if(n==1 || n==2)
    return 1;
  r = fib(n-1)+fib(n-2);
  return r;
}
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>