Data Structure & Algorithms

Trainer: Nilesh Ghule

Course Introduction

1. Backtracking
2. Greedy method
3. Dynamic programming
4. Divide & Conquer

- Data Structure and Algorithms
  - Data Structures: Linked list, Stack, Queue, Binary search tree, Heap, Graph. Hash Table
  - Algorithms: Sorting, Searching, Stack, Queue & Linked list applications, Graph algorithms.
- Course Goals
  - Implement each DS & Algorithms from scratch.
  - Understand complexity of algorithms.
- Course Schedules
  - 16th Feb 2023 to 4th Mar 2023 (*)
  - Mon-Sat: Lecture – ~~5~~:00 PM to ~~8~~:00 PM
- Resource sharing
  - https://github.com/nilesh-g/dsa-06
  - Recorded videos will be available for 7 days. http://students.sunbeamapps.org

15 days * 4 hrs

5th/6th (reserved)

4 pm

8 pm

- 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

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

  } efficiently
- Common data structures
  - Array
  - Linked List
  - Stack
  - Queue
  - Hash Table
- Advanced data structures
  - Tree
  - Heap
  - Graph

Data Structure

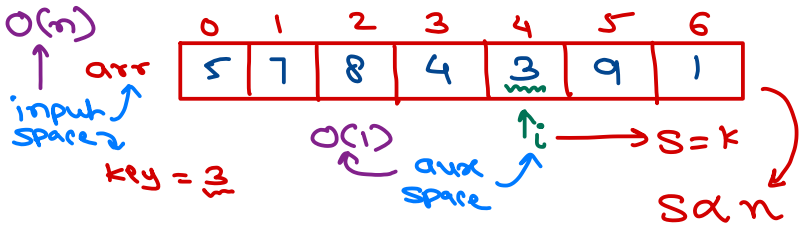
- Data Structure
  - Organizing data in memory
  - Processing the data
- Common data structures
  - Array
  - Linked List
  - Stack
  - Queue
  - HashTable
- Advanced data structures
  - Tree
  - Heap
  - Graph

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

time analysis : msec or usec → speed/clock, cores, ...
space analysis : bytes → architecture → int
gcc, vc, x86 → 4
arm gcc → 4
TC, avr gcc → 2
approx measure w.r.t. few params.

- 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)

S=k S∝n S∝n²



- Time complexity
  - Unit time required to complete any algorithm.
  - Approximate measure of time required to complete any algorithm.
  - Depends on loops in the algorithm. → num of 'iterations'
  - O(n³), O(n²), O(n log n), O(n), O(log n), O(1)

T=k
T∝log n
T∝n

Time complexity

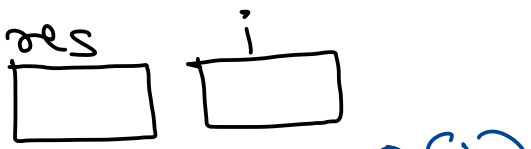
Space complexity

- Write a program to calculate factorial of given number.

```
res = 1;
for(i=1; i<=n; i++)
    res = res * i;
print(res);
```

$T \propto n$ → $O(n)$

aux space



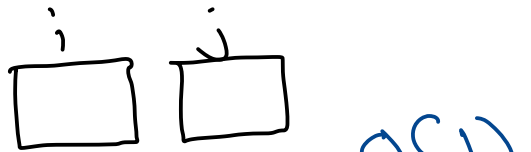
$S = k \rightarrow O(1)$

- Print 2-D matrix of n x n.

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

total itrs $n * n$

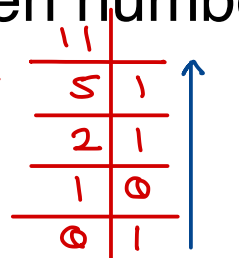
$T \propto n^2$ → $O(n^2)$



$S = k \rightarrow O(1)$

- Print given number into binary.

$11 = (1011)_2$




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

$2^{itr} = n$

$\log 2^{itr} = \log n$

$itr \log 2 = \log n$

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

$T \propto itrs$

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

$T \propto \log n$

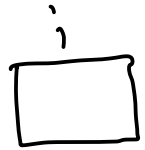
$O(\log n)$

$S = k \rightarrow O(1)$

- Print table of given number.

```
for(i=1; i<=10; i++)
    print(n * i);
```

$T = k$ → $O(1)$

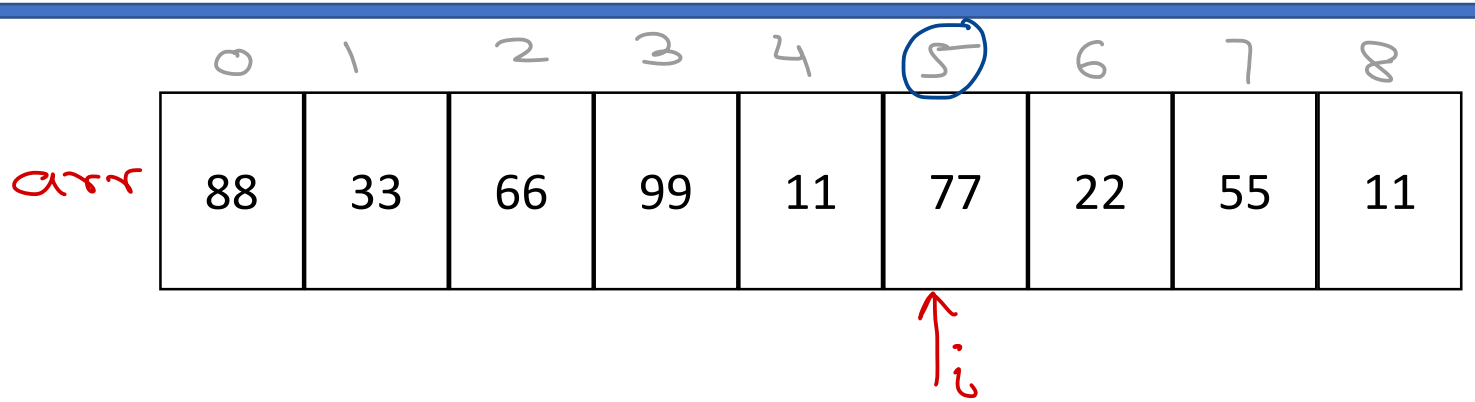


$S = k \rightarrow O(1)$

Linear Search

key = 77 → found at index 5

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

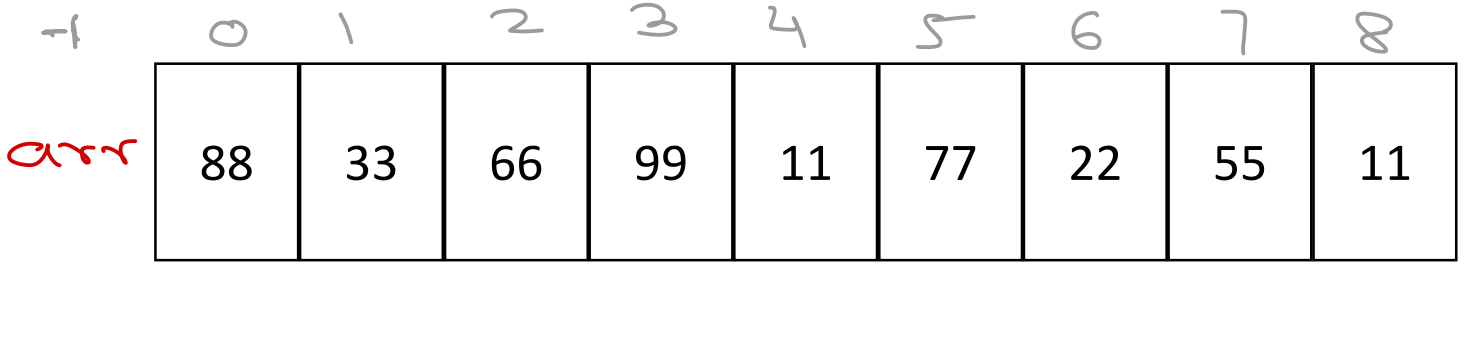


- Time complexity
  - Worst case
  - Best case
  - Average case

Linear Search

key = 70 → not found

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

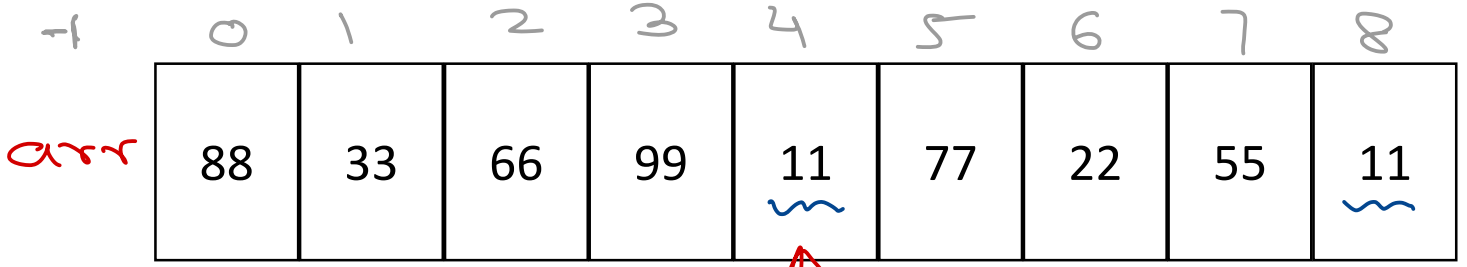


- Time complexity
  - Worst case
  - Best case
  - Average case

```
int linSearch(arr, key) {
    for(i=0; i<n; i++) {
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}
```

Linear Search

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



if element is repeated, the first occurrence index will return.

```
int linSearch(arr, key) {
    for(i=0; i<n; i++) {
        if(arr[i] == key){
            return i;
        }
    }
    return -1;
}
```

- Time complexity
  - Worst case ← key = 70 → n itrs → max itrs; T ∝ n; O(n)
  - Best case ← key = 88 → single itr → minimum itrs; T = k; O(1)
  - Average case ← random element find; T ∝ n/2; T ∝ n; O(n)

Binary Search

key = 88

- Possible only if array is already sorted.

```
int binSearch(arr, key) {
    l = 0;
    r = n-1;
    while( l <= r ) {
        m = (l+r) / 2;
        if(key == arr[m])
            return m;
        if(key > arr[m])
            l = m+1;
        else
            r = m-1;
    }
    return -1;
}
```

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

$2^{itr} \simeq n$

$\log 2^{itr} = \log n$

$itr \log 2 = \log n$

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

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

$T \propto \log n \longrightarrow O(\log n)$

best case
finding middle
single itr
$T = k \rightarrow O(1)$

avg/worst case
↳ not found.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>