Data Structure & Algorithms

Nilesh Ghule

Course Introduction

- Data Structure and Algorithms
  - Data Structures: Linked list, Stack, Queue, Binary search tree, Heap, Graph.
  - Algorithms: Sorting, Searching, Stack, Queue & Linked list applications, Graph algorithms.
- Course Goals
  - Implement each DS & Algorithms from scratch.
  - Understand complexity of algorithms.
- Course Schedules 10th
  - 12th Aug 2022 to ~~12th~~ Sep 2022
  - Mon-Fri: Lecture – 5:00 PM to 8:00 PM
- Resource sharing
  - https://gitlab.com/sunbeam-modular/dsa-05
    Recorded videos will be available for 7 days.
    http://students.sunbeamapps.org
- 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

Sunbeam youtube channel

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

ArrayList & HashMap & Stack

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 efficient
  - 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 → order of.

Exact Analysis → depend on machine, language. → CPU arch

1. how many bytes ?
2. how much time (sec) ?

- Space complexity
  - Unit space to store the data (Input space) and additional space to process the data (Auxiliary space).
  - O(1), O(n), $O(n^2)$

double arr[5];

1.1	2.2	3.3	4.4	5.5

$S \propto n$

Input Space → O(n)

sum

i

S = k

Aux space → O(1)

```
double sum=0.0;
for(int i=0; i<5; i++)
    sum = sum + arr[i];
```

- Time complexity
  - Unit time required to complete any algorithm.
  - (proportionate) Approximate measure of time required to complete any algorithm.
  - Depends on loops in the algorithm.
  - $O(n^3)$, $O(n^2)$, O(n log n), O(n), O(log n), O(1)

Time complexity

- Write a program to calculate factorial of given number.

f(n)

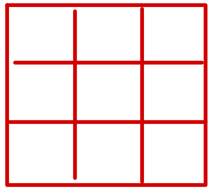
```
r = 1;
for(i=1; i<=n; i++)
    r = r * i
```

itrs = n

$T \alpha n$

$\boxed{O(n)}$

- Print 2-D matrix of n x n.



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

itrs = n * n

$T \alpha n^2$

$\boxed{O(n^2)}$

- Print given number into binary.

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

10 → 5 (0), 2 (1), 1 (0), 0 (1)

31 → 15 (1), 7 (1), 3 (1), 1 (1), 0 (1)

$2^{itr} = n$

$itr \log 2 = \log n$

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

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

$T \alpha \log n$

$\boxed{O(\log n)}$

- Print table of given number.

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

T = k

$\boxed{O(1)}$

Linear Search

key ← ele to find

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

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

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

- Time complexity
  - Worst case → max num of iterations
    → finding last ele or ele not exist in array. $\left.\begin{array}{l} itr = n \\ T \propto n \end{array}\right\} O(n)$
  - Best case → min num of iterations.
    → finding the first element → $\left.\begin{array}{l} itr = 1 \\ T = k \end{array}\right\} O(1)$
  - Average case → avg num of iterations
    → avg iterations $= n/2$
    $\left.\begin{array}{l} T \propto \frac{n}{2} \\ T \propto n \end{array}\right\} O(n)$

Binary Search

key = 50

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

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

$$2^{itr} = n$$

$$itr \log 2 = \log n$$

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

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

$$T \propto \log n$$

$$\boxed{O(\log n)}$$

Recursion

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

$2^3 = 2 * 2^2$
$2^2 = 2 * 2^1$
$2^1 = 2 * 2^0$
$2^0 = 1$

24 $4! = 4 * 3!$ 6
6 $3! = 3 * 2!$ 2
2 $2! = 2 * 1!$ 1
1 $1! = 1 * 0!$ 1
$0! = 1$

- 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);
```

Recursion

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

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

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

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

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

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

```
int main() {
    int res;
    res = fact(5);
    printf("%d", res);
    return 0;
}
```

120

stack

- ~~fact(0)~~
- ~~fact(1)~~
- ~~fact(2)~~
- ~~fact(3)~~
- ~~fact(4)~~
- ~~fact(5)~~
- ~~main()~~

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>