Lecture 3-4

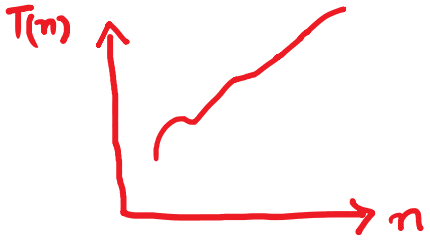
Friday, October 1, 2021 8:00 AM

Data Structure:

Storing the data in memory so that the data can be handled efficiently.

Operations / Functions → Traversal (or searching)
→ Insertion
→ Deletion

Complexity → Time
→ Space

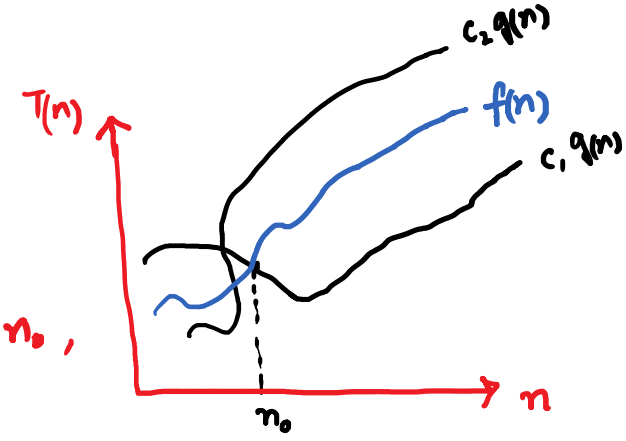


Asymptotic Notation :- θ, O, Ω

1) Theta Notation θ :

$\theta(g(n)) = \{ f(n)$: there exists positive constants, c_1, c_2, n_0, such that $\forall n \geq n_0$,

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$



eg:- if $f(n) = 2n+1$ $\quad g(n) = n$

$c_1 n \leq 2n+1 \leq c_2 n$

$c_1 = 2, \; c_2 = 3, \; n_0 \geq 1$

$2n \leq 2n+1 \leq 3n$

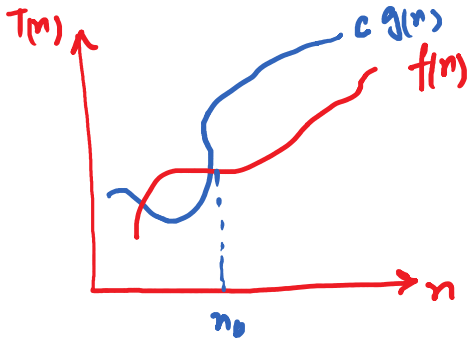
$f(n) = \theta(n)$

$f(n) \in \theta(n)$ ✓ $\quad f(n) \simeq \theta(n)$

Average case complexity

2) Big O notation

2) Big O notation



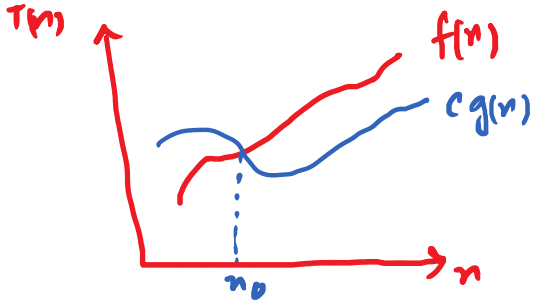
$O(g(n)) = \{ f(n)$: there exists positive constants c and n_0, such that $\forall n \geq n_0$,

$0 \leq f(n) \leq c\, g(n) \}$

Upper Bound — Worst case Complexity

eg:- $f(n) = 2n^2 + 4n - 6$ $\quad f(n) \in O(n^2)$

3) Omega notation Ω :



$\Omega(g(n)) = \{ f(n)$: there exists positive constants c & n_0, such that, $\forall n \geq n_0$

$0 \leq c\, g(n) \leq f(n) \}$

Lower Bound — Best case Complexity

eg:- $f(n) = n \log_2 n + 6n + 2$ $\quad f(n) \in \Omega(n \log_2 n)$

Benchmark functions :-

$\theta(1) < \theta(\log_2 n) < \theta(n) < \theta(n \log_2 n) < \theta(n^2) < \dots < \theta(n^n)$

Constant time — logarithmic time — linear time — quadratic time — $\theta(2^n)$, $\theta(n!)$ exponential time

$\theta(n^k)$ is polynomial if $k << n$

Is $P = NP$ or not ?

Types of Data Structures :-

Linear DS — Non Linear DS

Array, LinkedList, Stack, Queue — Tree, Graph

Array — Static Array — int a[100]; — Wastage of memory

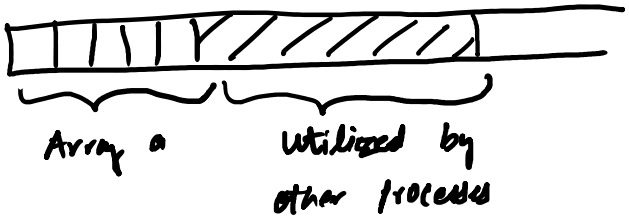
Dynamic Array - malloc

a = (int *) malloc (sizeof(int));

sizeof(char)
sizeof(int)
sizeof(float)

sizeof(int *)
sizeof(char *)
sizeof(float *)
— sizeof(int)

Array use contiguous memory allocation



Linked List uses non-contiguous memory allocation

Singly Linked List


```
struct node
{
  int data;
  struct node *next;
};
```

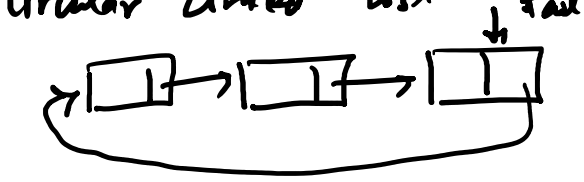
← self referential pointer

typedef struct node NODE;

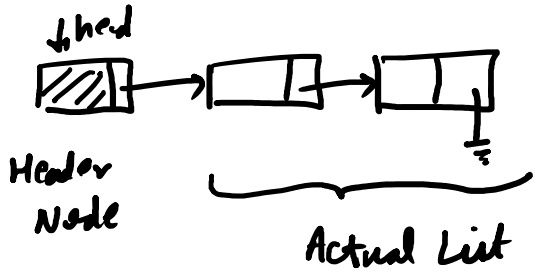
typedef int Rahul;

Rahul a = 10;

Circular Linked List

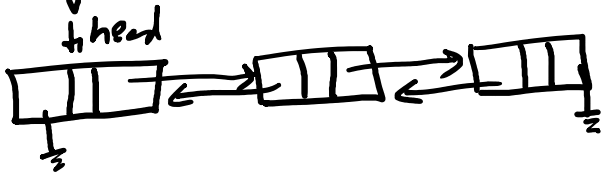


Header node based Linked List



Global Information

Doubly Linked List



union

int

float

char

struct

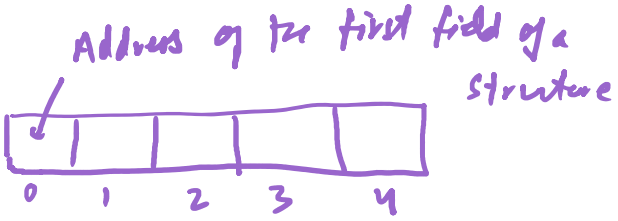
int

float

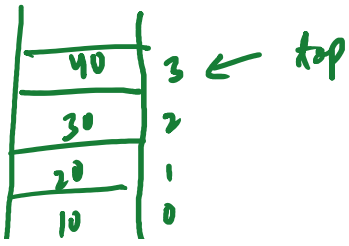
char

a

struct node obj[10];



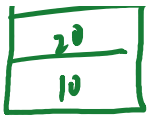
Stacks :- LIFO, top



A[2] is not accessible

Only A[top] is accessible

Push, Pop

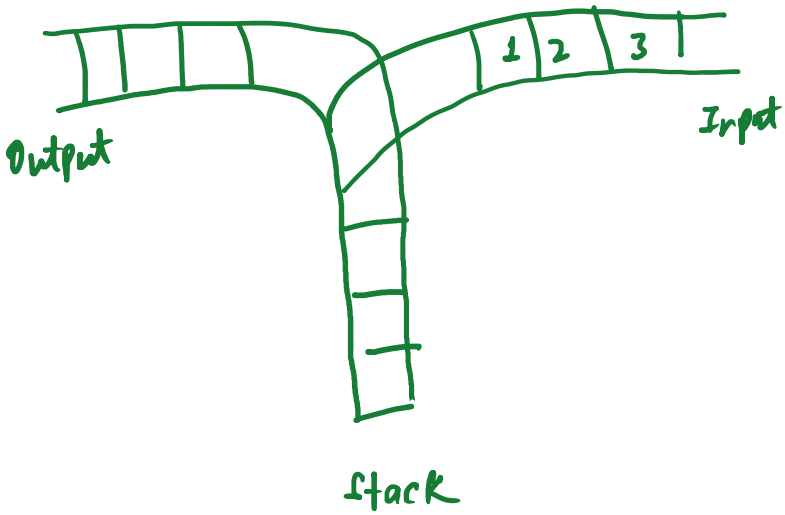


A

Push, Pop

Conversions of Notation Standards: Infix, Prefix, Postfix

eg:-



1 2 3 ✓

1 3 2 ✓

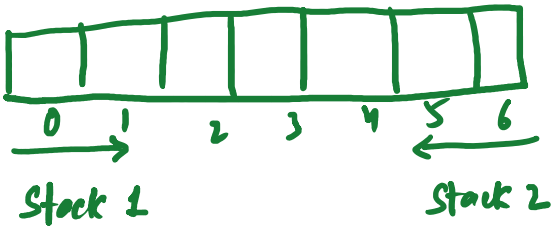
2 1 3 ✓

2 3 1 ✓

3 1 2 ✗

3 2 1 ✓

2 stacks in same memory:-

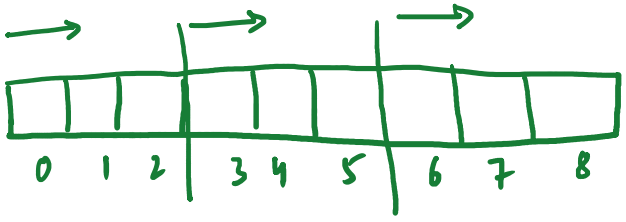


top1 = -1

top2 = 7

Overflow: if (top1 + 1 == top2)

n- stacks in common storage:-



top 1 = -1 base1 = -1

top 2 = 2 base2 = 2

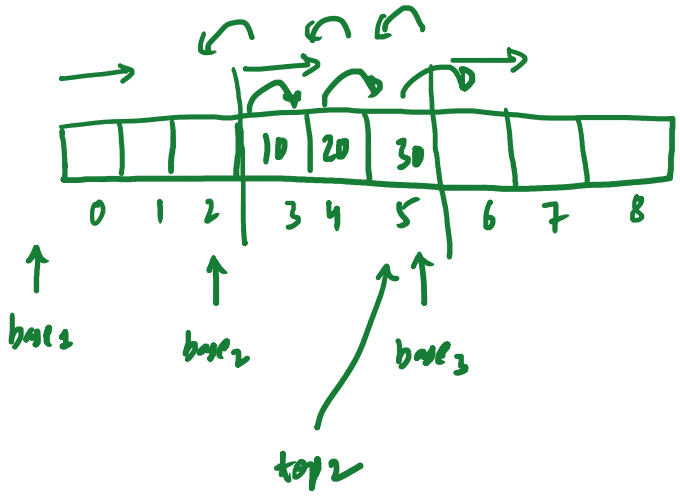
top 3 = 5 base3 = 5

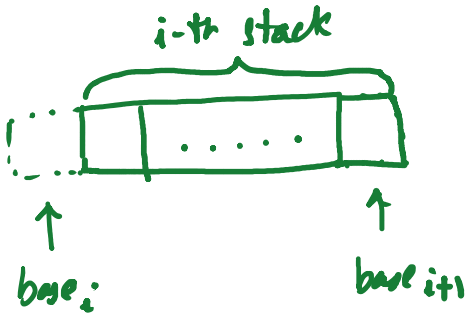
Initialization:

$base_i = top_i = (i-1)\frac{m}{n} - 1$

where n = # stacks

m = storage capacity





1) if ($top_i == base_i$)
i-th stack is empty

2) if ($top_i == base_{i+1}$)
i-th stack is full

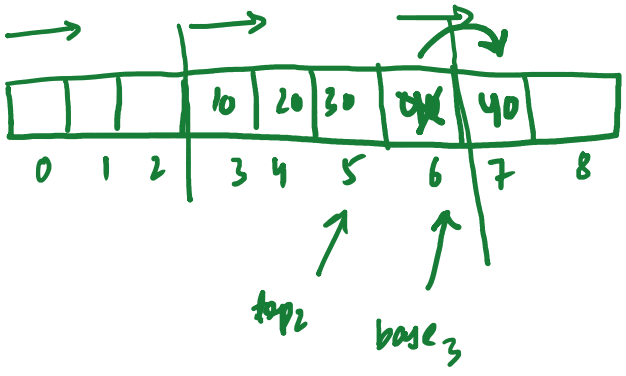
Load Overflow

a) Search space in i+1 th stack

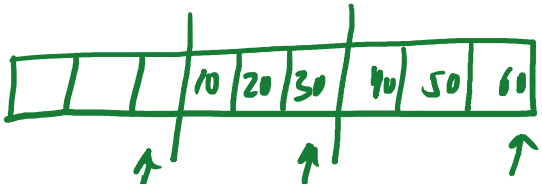
b) Search space in i-1 th stack

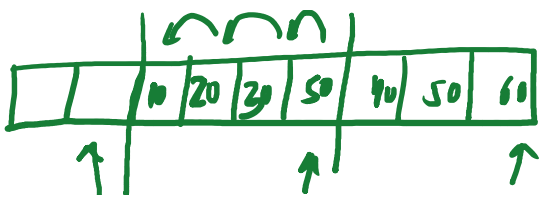
c) if no space in i+1 th and i-1 th stack then say "Overflow"

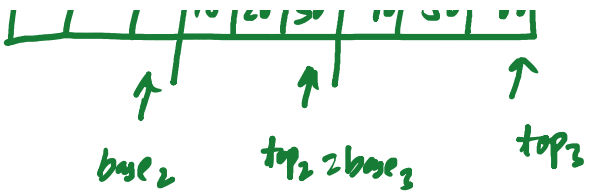
J. Garwick "Re packing algorithm". Donald Knuth vol 1

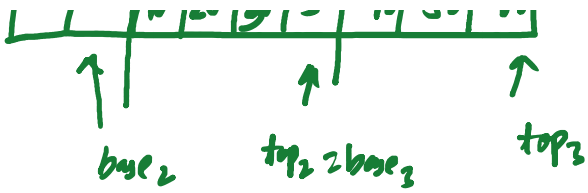


$base_i$ ++
shift the data towards right
$A[++top_2] = x$






base2
top2 = base3
top3


base2
top2 = base3
top3