

DATA STRUCTURES & ALGORITHMS (CS1004)

function oriented → supports lambda oriented functions

procedure oriented → doesn't support hoops concept

COURSE CONTENT :

- Review of elementary data structures
- analysis of recurrence relations

Linear Search → Best: 1st element.
Worst: last element.

Binary Search →

- if elements sorted, we can do.
- check if size of reqd. element is greater than / lesser than the middle element & search accordingly.

BOOK → M.A. Weiss, Data Structures and Algorithms.

EVALUATION :

CS1004

- 35 marks for pass, then on relative.
- 25 marks for continuous evaluation.
(10 marks assignment, 15 marks for surprise test)
- 25 marks for mid-sem.
- 50 marks for end-sem.

CS1005

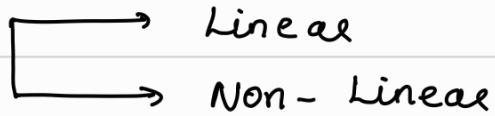
- min 35 marks, then on relative.
- 25 marks continuous evaluation.
(surprise test or viva)
- 25 marks for mid-sem.
- 50 marks for end-sem.

ELEMENTARY DATA STRUCTURES

- Data Structure → way of organizing & storing data in memory / external storage device.
→ helps facilitate insertion, retrieval and manipulation of data.

Volume, Variety and Velocity (VVV) of data

DATA STRUCTURES



- Linear :
- Allows data elements to be arranged in a sequential / linear fashion.
 - each element → attached to previous and next element.
 - E.g. Array, Stack, Queue, Linked List.

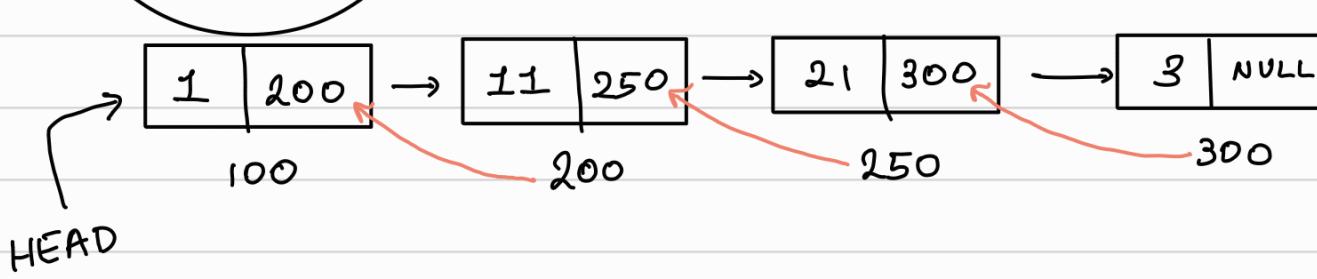
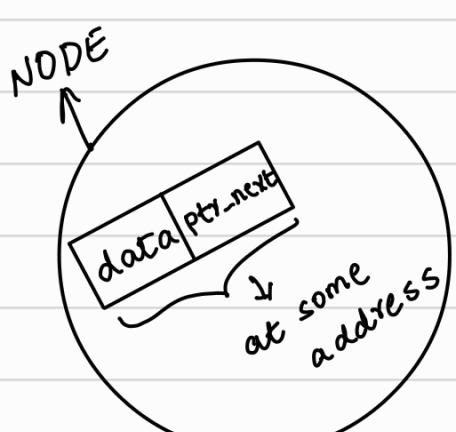
- Non-Linear :
- Data isn't arranged sequentially
 - can't traverse thru all elements in single run.
 - E.g. Graphs and Trees.

- ARRAY →
- Static memory allocation → can't add more elements
 - stores only one data-type
 - deletion of data / insertion of data is cumbersome.

SO we use *Linked List*

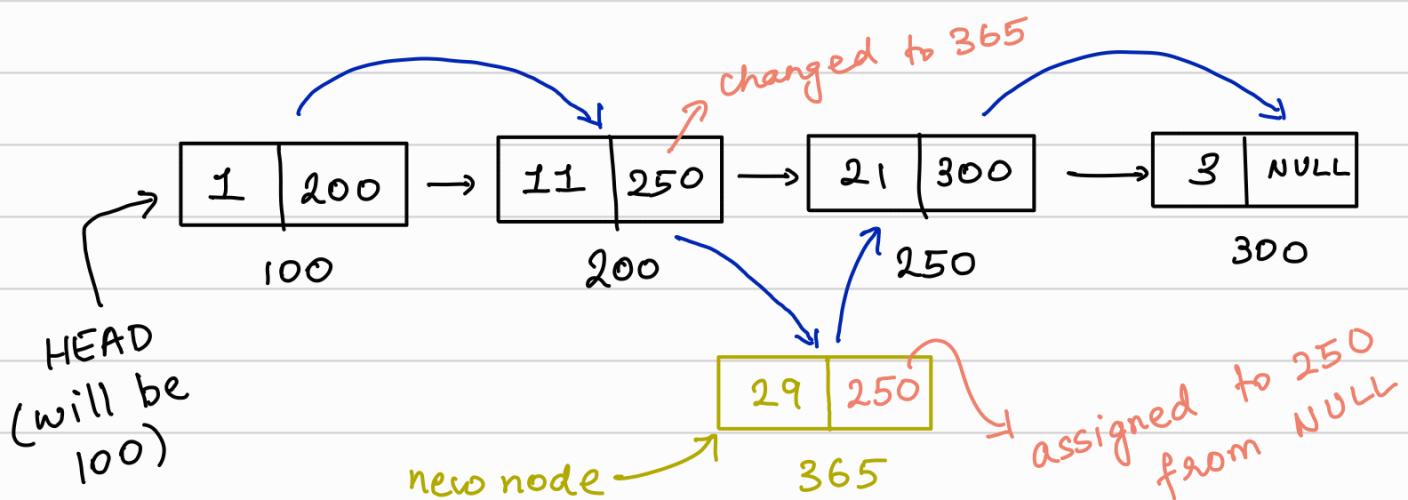
- LINKED LIST →
- Linear Data Structures in which the elements (nodes) are connected together via pointers (or) references.

- They allow Dynamic Memory Allocation, and data aren't stored in a contiguous manner. This makes insertion & deletion of elements more flexible.



GRAPHS → collection of edges and vertices forming a loop.

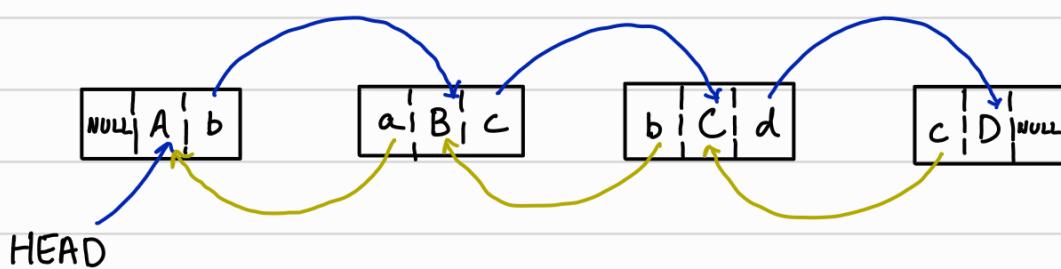
SINGLE LINKED LIST : • singly linked list if elements are not stored in contiguous memory locations, & each node is ONLY connected to its next element, using pointers.



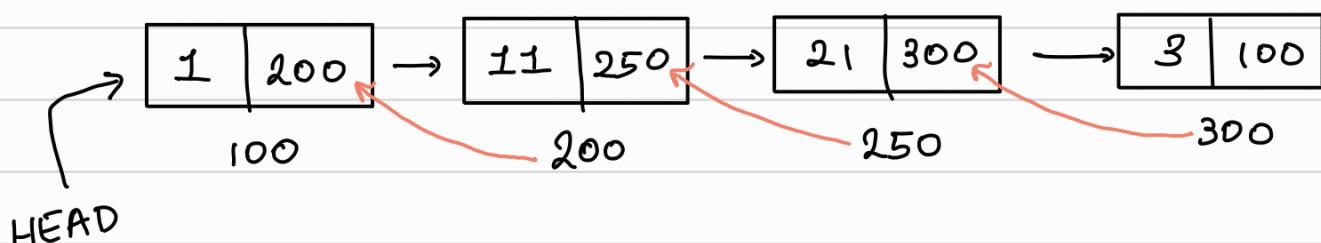
▲ To add a new node / data at ANY RANDOM POINT.

DRAWBACK of SINGLE LINKED LIST : MOVE only in forward direction.

DOUBLE LINKED LIST : • traverse forward and backward



CIRCULAR LINKED LIST : Last node is connected to first



STACK : follows Last In First Out (LIFO)
(or)

First In Last Out (FILO)

When stack is empty \rightarrow $\text{top} = -1$
 to insert element \rightarrow push (check if stack is full/not)
 to delete element \rightarrow pop

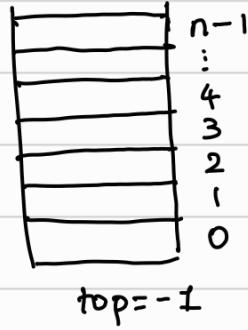
PUSH

stack is full if : $\text{top} = \text{size} - 1$

if $\text{top} \neq \text{size} - 1$, then :

$\text{top}++;$

$a[\text{top}] = \text{variable}$.



if $\text{top} = \text{some } K$, then some var
 is stored at K . so, $\text{top}++$ (goes above)
 and then push.
 then $\text{printf}("%d", a[\text{top}]);$

POP

stack is empty if : $\text{top} = -1$.

if $\text{top} \neq -1$, then remove the element

(can use linked list \rightarrow yet to be taught)

(or) $\text{printf}("%d", a[\text{top}]);$

$\text{top}--;$

Recursion \rightarrow implemented using stack.

find $5!$

$f(1)$
$f(2)$
$f(3)$
$f(4)$
$f(5)$

$$\begin{aligned}
 f(1) &= 1 \\
 2 \times f(1) &= 2 \times 1 = 2 = f(2) \\
 3 \times f(2) &= 3 \times 2 = 6 = f(3) \\
 4 \times f(3) &= 4 \times 6 = 24 = f(4) \\
 5 \times f(4) &= 24 \times 5 = 120 = f(5)
 \end{aligned}$$

$$\therefore f(5) = 120$$

defining a node :

Struct node

{

int data;

Struct node* next;

}

Struct node* head = NULL;

Create node

```
Struct node * newnode, *temp;
newnode = (struct node*) malloc(sizeof(node));
printf("enter value");
scanf("%d", &newnode->data);
newnode->next = NULL;
if (head == NULL)
{
    head = newnode;
    temp = head;
}
else
{
    temp->next = newnode;
    temp = temp->next;
}
```

print data

```
void display()
{
    Struct node *temp = head;
    if (head == NULL) { printf("empty"); }
    else
    {
        while (temp != NULL)
        {
            printf("%d\n", temp->data);
            temp = temp->next;
        }
    }
}
```

WACP : to insert a new node at the beginning
of a linked list.

void begin()

```
{
```

```
Struct node *newnode;
newnode = (struct node*) malloc(sizeof(node));
scanf("%d", &newnode->data);
newnode->next = head;
head = newnode;
```

}

WACP: insert at end.

Void end ()

{

struct node * newnode = (struct node*) malloc
(sizeof
struct
node));

scanf("%d", &newnode->data);

newnode->next = NULL;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newnode;

}

WACP: to insert node at a random position.

① count nodes:

Void add()

{

int count = 0;

struct node * temp = head;

while (temp != NULL)

{

count++;

temp = temp->next;

}

if (n > count)

{

printf("N.A");

}

else

{

struct node * newnode = (struct node*)

malloc(sizeof(struct
node));

scanf("%d", &newnode->data);

```

newnode->next = NULL; temp [] []
for(int i=1; i<n; i++)
{
    temp = temp->next;
}
newnode->next = temp->next
[] [] new

```

This only works if position is not at the end.

deletion :

- assume list is created

- Struct node

{

int data;

Struct node *next;

};

Struct node *head = NULL;

(a) delete 1st:

```

if [head->next] ← { Struct node * temp = head ;
                      head = temp->next;
                      free(temp);
                      temp = NULL;
}

```

```

if [head->next] ← { free(head);
                      head = NULL;
}

```

if [head=NULL] ← { enjoy }

(b) delete at end:

```

Struct node * temp = head;
Struct node * prevnode = NULL;
while (temp->next != NULL)
{

```

prevnode = temp;
temp = temp->next;

}

free(temp);
temp = NULL;
prevnode->next = NULL;

③ delete i'th index;

Assuming $i < \text{count}$

```
struct node *temp = head, *prevnode = NULL;
int j = 1;
while (j < i)
{
    prevnode = temp;
    temp = temp->next;
    j++;
}
prevnode->next = temp->next;
free(temp);
temp = NULL;
```

if $i = 3$
 $1 \rightarrow p = 1 \ j = 1$
 $t = 2$
 $2 \rightarrow p = 2 \ j = 2$
 $t = 3$
↓
We have
the node
to be
deleted.

else → printf ("poda loosu");
return 0;

Count → got by :

```
int count = 0; temp = head
while (temp != NULL)
{
    count++;
    temp = temp->next;
}
```

display elements in reverse order:
(assume list is ready)

```
struct node *temp = head, *prevnode = NULL,
temp2 = head;
while (temp2 != NULL)
{
    temp2 = temp2->next;
    temp->next = prevnode;
    head = temp;
    prevnode = temp;
    temp = temp2;
}
head = prevnode;
```

take 1 100 2 200 3 NULL
 50 100 200

head = 50.

1st : temp2 = 100;
temp->next = NULL;
prevnode = temp = 50
temp = temp2 = 100.

1	NULL	2	200
50	100		
3	NULL	200	

2nd: $\text{temp}2 = 200;$
 $\text{temp} \rightarrow \text{next} = 50;$
 $\text{prevnode} = 100;$
 $\text{temp} = 200;$

1	NULL	2	50
50		100	
	3	NULL	
		200	

3rd $\text{temp}2 = \text{NULL};$
 $\text{temp} \rightarrow \text{next} = 100;$
 $\text{prevnode} = 200;$
 $\text{temp} = \text{NULL};$

1	NULL	2	50	3	100
50		100		200	

IMPLEMENTATION of STACK using Arrays

PUSH

```

int x;
if (top == size - 1)
{
    printf ("stack is full \n");
}
else
{
    printf ("enter value: ");
    scanf ("%d", &x);
    top++;
    a[top] = x;
}
    
```

size \rightarrow size of stack.
top \rightarrow where at stack the last value lies.

if n(elements) = 0,
then top = -1
(initially)

POP

top--; if top == -1 \rightarrow underflow

PEAK

if (top == -1) \rightarrow empty

printf ("%d", a[top]);

DISPLAY

```

temp = top;
while (temp != -1)
{
    printf ("%d", a[temp - 1]);
}
    
```

IMPLEMENTATION of STACK using Linked List

struct node

{

int data;

struct node * next;

}

Struct node * top = NULL;

push (or) pop in the linked list → at first node.

as push / pop → O(1)

PUSH :

struct node * newnode = (struct node *) malloc
(sizeof(struct node));

newnode → next = top;

scanf("%d", &newnode → data);

top = newnode;

POP :

- if (top == NULL) → empty.

else

{

struct node * temp = NULL;

temp = top;

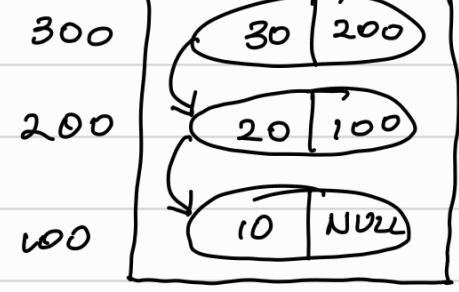
top = top → next

temp → next = NULL;

free(temp);

temp = NULL;

}



DOUBLY LINKED LIST

* addr. of prevnode, nextnode + the data itself.

struct node

{

struct node * prev, * next;

int data;

}

Struct node * head = NULL, * tail = NULL;

create node

```

struct node *newnode = (struct node*) malloc (sizeof(
    struct node));
scanf ("%d", &newnode->data);
newnode->next= NULL;
newnode->prev= NULL;
if (head == NULL)
{
    head = tail = newnode;
}
else
{
    tail->next = newnode;
    newnode->prev= tail;
    tail = tail->next;
}

```

display elements

```

temp = head;
while (temp != NULL)
{
    printf ("%d", temp->data);
    temp = temp->next;
}

```

} → forward dir.

```

temp = head;
while (temp->next != NULL)
{
    temp = temp->next;
}
while (temp != NULL)
{
    printf ("%d", temp->prev);
    temp = temp->prev;
}

```

} → backward dir
(dumb)

```

temp = tail;
while (temp != NULL)
{
    printf ("%d", temp->data);
    temp = temp->prev;
}

```

} → backward dir
(genius)

Insert newnode

(a) At beginning :

```
struct node *newnode = (struct node*) malloc (sizeof( struct node));
```

```
scanf ("%d", &newnode->data);
```

```
newnode->next = NULL;
```

```
newnode->prev = NULL;
```

```
if (head == NULL)
```

```
{
```

```
    head = tail = newnode;
```

```
}
```

```
else
```

```
{
```

```
    head->prev = newnode;
```

```
    newnode->next = head;
```

```
    head = newnode;
```

```
}
```

(b) At end :

```
struct node *newnode = (struct node*) malloc (sizeof( struct node));
```

```
scanf ("%d", &newnode->data);
```

```
newnode->next = NULL;
```

```
newnode->prev = NULL;
```

```
tail->next = newnode;
```

```
newnode->prev = tail;
```

```
tail = newnode;
```

(c) at i'th index :

```
int i, count = 0;
```

```
scanf ("%d", &i);
```

```
temp = head;
```

```
while (temp != NULL)
```

```
{
```

```
    count++;
```

```
    temp = temp->next;
```

```
}
```

```

if (i > count)
{
    printf ("gay");
}
else
{
    temp = head;
    for (int j = 1; j < i; j++)
    {
        temp = temp->next;
    }
    struct node *newnode = (struct node *) malloc (sizeof(
        struct node));
    scanf ("%d", &newnode->data);
    newnode->next = NULL;
    newnode->prev = NULL;
    newnode->prev = temp;
    newnode->next = temp->next;
    temp->next = newnode;
    temp->next->prev = newnode;
}

```

delete nodes

(a) at beginning :

```

temp = head;
head = temp->next;
temp->next = NULL;
temp->prev = NULL;
free(temp);

```

(b) At end :

```

temp = tail;
tail = temp->prev;
temp->next = NULL;
temp->prev = NULL;
free(temp);

```

P. T. O

(c) i'th index :

```
int i, count = 0;  
scanf("%d", &i);  
temp = head;  
while (temp != NULL)  
{
```

count++;
temp = temp->next;

۳

if ($i > \text{count}$)

۲

```
printf("gay");
```

3

else

2

temp = head ; temp2 = NULL;

```
for (int j=1 ; j<i ; j++)
```

۹

Temp2 = temp;

temp = temp → next;

3

$\text{temp2} \rightarrow \text{next} = \text{temp} \rightarrow \text{next};$

`temp->next->prev = temp2;`

$\text{temp} \rightarrow \text{next} = \text{NULL};$

`temp → prev = NULL;`

`free(temp);`

۳

we stored
temp → prev
as temp 2

REVERSE doubly linked list

(SOLVE as H.W.) tail = head;

head = tail. bruv.
foo easi

CIRCULAR LINKED LIST

for single L.L.

Struct node

{

 int data;

 Struct node *next;

}

Struct node *head, *tail;

head = tail = NULL;

Create node :

```

struct node *newnode = (struct node*) malloc
    (sizeof(struct node));
scanf("%d", &newnode->data);
newnode->next = NULL;
if (head == NULL)
{
    head = tail = newnode;
    tail->next = head;
}
else
{
    tail->next = newnode;
    tail = newnode;
    tail->next = head;
}

```

insert newnode at beginning :

```

newnode = (struct node*) malloc (sizeof (struct node));
newnode->next = head;
head = newnode;
scanf ("%d", &newnode->data);
tail->next = head;

```

after i'th index → same as single linked list
 (the difference is only in the first & last node)

```

temp = head;
do
{
    i++;
    temp = temp->next;
} while (temp != head);

```

Delete nodes

ⓐ Beginning : $tail \rightarrow next = head \rightarrow next;$

$head \rightarrow next = NULL;$

`free(head)`

① check if $head == NULL$

② check if $head \rightarrow next == head$

③ else

$head = tail \rightarrow next;$

(b) end : $\text{temp} = \text{head}; \text{temp2} = \text{head}$
 check if
 ① $\text{head} = \text{NULL}$; {
 ② $\text{head} \rightarrow \text{next} = \text{head}$; }
 $\text{temp2} = \text{temp};$
 $\text{temp} = \text{temp} \rightarrow \text{next};$
 $\text{temp2} \rightarrow \text{next} = \text{temp} \rightarrow \text{next};$
 $\text{temp} \rightarrow \text{next} = \text{NULL};$
 $\text{free}(\text{temp});$
 $\text{temp} = \text{NULL};$

QUEUE

* using Linked list :

enqueue - $\text{int } x;$
 $\text{newnode} = (\text{struct node} *)$
 $\text{malloc}(\text{sizeof}(\text{struct node}));$
 $\text{scanf}("%d", &x)$
 $\text{newnode} \rightarrow \text{data} = x;$
 $\text{newnode} \rightarrow \text{next} = \text{NULL};$
 $\text{if } (\text{front} == \text{NULL})$
{
 $\text{front} = \text{rear} = \text{newnode};$
}
 else
{
 $\text{rear} \rightarrow \text{next} = \text{newnode};$
 $\text{rear} = \text{newnode};$
}

dequeue - $\text{if } (\text{front} == \text{NULL})$
{
 $\text{printf}("empty\n");$
}
 $\text{else if } (\text{front} \rightarrow \text{next} == \text{NULL})$
{
 $\text{front} = \text{rear} = \text{NULL};$
}
 else
{
 $\text{temp} = \text{front};$
 $\text{front} = \text{front} \rightarrow \text{next};$
 $\text{temp} \rightarrow \text{next} = \text{NULL};$
}

free(temp);
}

display -

```
temp = front;  
while (temp != NULL)  
{  
    printf("%d", temp->data);  
    temp = temp->next;  
}
```

peak -

```
printf("%d", front->data);
```

EFFICIENCY OF ALGORITHMS

- * determined wrt time (time complexity) and storage (space complexity)
- * measured with the help of asymptotic notations.
- * Algorithms - diff performance with diff inputs.

POSTERIOR ANALYSIS - evaluates actual performance of the implemented algorithm based on real-world data (or) empirical measurements

- usually not done as its not universal (diff. computers may give diff. measurements)

PRIOR ANALYSIS - Predicts performance of an algorithm before its execution, often using theoretical models & asymptotic analysis

- we use this.

P.T.O.

CLASSIFICATION of ALGORITHMS :

- * Iterative : computational procedure that repeats a set of instructions until a condition is met.
- * Recursive : calls a function repeatedly.

TIME & SPACE COMPLEXITY

- * Time : measure of time of execution as a fn. of sizeof(input)
- * Space : space taken by the algorithm wrt its input size. Includes auxiliary space & space used by input.

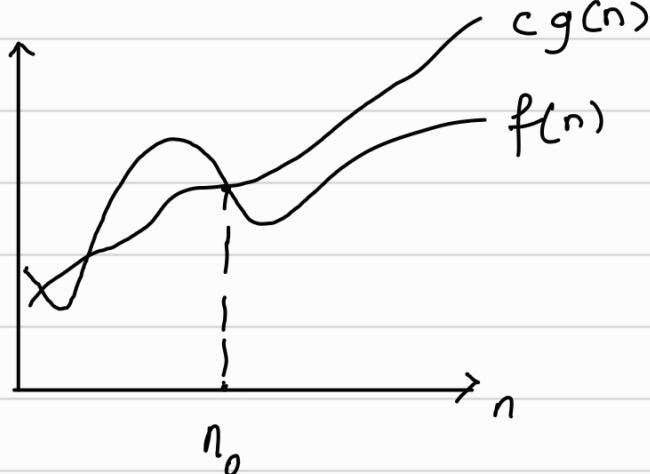
ASYMPTOTIC NOTATION

- * mathematical notation - used to describe limiting behaviour of a fn. as the input $\rightarrow \infty$.
- * Big O, big Omega, big Theta.

$$g(n) = O(f(n)) \rightarrow \text{if } g(n) \leq c f(n);$$

Big O \rightarrow upper bound / worst case scenario.

$$O(g(n)) = \{f(n) : \exists c, n_0 \geq 0, n_0 \geq 1 \text{ s.t. } \forall n \geq n_0, 0 \leq f(n) \leq c g(n)\}$$



Set of all fns. whose rate of growth is same as or lower than that of $g(n)$.

$$f(n) = O(g(n)).$$

Big Omega Notation (Ω) :

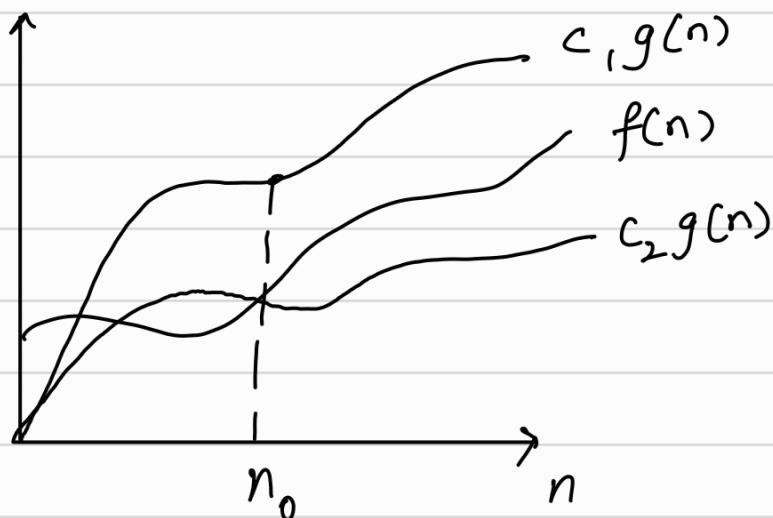
- * $\Omega(g(n))$ represents a lower bound on the growth rate of a function.
- * $\Omega(g(n)) = \{f(n) : \exists c, n_0 \geq 0, n_0 \geq 1, 0 \leq c g(n) \leq f(n)\}$
set of all functions whose rate is same as or higher than that of $g(n)$.
then, $f(n) = \Omega(g(n))$

Big Theta (Θ):

- $\Theta(f(n)) \rightarrow$ upper & lower bounds of the time complexity.
- provides a tight bound.

$$c_2 g(n) \leq f(n) \leq c_1 g(n) \quad \forall n \geq n_0, n_0 \geq 1$$

$\text{if } c_1, c_2 \geq 0$



$$\Theta(g(n)) = \{f(n) : \text{if } c_1, c_2, n_0 \geq 0, n_0 \geq 1 \text{ s.t. } \forall n \geq n_0, 0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n)\}$$

(WORK ON THIS LATER.)

```

A(n)
{
    i=1, s=1;
    while(s <= n)
    {
        i++;
        s+=i;
        printf("save me\n");
    }
}

```

$i=1, s=1 \downarrow^2$
 ① $i=2, s=3 \downarrow^3$
 ② $i=3, s=6 \downarrow^3$
 ③ $i=4, s=10 \downarrow^4$
 ④ $i=5, s=15 \downarrow^5$

$s_n = 1, 3, 6, 10, 15$

A(n)

```

{
    i=1;
    for(i; i*i <= n; i++)
    {
        printf("save me\n");
    }
}

```

$i^2 \leq n$
 $\Rightarrow i \leq \sqrt{n}$

So, $O(\sqrt{n})$

A(n)

{

```

int i,j,k,n;
for(i=1; i<=n; i++)
{
    for(j=1; j<=i; jj++)
    {
        for(k=1; k<=100; k++)
    }
}

```

i	1	2	3	...	n
j	1	2	3	...	n
k	100	2x100	3x100	...	nx100

$$100(1+2+\dots+n)$$

$$= 100 \times \frac{n(n+1)}{2}$$

$$= 50(n^2+n)$$

$$n^2+n \approx n^2$$

}

~~O(n²)~~

A(n)

{

```

int i,j,k,n;
for(i=1; i<=n; i++)
{
    for(j=1; j<=i*i; j++)
    {
        for(k=1; k<=n/2; k++)
    }
}

```

i	1	2	...	k	...	n
j	1	4		k^2	...	n^2
k	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$

$$\frac{n}{2} + 4 \times \frac{n}{2} + 9 \times \frac{n}{2}$$

$$+ \dots + n^2 \times \frac{n}{2}$$

$$\frac{n}{2} (1 + 4 + 9 + \dots + n^2)$$

$$\frac{n}{2} \left(\frac{n(n+1)(2n+1)}{6} \right)$$

$$\underline{\underline{O(n^4)}}$$

A(n)

{

```

for(i=1; i<=n; i*=2)
{
}

```

1	2	4	8	16	...	2^{n-1}
---	---	---	---	----	-----	-----------

printf("Save me\n");

Let loop run x times

$$2^{x-1} \leq n$$

$$\Rightarrow x-1 \leq \log_2 n$$

$$\Rightarrow x \leq \log_2 n - 1$$

~~O(log₂n)~~

A(n)

{

int i, j, k;

for (i = n/2; i <= n; i++)

{

for (j = 1; j <= n/2; j++)

{

for (k = 1; k <= n; k *= 2)

{

printf("Save me\n");

}

}

}

i → n/2

j → n/2

k → log₂ n

$$\frac{n/2 \times n/2 \times \log_2 n}{}$$

O(n² log n)

A(n)

{

int i, j, k;

for (i = n/2; i <= n; i++)

{

for (j = 1; j <= n; j *= 2)

{

for (k = 1; k <= n; k *= 2)

{

printf("Save me\n");

}

}

$$\frac{n/2 \cdot \log_2 n \cdot \log_2 n}{}$$

O(n log² n)

A(n)

{

for (i = 1; i <= n; i++)

{

i = 1 2 3 4 ... n

j = 0 0 0 0 0

for (j = 0; j <= n; j = j + i)

{

1 2 3 4 K

2 4 6 8 2K

3 6 9 12 :

:

⋮ ⋮ ⋮ ⋮ ⋮

n n n n n

n $\frac{n}{2}$ $\frac{n}{3}$ $\frac{n}{4}$... $\frac{n}{n}$

$$n \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} \right)$$

$$\frac{1(x^n - 1)}{x-1} = 1 + x + x^2 + \dots$$

$$\int \frac{dx}{x-1} = \int (1 + x + x^2 + \dots) dx$$

$$\Rightarrow \ln(x-1) = 1 + \frac{x}{2} + \frac{x^3}{3} + \dots$$

order $\rightarrow O(\log n)$.

So, net $\equiv \underline{\underline{O(n \log n)}}$

$A(n)$

{

$i=1; s=1;$

$i: 1 2 3 4 5 6 \dots K$

while ($s \leq n$)

$s: 1 3 6 10 15 21 \dots$

{

$i++;$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \curvearrowright & \curvearrowright & \curvearrowright & \curvearrowright & \curvearrowright & \curvearrowright \\ 1 & 3 & 6 & 10 & 15 & 21 \end{matrix}$

$s+=ij$

$n^{\text{th}} \text{ value of } s$

$\text{printf("Save me \%n");}$

$$s_n = 1+2+3+\dots+n$$

}

$$= \frac{n(n+1)}{2}$$

$$s_K = \frac{K(K+1)}{2}$$

$$s_K \leq n$$

$$\Rightarrow \frac{K(K+1)}{2} \leq n$$

$$\Rightarrow \frac{k^2}{2} \leq n$$

$$\Rightarrow K \leq \sqrt{n}$$

complexity
 $O(\sqrt{n})$

$A(n)$

{

while ($n \geq 1$)

$$n = K, K/2, K/4, \dots K/2^{n-1}$$

{

$\text{printf("Save me \%n");}$

$$\frac{K}{2^{n-1}} < 1$$

$$n = n/2;$$

$$\Rightarrow K < 2^{n-1}$$

}

$$n = n/2;$$

$$\Rightarrow \log_2 K < n$$

complexity $\rightarrow O(\log_2 n)$

RECURSIVE ALGORITHMS:

- Back-Substitution mechanisms
- Recursive trees.
- Master's theorem → Advanced Master's theorem.
(newer one)

$A(n)$
 {
 if ($n > 1$)
 {
 return $A(n-1)$;
 }
 }

(if $n > 1$)
 [if $n = 1$
 ↓
 runs only
 ONCE.]

$$t(n) = 1 + t(n-1) \quad \textcircled{1}$$

$$t(n-1) = 1 + t(n-2) \quad \textcircled{2}$$

$$t(n-2) = 1 + t(n-3) \quad \textcircled{3}$$

② in ①

$$\Rightarrow t(n) = 1 + 1 + t(n-2) = 2 + t(n-2) \quad \textcircled{4}$$

③ in ④

$$\Rightarrow t(n) = 3 + t(n-3) \quad \textcircled{5}$$

\Rightarrow for some K ,

$$\Rightarrow t(n) = K + t(n-K) \quad \textcircled{6}$$

We know $t(1) = 1$

$$\Rightarrow n - K = 1$$

$$\Rightarrow K = n - 1$$

$$\therefore t(n) = (n-1) + t(1) \\ = (n-1) + 1$$

$$= n$$

$$\therefore t(n) = n$$

→ algorithm runs for ' n ' times.

$$\therefore \underline{\underline{O(n)}}$$

$$T(n) = n + T(n-1)$$

$$T(1) = 1 .$$

$$T(n-1) = (n-1) + T(n-2)$$

$$T(n-2) = (n-2) + T(n-3)$$

⋮

$$n - K = 1$$

$$\Rightarrow K = n - 1$$

$$T(n-K) = (n-K) + T(n-K-1)$$

⋮

$$T(1) = 1 + T(0)$$

$(\checkmark n-0) + (\checkmark n-1) + (\checkmark n-2) + \dots + (\checkmark n-(n-2)) + (\checkmark n-(n-1))$

\downarrow
n terms.

$$\begin{aligned}
 & n^2 - (1+2+\dots+n-1) \\
 &= n^2 - \frac{(n-1)(n)}{2} \\
 &= n^2 - \frac{n(n-1)}{2} = \frac{2n^2 - n^2 + n}{2} = \frac{n^2 + n}{2} \\
 &= \underline{\underline{\frac{n(n+1)}{2}}}
 \end{aligned}$$

complexity $\rightarrow O(n^2)$

SORTING TECHNIQUES

- ① Arrange elements in specific order.
- ① → Bubble Sort → Merge Sort
- Selection Sort → Heap Sort
- Insertion Sort → Counting Sort
- Quick Sort → Radix Sort

BUBBLE SORT:

Take $\rightarrow 5 \ 9 \ 2 \ 15 \ 6 \ 11 \rightarrow$ apply bubble sort to
sort by ascending
order

- ① $5 < 9 \rightarrow$ TRUE, no need to swap.
- ② $9 < 2 \rightarrow$ FALSE \rightarrow swap $(9, 2)$ $(5 \ 2 \ 9 \ 15 \ 6 \ 11)$
- ③ $9 < 15 \rightarrow$ TRUE, no swap
- ④ $15 < 6 \rightarrow$ FALSE \rightarrow swap $(15, 6)$ $(5 \ 2 \ 9 \ 6 \ 15 \ 11)$
- ⑤ $15 < 11 \rightarrow$ FALSE \rightarrow Swap $(15, 11)$ $(5 \ 2 \ 9 \ 6 \ 11 \ 15)$

\downarrow

go back to 1st index

- ⑥ $5 < 2 \rightarrow$ FALSE \rightarrow swap $(5, 2)$ $(2 \ 5 \ 9 \ 6 \ 11 \ 15)$
- ⑦ $5 < 9 \rightarrow$ TRUE
- ⑧ $9 < 6 \rightarrow$ FALSE \rightarrow swap $(9, 6)$ $(2 \ 5 \ 6 \ 9 \ 11 \ 15)$
- ⑨ $9 < 11 \rightarrow$ TRUE
- ⑩ $11 < 15 \rightarrow$ TRUE

\downarrow

go back to 1st index

~~REPEAT for $(n-1)$ times~~

Same for descending, except: check if current > next is true ...

void bubblesort(int arr[], int n)

{

int i, j;

for (i=0; i<n-1; i++)

{

for(j=0; j<n-i-1; j++)

{

if (arr[j] > arr[j+1])

{

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

i = 0 1 3 ... n-2

j = n-1 n-2 n-3 n-1-(n-2)

= 1

$\overbrace{n-1 + n-2 + n-3 + \dots + n-k}^{nk - k(k+1)}$

$$= nk - \frac{k(k+1)}{2}$$

$$= nk - \frac{(n-1)n}{2}$$

$$= n(n-1) - \frac{n(n-1)}{2}$$

$$= \frac{n(n-1)}{2}$$

SELECTION SORT :

- ① selects max/min of unsorted part and pushes it to the last/first index. This is done until all the elements are sorted.

Example: 5 9 2 15 6 11

① min = 2, swap 5 and 2

2 9 5 15 6 11

② min = 5, swap 9 and 5

2 5 9 15 6 11

③ min = 6, swap 9 and 6

2 5 6 15 9 11

④ min = 9, swap 15 and 9

2 5 6 9 15 11

⑤ min = 11, swap 15 and 11

2 5 6 9 11 15

P.T.O

```
void selectionSort(int a[], n)
```

{

```
int i, j, min;
```

```
for(i=0; i<n-1; i++)
```

{

```
min = i;
```

```
for(j=i+1; j<n; j++)
```

{

```
if(a[j] < a[min])
```

{

```
min = j;
```

{

}

```
if(i!=min)
```

{

```
temp = a[i];
```

```
a[i] = a[min];
```

```
a[min] = temp;
```

{

}

{

 $i = 0 \pm \dots k \dots \dots (n-2)$ $n-1 \ n-2$ $n-(0+1) \ n-(0+2) \ \dots \ n-(0+k)$ $\dots n-(0+n-2)$ $= 2$ $n-1 + n-2 + \dots + n-(n-2)$ $= (n-2)n - \frac{(n-2)(n-1)}{2}$ \downarrow
 $O(n^2)$

INSERTION SORT :

5	1	8	2	16
---	---	---	---	----

① pick 1: check $5 > 1$ (or) $1 < 5 \rightarrow$ sort.

1	5	8	2	16
---	---	---	---	----

② pick 8: check $5 > 8$ (or) $8 < 5 \rightarrow$ sort.

1	5	8	2	16
---	---	---	---	----

③ pick 2: check $8 > 2$ (or) $8 < 2 \rightarrow$ sort.

check $5 > 2$ (or) $2 > 5 \rightarrow$ sort.

check $1 > 2$ (or) $2 > 1 \rightarrow$ sort.

1	2	5	8	16
---	---	---	---	----

 \rightarrow sorted...

P.T.O
~~~~~

```
void insertionsort( int a[], n )
```

```

{
    int i,j,temp;
    for (i=1 ; i<n ; i++)
    {
        temp=a[i];
        j=i-1;
        while (j>=0 && a[j]>temp)
        {
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=temp;
    }
}

```

$\omega(n)$   
 $\downarrow$   
best case

## QUICK SORT

56, 28, 95, 18, 78, 33, 45, 57, 20

↓ after pass 1

33 28 20 18 45 56 78 57 95

↓ perform Quick sort for RHS and LHS

LAS :

—

33 28 20 18 45

$\downarrow$

$\downarrow$   
 $j=4$

PIVOT = 33.

$$a[i] = a[0] = \text{pivot} = 33. \Rightarrow 33 \leq \text{pivot}_{i+1}.$$

33 28 20 18 45  
↑ ↓  
i=1 j=4

$28 \leq \text{pivot}$      $i++$

33 28 20 18 45

1  
i = 2

$20 < \text{pivot } i++$

33 28 20 18 45 ✓ ↙ ↙  $j=4$

33 28 20 18 45

18 <= pivot i++;

```
void swap (int *a, int *b)
```

```
{
```

```
    int t = *a;
```

```
    *a = *b;
```

```
    *b = t;
```

```
}
```

```
int partition (int arr[], int low, int high)
```

```
{
```

```
    int pivot = arr[high];
```

```
    int i = low - 1;
```

```
    for (int j = low; j <= high - 1; j++)
```

```
{
```

```
    if (arr[j] < pivot)
```

```
{
```

```
        i++;
    
```

```
    swap(&arr[i], &arr[j]);

```

```
}
```

```
    swap(&arr[i], &arr[j]);

```

```
}
```

```
    swap(&arr[i+1], &arr[high]);

```

```
    return (i+1);

```

```
}
```

```
void quicksort (int arr[], int low, int high)
```

```
{
```

```
    if (low < high)

```

```
{
```

```
        int p = partition (arr, low, high);

```

```
        quicksort(arr, low, p-1);

```

```
        quicksort(arr, p+1, high);

```

```
}
```

```
int main()

```

```
{
```

```
    int n;

```

```
    Scanf ("%d", &n);

```

```
    int a[n];

```

```
    for (int i = 0; i < n; i++)

```

```
{
```

```
        Scanf ("%d", &a[i]);

```

```
}
```

```
    quicksort (arr, 0, n-1);

```

```
}
```

33 28 20 18 45

↓

i=4, j=4

$45 \leq \text{pivot} \rightarrow \text{false}$ .

$a[4] = 45 > \text{pivot} \rightarrow j--$

33 28 20 18 45

↓ ↓

j=3 i=4

$a[3] = 18 > \text{pivot} \rightarrow \text{false}$ .

$i < j \rightarrow \text{false}$ . so, swap pivot and  $a[j]$

18 28 20 33 45

→ pass 1 completed for LHS.

RHS :

78 57 95  
↓  
i=0              ↓  
                j=2

pivot = 78.

$78 \leq \text{pivot} \rightarrow \text{true}$ . i=1

78 57 95  
i=1              j=2

$57 \leq \text{pivot} \rightarrow \text{true}$     i=2

78 57 95  
i=2, j=2

$95 \leq \text{pivot} \rightarrow \text{false}$ .

$95 > \text{pivot} \rightarrow \text{true}$     j=1

78 57 95  
j=1              i=2

$57 > \text{pivot} \rightarrow \text{false}$ .

$i < j \rightarrow \text{false}$ .

so, swap pivot and 57.

57 78 95

→ pass 1 completed for RHS...

like this we do until its sorted - code → write later.

P.T.O.

## MERGE SORT

learn on your own ....

## HEAP SORT

binary tree → each node : 2 children at max (0, 1, 2)

arrange in ↑↑ order → maxHeap.

arrange in ↓↓ order → minHeap.

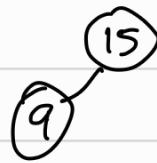
maxHeap :

9 15 16 1 5 30

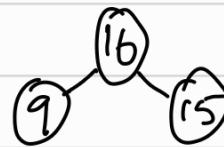
parent  $\geq$  children

(9) → parent.

Check:  $15 > 9 \rightarrow$  swap; (9) → child, (15) → parent



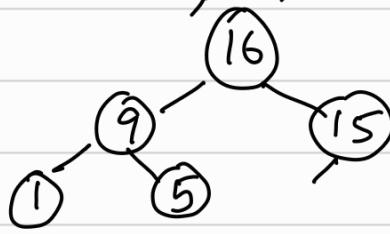
Check:  $16 > 15 \rightarrow$  swap; (16) → parent, (15) → child



(9) → parent.

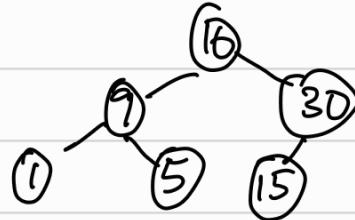
Check:  $9 > 1$ , no swap

Check:  $9 > 5$ , no swap.

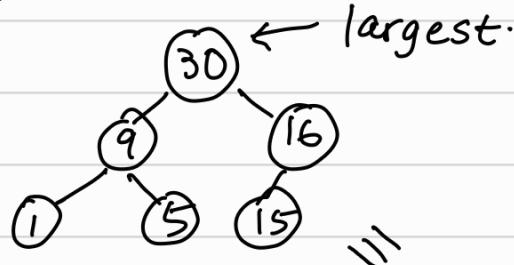


(15) → parent.

Check:  $30 > 15 \rightarrow$  Swap.



$30 > 16$ : Swap.



30 9 16 1 5 15



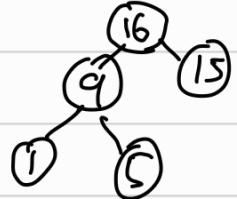
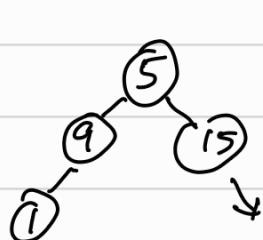
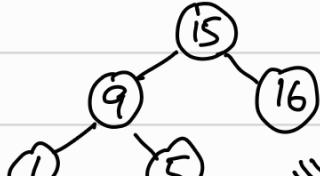
15 9 16 1 5 (30)

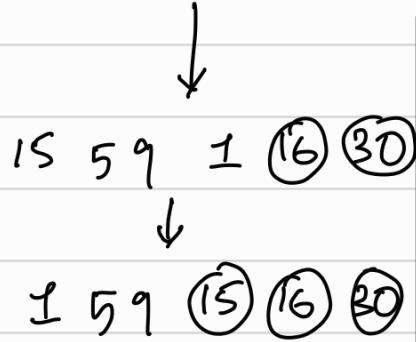


16 9 15 1 5

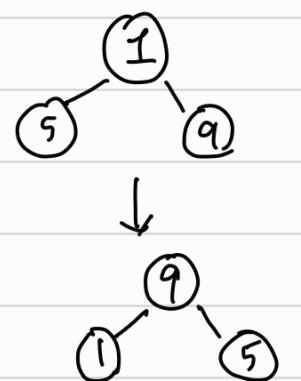
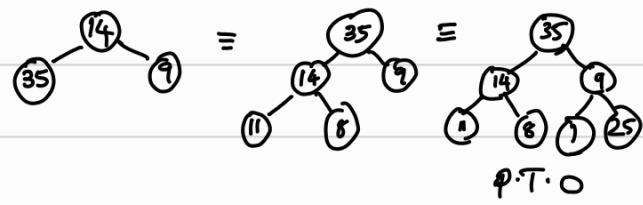


5 9 15 1 (16) (30) →

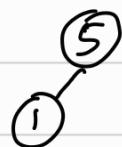
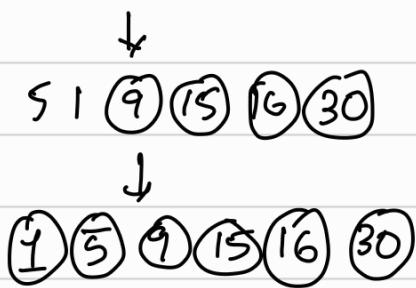




14, 35, 9, 11, 8, 1, 24, 50.



9 15 15 6 30



do maxHeap for: 16 1 5 8 13 12 → done.

for quickSort →

$$T(1) = 1$$

$$T(n) = 2 \times T\left(\frac{n}{2}\right) + n$$

$$T\left(\frac{n}{2}\right) = 2 \times T\left(\frac{n}{4}\right) + \frac{n}{2}$$

$$T\left(\frac{n}{4}\right) = 2 \times T\left(\frac{n}{8}\right) + \frac{n}{4}$$

$$\therefore T(n) = 4 \times T\left(\frac{n}{4}\right) + 2n$$

$$= 4 \times \left( 2 \times T\left(\frac{n}{8}\right) + \frac{n}{4} \right) + 2n$$

$$= 8 \times T\left(\frac{n}{8}\right) + 3n$$

⋮

$$= 2^k \times T\left(\frac{n}{2^k}\right) + nk$$

$$= 2^k \times T(1) + n \log n$$

$$= n + \underline{n \log n}$$

$\Omega(n \log n)$

$$\frac{n}{2^k} \geq 1$$

$$n \geq 2^k$$

$$\Rightarrow \underline{\log n \geq k}$$

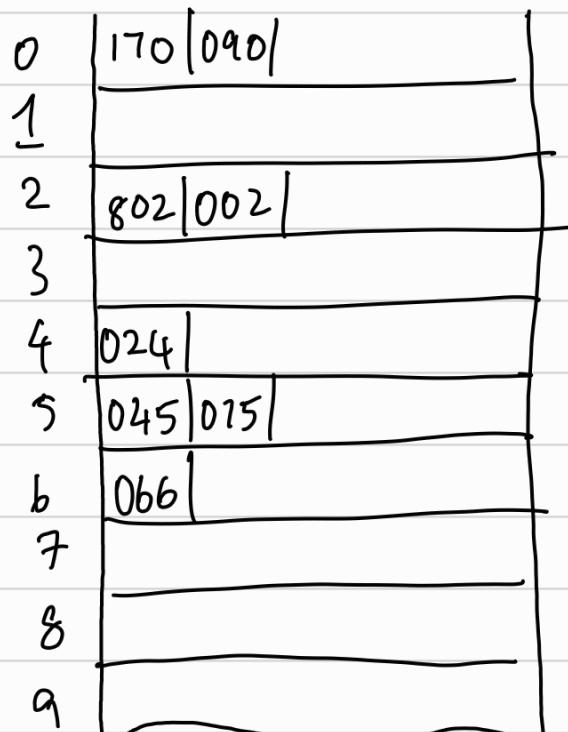
$O(n^2) \rightarrow$  if we need ascending, but given descending.

## RADIX SORT

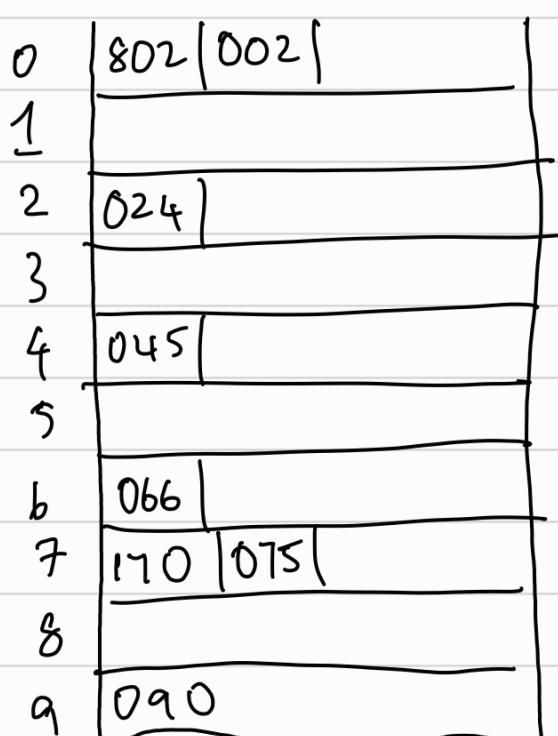
- no comparisons
- distribute elements into buckets based on individual digits
- processes digits - from LSD to MSD (or) vice versa.

170, 45, 75, 90, 802, 24, 2, 66

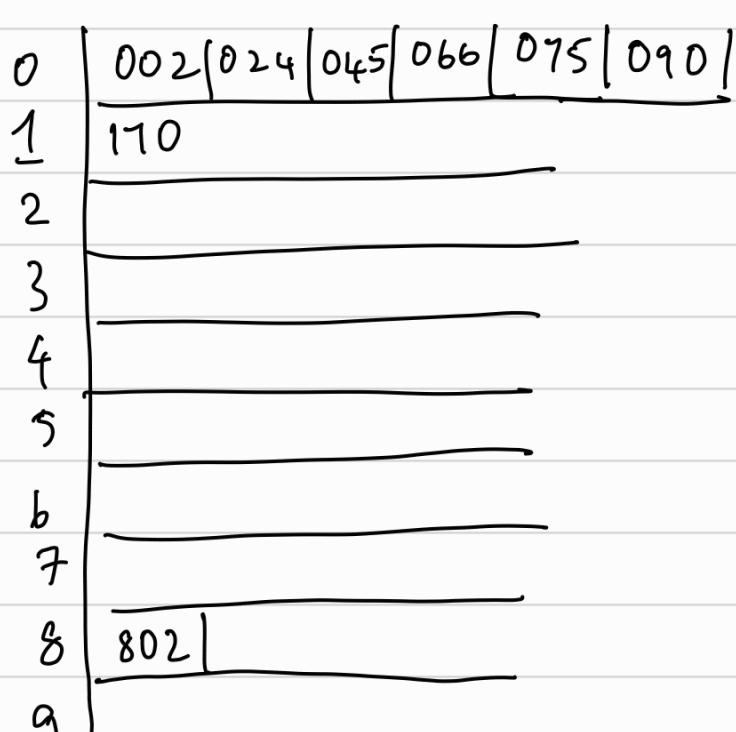
↓  
170, 045, 075, 090, 802, 024, 002, 066



170, 090, 802, 002, 024, 045, 075, 066



802, 002, 024, 045, 066, 170, 075  
090



002, 024, 045, 066, 075, 090,  
170, 802

## COUNTING SORT

- Non comparative - counts the  $n$ (occurrences) of each element in input.
  - works well if  $\Delta$  of range b/w elements is not that great.

2 9 7 4 1 8 4  
 $\downarrow$   
 largest element

Create arr of  $(9+1) = 10$ .

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

① 2  $\rightarrow$  go to 1st  $\nexists \uparrow\uparrow$  value by 1:

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

② 9  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

③ 7  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

④ 4  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

⑤ 1  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

⑥ 8  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

⑦ 4  $\rightarrow$ 

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 2 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|

  
 0 1 2 3 4 5 6 7 8 9

$\downarrow$        $\downarrow$        $\downarrow$

add  $\curvearrowleft$ 

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 2 | 4 | 4 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|

  
 cumulatively 0 1 2 3 4 5 6 7 8 9

Start from last 2 9 7 4 1 8 ④

④ check value at 4th index  $\rightarrow$  4  
subt. 1  $\rightarrow$   $4 - 1 = 3$ .

add 4 at 3rd index.

— — — 4 — — —

⑧  $\rightarrow$  at 8th  $\rightarrow$  6  $\rightarrow$   $6 - 1 = 5$   
add 8 at 5th index

— — — 4 — 8 —

①  $\rightarrow$  at 1  $\rightarrow$  2  $\rightarrow$   $2 - 1 = 1$   
add 2 at 1.

— 2 — 4 — 8 —

④  $\rightarrow$  at 4  $\rightarrow$  4  $\rightarrow$   $4 - 1 = 3$   
add 4 at 3

— 2 4 4 — 8 —

⑦  $\rightarrow$  at 7, 5  $\rightarrow$   $5 - 1 = 4$ .  
add 7 at 4

— 2 4 4 7 8 —

⑨  $\rightarrow$  at 9, 7  $\rightarrow$   $7 - 1 = 6$   
add 9 at 6

— 2 4 4 7 8 9 —

②  $\rightarrow$  at 1, 1  $\rightarrow$   $1 - 1 = 0$   
add 1 at 1

1 2 4 4 7 8 9 //

P.T.O

## INPLACE SORTING ALGORITHMS

- doesn't require extra space.
- sorts within its own memory.
- Bubble, Insertion, Selection, Heap, Quick → inplace

## STABLE SORTING ALGORITHMS

- relative order of equal elements in the sorted output follow their relative order in the unsorted input.  
→ merge, insertion, bubble, selection, counting, radix.

## TIME COMPLEXITY of Algorithms

- Bubble → best:  $O(n)$   
worst:  $O(n^2)$   
avg:  $O(n^2)$
- Selection → best:  $O(n^2)$   
worst:  $O(n^2)$   
avg:  $O(n^2)$
- Insertion → best:  $O(n)$   
worst:  $O(n^2)$   
avg:  $O(n^2)$
- Merge & Heap → best:  $O(n \log n)$   
worst:  $O(n \log n)$   
avg:  $O(n \log n)$
- Quick → best:  $O(n \log n)$   
worst:  $O(n^2)$   
avg:  $O(n \log n)$
- Counting → best:  $O(n+k)$ ;  $k \rightarrow$  range of input of non-negative key values.
- Radix → best:  $O(n \cdot k)$   
worst:  $O(n \cdot k)$ ;  $k \rightarrow n$  (digits) in max no.

P.T.O

## MASTER'S THEOREM

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k(\log n)^p)$$

$a >= 1$ ,  $b > 1$ ,  $k >= 0$ ,  $p$  is Real Number.

1. if  $a > b^k$ , then  $T(n) = \Theta(n^{\log_b a})$

2. if  $a = b^k$ ,

(a) if  $p > -1$ , then  $T(n) = \Theta(n^{\log_b a} (\log n)^{p+1})$

(b) if  $p = -1$ , then  $T(n) = \Theta(n^{\log_b a} \log(\log n))$

(c) if  $p < -1$ , then  $T(n) = \Theta(n^{\log_b a})$

3. if  $a < b^k$ ,

(a) if  $p >= 0$ , then  $T(n) = \Theta(n^k(\log n)^p)$ .

(b) if  $p < 0$ , then  $T(n) = \Theta(n^k)$ .

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad ; a = 2, b = 2, k = 1, p = 0$$

$$b^k = 2^1 = 2.$$

$$\Rightarrow \boxed{a = b^k}$$

$$\begin{aligned} p = 0 \Rightarrow p > -1 \Rightarrow T(n) &= \Theta\left(n^{\log_b a} (\log n)^{p+1}\right) \\ &= \Theta\left(n^{\log_2 2} (\log n)^1\right) \\ &= \Theta(n \log n) \end{aligned}$$

$$\Rightarrow \boxed{T(n) = \Theta(n \log n)}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

$$aT\left(\frac{n}{b}\right) + \Theta(n^k(\log n)^p)$$

$$a = 2, b = 2, k = 1, p = 1.$$

$$a = b^k \text{ and } p > -1.$$

$$\begin{aligned} \Rightarrow T(n) &= \Theta\left(n^{\log_b a} (\log n)^{p+1}\right) \\ &= \Theta\left(n^{\log_2 2} (\log n)^2\right) \\ &= \Theta(n(\log n)^2) \\ \therefore \boxed{T(n) = \Theta(n(\log n)^2)} \end{aligned}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n/\log n)$$

$$a = 2, b = 2, k = 1, p = -1.$$

$$a = b^k \text{ and } p = -1.$$

$$\begin{aligned} & \Theta(n^{\log_b a} \cdot \log(\log n)) \\ &= \Theta(n^{\log_2 2} \cdot \log(\log n)) \\ &= \Theta(n \log(\log n)) \\ \therefore T(n) &= \Theta(n \log(\log n)) \end{aligned}$$

$$T(n) = 3T(n/2) + n^2$$

$$a=3, b=2, k=2, p=0$$

$$\begin{aligned} a < b^k \quad (3 < 4) \quad \text{and } p \geq 0 \\ \Rightarrow T(n) &= \Theta(n^k (\log n)^p) \\ &= \Theta(n^2) \\ \Rightarrow T(n) &= \Theta(n^2) \end{aligned}$$

$$T(n) = 2T(n/2) + n^{0.51}$$

$$a=2, b=2, k=0.51, p=0$$

$$a > b^k \text{ and } p=0$$

$$\begin{aligned} \Rightarrow T(n) &= \Theta(n^{\log_b a}) \\ &= \Theta(n^{\log_2 2}) \\ &= \Theta(n) \\ \Rightarrow T(n) &= \Theta(n) \end{aligned}$$

$$T(n) = 0.5T(n/2) + 1 \quad ; \quad \boxed{\text{let } T(1) = 1}$$

$$a=0.5, b=2, k=-1, p=0 \rightarrow \text{NO MASTER'S THEOREM} \\ (a >= 1, \text{ here its } 0.5)$$

$$T(n) = 3T(n/2) + (\log n)^2$$

$$a=3, b=2, k=0, p=2$$

$$a > b^k \quad (3 > 2^0)$$

$$\Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

P.T.O.

$$T(n) = 2T\left(\frac{n}{2}\right) + n(\log n)^2$$

$$a=2, b=2, k=1, p=2$$

$$a=b^k, p>-1 \Rightarrow T(n) = \Theta\left(n^{\log_b a} \cdot (\log n)^{k+1}\right)$$

$$= \Theta\left(n^{\log_2 2} \cdot (\log n)^3\right)$$

$$= \Theta(n(\log n)^3)$$

$$T(n) = 16T\left(\frac{n}{4}\right) + n$$

$$a=16, b=4, k=1, p=0$$

$$16 > 4 \quad (a > b^k) \Rightarrow T(n) = \Theta(n^{\log_b a})$$

$$= \Theta(n^{\log_4 16})$$

$$= \Theta(n^2)$$

$T(n) = 2^n T\left(\frac{n}{2}\right) + n^n \rightarrow \text{CAN'T APPLY MASTER'S THEOREM!}$   
 (use back substitution)

LITTLE OH:

if  $f(n) \leq c \cdot g(n)$   
 then  $f(n) = O(g(n))$

$$f(n) \leq c \cdot g(n); c > 0$$

$$f(n) = n \text{ and } g(n) = n^2$$

if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ , then  $f(n) = o(g(n)) \rightarrow \text{Little Oh.}$

$$\lim_{n \rightarrow \infty} \frac{n}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0 \therefore f(n) = o(g(n))$$

find ' $c$ '. s.t  $f(n) \leq c \cdot g(n)$  ALWAYS!

$$f(n) = \log n, g(n) = n.$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\log n}{n} = 0$$

$$f(n) = o(g(n))$$

$$\log n = o(n)$$

LITTLE OMEGA:

↳ Lower bound, not tight

↳ if  $f(n) = \omega(g(n))$ , then:  $|f(n)| > c|g(n)|$

↳  $f(n)$  grows much faster than  $g(n)$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) \sim \omega(g(n))$$

$$f(n) = n ; g(n) = \log n \quad \frac{f(n)}{g(n)} = \frac{n}{\log n} \equiv \frac{1}{\frac{1}{n}} = n.$$

$$n \rightarrow \infty \Rightarrow \frac{f(n)}{g(n)} = \infty$$

$$\therefore f(n) = \omega(g(n))$$

$$f(n) = 3^n, g(n) = 2^n$$

$$\frac{f(n)}{g(n)} = \frac{3^n}{2^n} = \left(\frac{3}{2}\right)^n \quad \text{as } n \rightarrow \infty, \left(\frac{3}{2}\right)^n = \infty$$

$$\therefore f(n) = \omega(g(n))$$

## SPACE COMPLEXITY :

- Measure of amount of storage space an algorithm / program uses to solve a problem as a fn. of size of the input.
- expressed in "Big Oh" notation.

printf(" %d", a+b); (or)  $s = a+b \rightarrow \text{printf}("%d", s)$  ;

$$SP = C + S_p$$

↓      ↴ dependent  
independent

$$O(2) \equiv O(1)$$

int n, a[n];

Sum=0

for(int i=0; i<n; i++)

{

  sum+=a[i];

}

$$SP = C + S_p$$

$$SP = 3 + n \rightarrow \{n \text{ integers}\}$$

$$= O(n+3)$$

$$= O(n) //$$

another defn  $\rightarrow SP \equiv$  extra memory storage used.

## for Sorting :

- bubble :  $O(1)$
- selection :  $O(1)$
- insertion :  $O(1)$
- merge :  $O(n)$
- quick :  $O(n)$
- heap :  $O(1)$
- counting :  $O(k)$
- radix :  $O(n+k)$

## SEARCHING ALGORITHMS :-

### Linear Search :

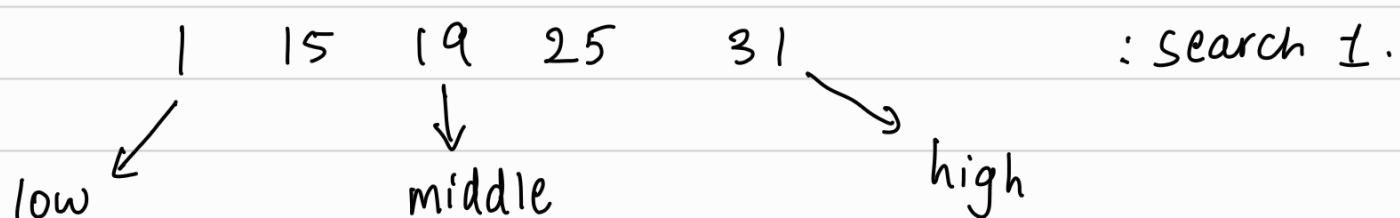
→ sequential search. Checks until match is found.

```
int linearSearch (int a[], int target, int n)
{
    for (int i=0; i<n; i++)
    {
        if (a[i]==target)
        {
            return i; // found, return index
        }
    }
    return -1; // not found error 404 :(
}
```

### Binary Search

↳ for sorted arrays

↳ divides search in half; narrows down possible locations for the target value.



$$\text{low} = 0 ; \text{high} = 4$$

$$\text{middle} = \left( \frac{\text{low} + \text{high}}{2} \right) = \frac{0+4}{2} = 2$$

Compare  $a[m] == \text{target}$  ( $19 \neq 1$ )  $\rightarrow$  false.

since false, we narrow search to LHS of m as  
target < middle

$$\text{low} = 0, \text{high} = \text{middle} - 1 = 2 - 1 = 1$$

$$\text{middle} = \frac{(0+1)}{2} = 1/2 = 0$$

compare  $a[\text{middle}] == \text{target} \rightarrow \text{TRUE!}$

so, found. THIS IS BINARY SEARCH.

(if  $a[\text{middle}] < \text{target}$ ,  $m = m + 1$  and high doesn't change).

int binarySearch (int a[], int low, int high, int target)

{

    while (low <= high)

{

        int mid = (low + high) / 2;

        if (a[mid] == target)

{

            return mid;

}

        else if (a[mid] < target)

{

            low = mid + 1;

}

    else

{

        high = mid - 1;

}

    return -1;

}

for binary search:  $T(n) = T(n/2) + 1$   
(1)

$$T(n) = aT(n/b) + \Theta(n^k(\log n)^p)$$

$$a = 1, k = 0, p = 0$$

$$b = 2$$

$$p > -1$$

$$T(n) = n^{\log_b a} (\log n)^{p+1}$$

$$T(n) = \log n$$

