Quiz#1 : 6th may

Stack, Queue, Linked List

CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Outline

- Stack Application: Recursion
- Linked list data structures operations
  - insert_begin ()
  - insert_end ()
  - Insert_pos ()
  - Traversal ()

Stack Application: Recursion

- Base case
- Sub task
- Recursive case

- *Recursion:* (i) Any function which calls itself is called *recursive.*
  (ii) Recursion terminates → *we need to make sure*
  (iii) The small-small recursive functions should be convergence
  (iv) The code is shorter

```
//Calculate the factorial of a positive integer
int Fact(int n){
    if (n == 1) // base case: fact of 0 or 1
        return 1;
    else if (n == 0)
        return 1;
    else  //recursive case: multiply n by (n-
1) factorial
        return n*Fact(n-1);
}
```

n = 1

3 * Fact(3-1)

4 * Fact(4-1)

3	1*Fact(1-1)
2	2*Fact(2-1)
1	3*Fact(3-1)
0	4*Fact(4-1)

= 24

Stack Application: Recursion Exercise

What is the output of the below program?

```
int f(int n)
{ static int r=0;
if(n<=0) return 1;
if(n>3)
{ r=n;
  return f(n-2)+2;
}
return f(n-1)+r;
}
```

What is the value of f(4)

a) 6 b) 7 c) 11 d) 9

f(4) ① ②

Case#: f(4)

$r = 4$

$f(4-2)+2$

11

$f(2)$ 9

$f(2-1)+4$

③ $\boxed{f(1-1)}+4$

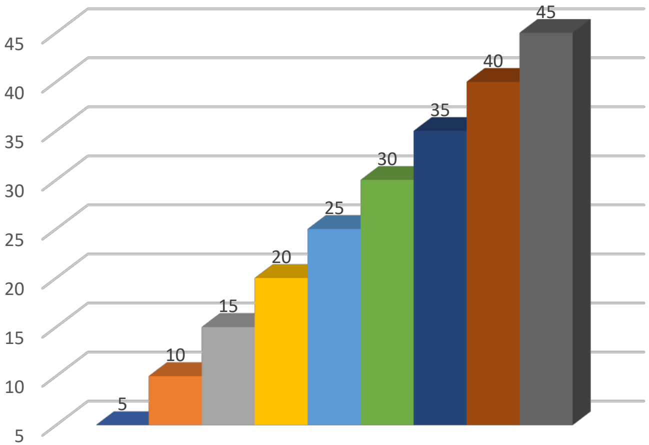
1

$= 5$

Recursion: Example Algorithms

- Quick Sort, Merge Sort
- Divide and Conquere Algorithm
- Tower of Hanoi
- Binary Search
- Tree Traversal
- Graph Traversal

Sorting (1)



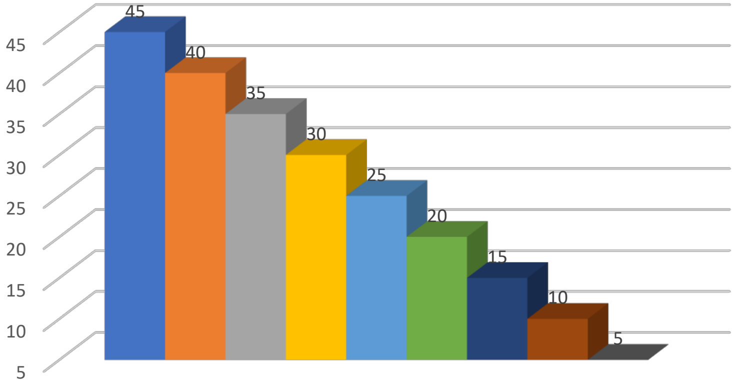
A									
A	10	15	5	25	20	35	30	45	40

A_{ase}									
A_{ase}	5	10	15	20	25	30	35	40	45

A_{des}									
A_{des}	45	40	35	30	25	20	15	10	5

Ascending: smallest → largest

Descending: largest → smallest



sort

verb

UK /sɔːt/ US /sɔːrt/

sort *verb* (ORDER)

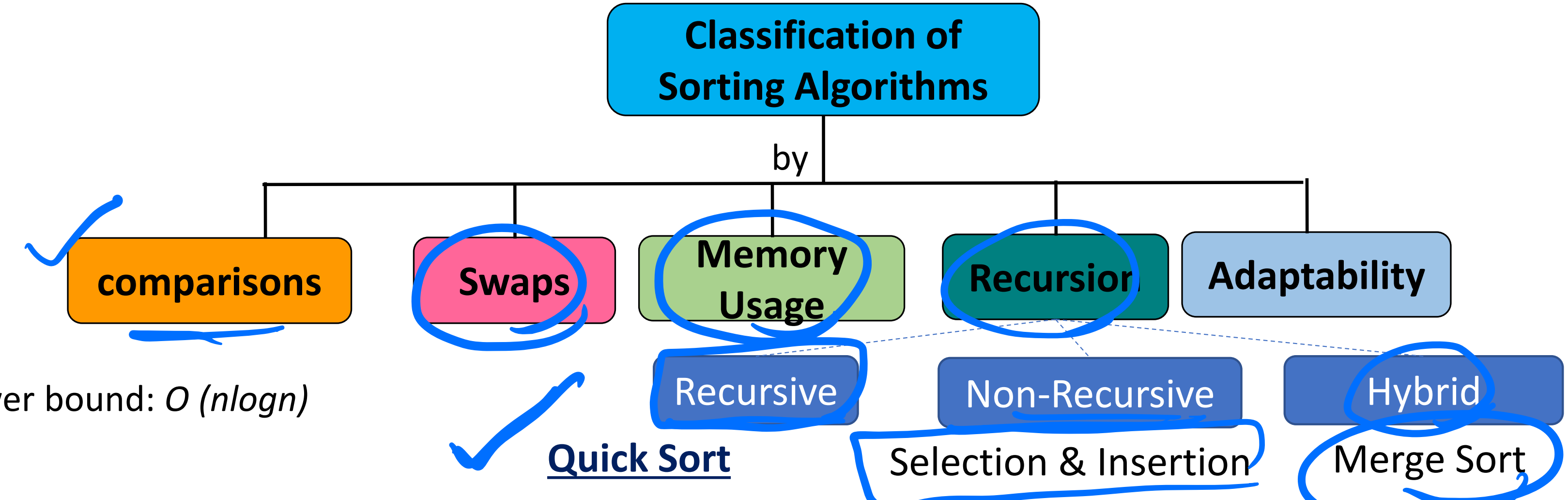
B2 [I or T]

to put a number of things in an order or to separate them into groups:

Sorting (2)

- *What is Sorting?*
  - *Sorting refers to re*arranging the elements of a list in a certain order [either ascending or descending]
- *Why is Sorting necessary or so important?*
  - Sorting is one of important principles of algorithm design.
    - – Sorting helps to reduce the complexity of the problem.
    - – Sorting is used as a technique to reduce the search complexity (e.g., binary search)

➢ ***Classification of Sorting Algorithms!***



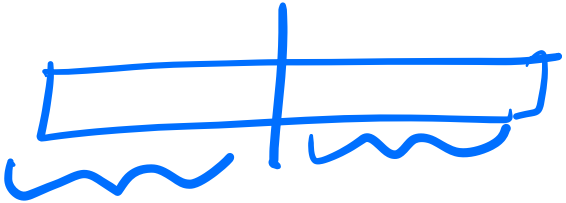
Internal Sort
External Sort

Assignment#2: Quick Sort

- Objective: Implement <u>Quick sort</u> to sort the integers in the input file in ascending order
- Inputs: Command-line argument, *the **input.txt** file*
- Output: A file (e.g. ***quicksort.txt***)
  - What the output file should contain?
    - The output file should contain sorted [illegible] integers with ascending order (the first line of file should contain the smallest integer)

Credit to: Prof. Sandeep R B, IIT Dharwad, India

Divide-and-Conquer



- Design Principle: Quick sort follows the divide-and-Conquer design approach

- Divide-and-Conquer: Command-line argument, *the* ***input.txt*** *file*

  - Divide: if the problem input size is too large → divide the problem into two or more smaller instances (i.e., sub-problems)

  - Conquer: the sub-problems are solved recursively by following divide-and-conquer approach again

  - Combine: Combine the results of all solutions to all sub-problems to get the final solution

Quick Sort Algorithm

- Recursive steps:

  (i) if there are one or no elements in the list to be sorted, *return*;

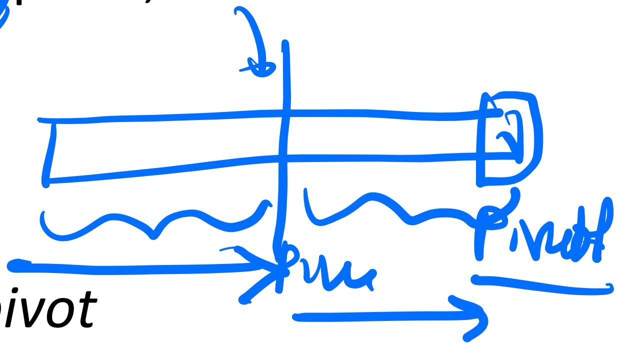
  (ii) Pick an element in the list to serve as the *pivot* point;

  (iii) Split the list into two parts:

  ✓ one with elements larger than the *pivot*

  ✓ the other with elements smaller than the *pivot*

  (iv) Recursively repeat the algorithm for both halves of the original list



Quick Sort Algorithm (2)

- **Algorithm:**

```
QUICKSORT (A, l, r)
    if (l < r)
        q = PARTITION (A, l, r);
        QUICKSORT(A, l, q-1);
        QUICKSORT(A, q + 1, r)
```

```
PARTITION (A, l, r)
    x = A[r] //pivot
    i = l-1
    for j = l to r-1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1
```

7

3<4

i=0 j=1

i=1

Step 2: PARTITION (A, 1, 8)

l=1 r=8

	1	2	3	4	5	6	7	8
	2	8	7	1	3	5	6	4
i = 2; j = 5	2	1	7	8	3	5	6	4
i = 3; Exchange A[3] with A[5]	2	1	3	8	7	5	6	4
i = 3; j = 6	2	1	3	8	7	5	6	4
i = 3; j = 7	2	1	3	8	7	5	6	4
Exchange A[4] with A[8]	2	1	3	4	7	5	6	8
return i+1	2	1	3	4 (q)	7	5	6	8

Pivot

QUICKSORT(A, 1, 3); QUICKSORT(A, 5, 8);

Quick Sort Algorithm (3)

- **Algorithm:**

```
QUICKSORT (A, l, r)
  if (l < r)
     q = PARTITION (A, l, r);
     QUICKSORT(A, l, q-1);
     QUICKSORT(A, q + 1, r);
```

Step 2: PARTITION (A, 1, 8)

1	2	3	4	5	6	7	8
2	**1**	**3**	**4**	**7**	**5**	**6**	**8**

q

QUICKSORT(A, 1, 3);   QUICKSORT(A, 5, 8);

```
PARTITION (A, l, r)
 x = A[r] //pivot
 i = l-1
 for j = l to r-1
     if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
 exchange A[i+1] with A[r]
 return i+1
```

50 25 30 15 60

left

left right

if

if (left < right)

15 25 30 50 60

Linked List: Operations

Linked List Data Structures

- **Create**
- **Insert**
- **Delete**
- **IsListEmpty**
- **Traversal**

```
*create();
```

Insert
- at beginning
- at end
- at the given position

```
insert_begin();
insert_end();
insert_pos();
```

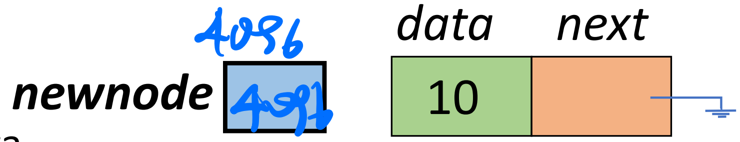
Delete
- at beginning
- at end
- at the given position

```
delete_begin();
delete_end();
delete_pos();
```

Traversal
- at beginning
- at end
- at the given position

```
display();
get_begin();
get_end();
get_pos();
```

Linked List: Insert at the beginning or Insert at the head



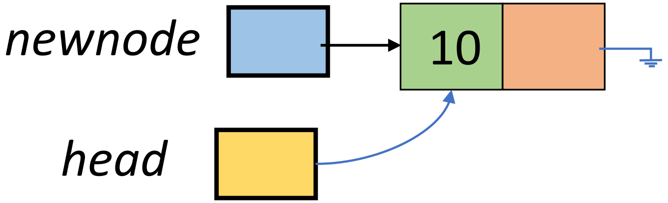

```
struct node {
        int data;
        struct node *next;
}
```

Steps:

(i) Creating a node with data

*struct node *newnode = malloc(sizeof(struct node));*
newnode → data = 10;
newnode → next = NULL;

(ii) Adding a node to an empty linked list



if (head == NULL)
head = newnode

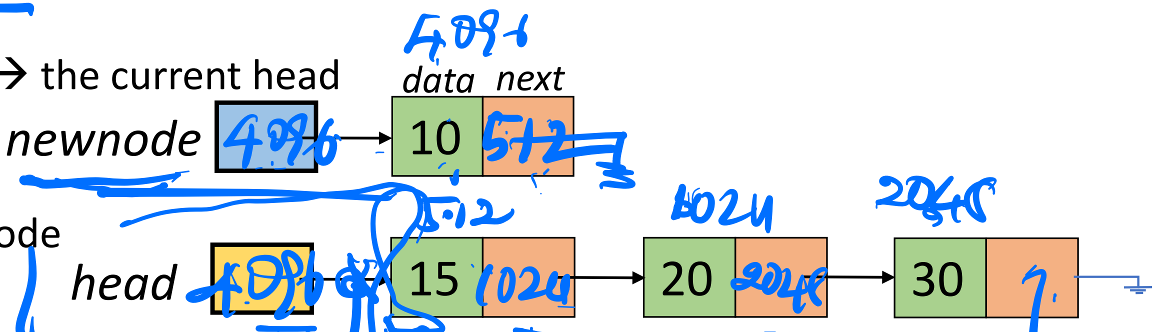
(iii) Adding a node to the beginning of a linked list

a) Update the next pointer of new node → the current head

newnode → next = head

b) Update the head pointer to the new node

head = newnode



Linked List: traversal or display

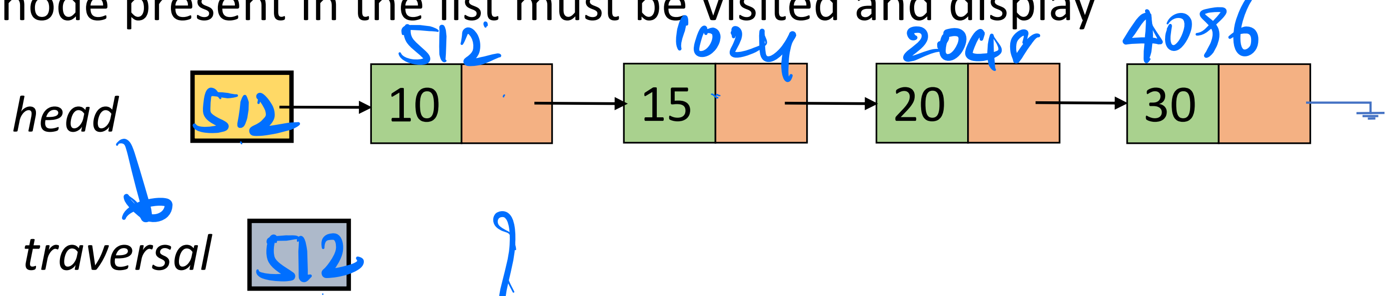
Steps:

(i) Check if the linked list empty or not

head *struct node *head = NULL*

if (head == NULL)
printf ("Linked List is Empty\n");

(ii) List Traversal: Each node present in the list must be visited and display the data value

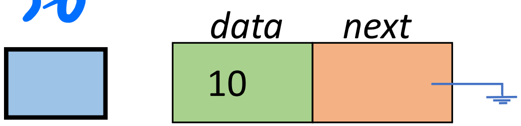


1. *struct node *traversal*
2. *traversal = head;*
3. *while (traversal != NULL)*
   display the element: traversal → data
   traversal = traversal → next

printf("%d", traversal→data

10, 15, 20, 30

Linked List: Insert at the end or Insert at the tail

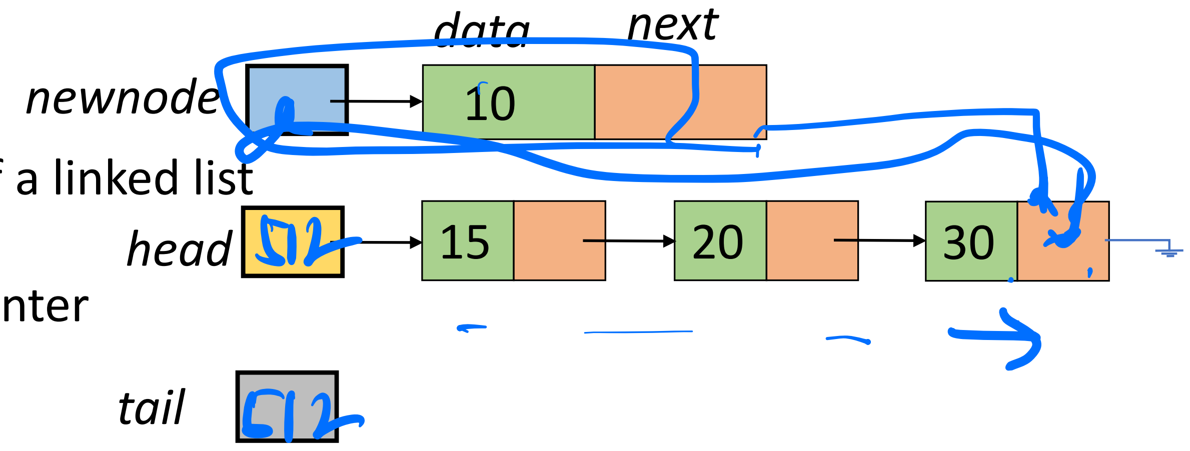


Steps:

(i) Creating a node with data

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

*struct node *newnode = malloc(sizeof(struct node));*
newnode → data = 10;
newnode → next = NULL;



(ii) Adding a node to at the end of a linked list

tail = head;

a) Traversal the list till the tail pointer

*struct node *tail*

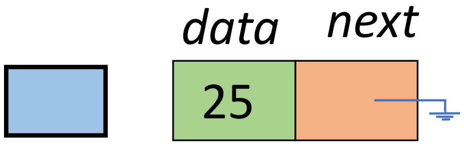
while (tail → next != NULL)
tail = tail → next

b) tail node pointer points to the new node *tail → next = newnode*

Linked List: Insert at the given position

***Steps*:**

(i) Creating a node with data



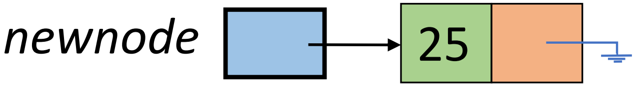
*struct node *newnode = malloc(sizeof(struct node));*

newnode → data = 10;

newnode → next = NULL;

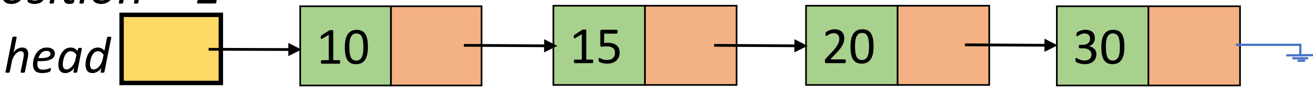
struct node {
int data;
struct node *next;
}

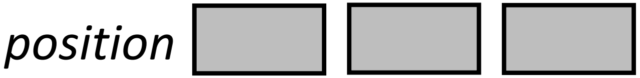
(ii) Adding a node to at the given position



a) Traversal the list till the *position – 1*

*struct node *position*
position = head
i = 0
while (i<pos)
position = position → next
i++;





b) Point *newnode* → next to the position-node → next

newnode → next = position → next

c) Point position-node → next to the *newnode*

position → next = newnode

Exercise: Linked List

Which of the following points is/are true about Linked List data structure when it is compared with array?

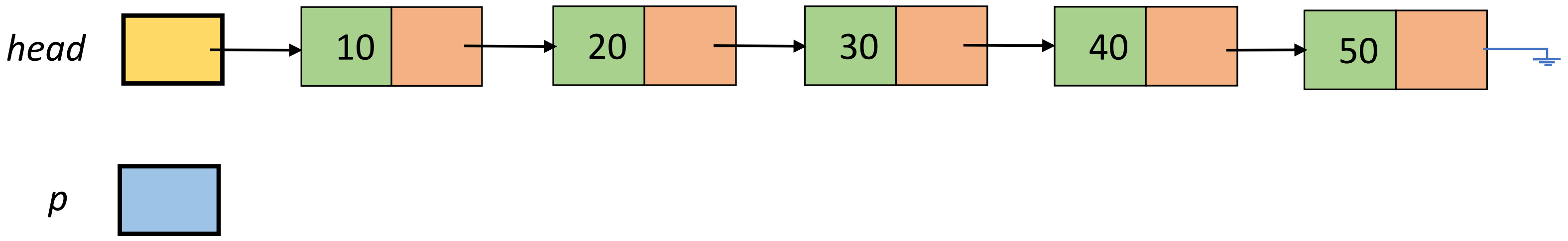
a) It is easy to insert and delete elements in Linked List

b) Random access is not allowed in a typical implementation of Linked Lists

c) The size of array has to be pre-decided, linked lists can change their size any time

d) All of the above

Exercise: Singly Linked List (2)



*struct node *p;*

p=head;

(i) p=head → next → next ;

(ii) p → next → next = head;

(iii) print("%d", p → next → next → data);

What is the output if the above statements are executed in the same sequence

a) 10 b) 20 c) 40 d)50

thank you!

email:
k.kondepu@iitdh.ac.in

NEXT Class: 24/04/2023