GOVERNMENT POLYTECHNIC

GULZARBAGH, PATNA - 800 007

BRANCH : CSE

NAME : Harsh Deep

CLASS ROLL No.: 16/CSE/21

BOARD ROLL No. : 411181821016 **GROUP** .....

SESSION : 2021-2024 **YEAR/SEMESTER** 4th

SUBJECT : Data structure & Algo. using C (LAB)

~2018107

14/08/2023

Professor's Signature

Name	Harsh Deep	Year	2021-24
Subject	Data Structure & Algo. using C (LAB) - 2018407	Class	
Semester	Fourth (IV)	Roll No.	46/CSE/21

INDEX

Sr. No.	Experiment Description	~~Experiment Date~~ Page no.	Submission Date	Remarks / Signature
01.	Develop a 'C' program to create and implement BINARY SEARCHING.	01 - 04		
02.	Develop a 'C' program to create and implement BUBBLE SORTING	05 - 10		
03.	Develop a 'C' program to create and implement MERGE SORTING on two sorted list.	11 - 17		Asp 14/08/2023
04.	Develop a 'C' program to create and implement a STACK using arrays.	18 - 24		
05.	Develop a 'C' program to create and implement a queue using array	25 - 28		

• Experiment No - 01 •

• **Aim** : Develop a 'C' program to create and implement BINARY SEARCHING.

• **Theory** :

* What is Searching ?

→ Searching is a method to find an element from data structure with there appropriate location.

* Types of Searching

→ There are two types of Searching :

1) Linear Search
2) Binary Search

* Binary Search

→ Binary search is a searching algorithm for finding an element position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array.

Example :

Array of 7 digits

2	4	6	8	10	12	14

Find : 10

left = l right = r

0	1	2	3	4	5	6
2	4	6	8	10	12	14

First iteration

Mid

Array [mid] = $\frac{l+r}{2}$

4	5	6
10	12	14

2nd Iteration

8 < 10, then 10 is found by index 4.

- 'C' Program for Binary Search

→

```
# include <stdio.h>
  int main ()
{
  int i, low, high, mid, n, key, a[100];
  printf ("Enter the number of Elements in array: ");
  scanf (" %d ", &n);

  for (i=0; i<n; i++)
  {
    printf ("array [%d] : ", i);
    scanf (" %d ", & a [i]);
  }
```

```
printf ("Enter value to find :");
scanf ("%d", &key);
low = 0 ;
high = n-1 ;
mid = (low + high) / 2 ;

while (low <= high)
{
  if (a[mid] < key)
    low = mid + 1 ;
  else if (a [mid] == key)
    {
      printf ("%d found at position [%d]",
               key, mid);
      break ;
    }
  else {
  high = mid - 1 ;
    mid = ((low + high)/2 ;
    }
  if (low > high)
    {
      printf (" Element not found");
      return
    }
  return 0;
}
```

* Output :

→ Enter the number of Elements in array : 5
array [0] : 2
array [1] : 4
array [2] : 6
array [3] : 8
array [4] : 10

Enter value to find : 10
10 found at index [4]

* Time Complexity

→ Best case : $O(1)$
Average case : $O(\log n)$
Worst Case : $O(\log n)$

* Space Complexity

→ The space complexity of the binary search is $O(1)$.

—— × ——

• Experiment No - 02 •

• **Aim :** Develop a 'C' program to create and implement BUBBLE SORTING.

• **Theory**

* What is Sorting ?

→ Sorting is a process of arranging elements either in ascending or descending order. By default, we sort numbers in an ascending order.

Example :

Unsorted input

23	78	3	2	1	100

Sorted input

1	2	3	23	78	100

* Bubble Sorting

→ Bubble is a simple sorting algorithm. This sorting is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are

not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n are number of items.

- **Concept**

* How can we sort an array?

→ In Bubble sort, we take the simplest possible approach to sort an array.

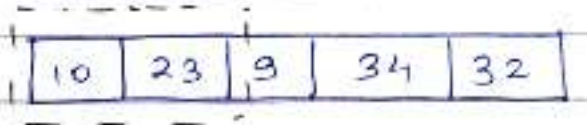
⇒ We look through the array in an orderly fashion, compairing only adjacent element at a time.

⇒ Whenever we see two elements which are out of order, we swaped them so that the smaller element comes before the greater element.

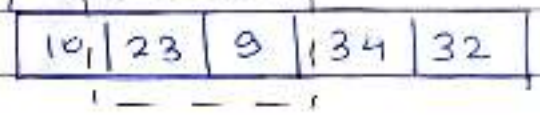
⇒ We keep performing the above steps over the array again and again till we get the sorted form.

* Step by Step process for one Iteration.

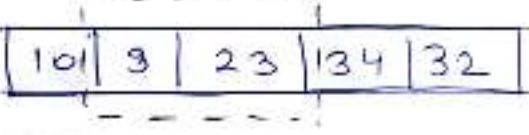
1) Comparing 10 and 23
10<23, no swap required



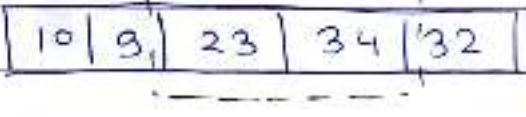
2) Comparing 23 and 19
23>9, Swap Required



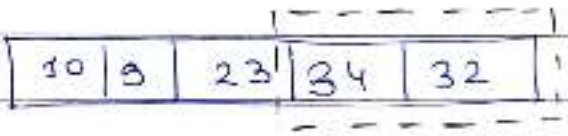
3) Swaping 9 and 23
as required



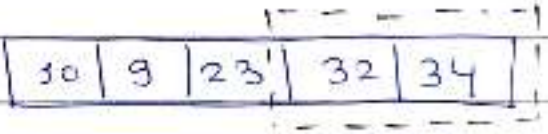
4) Comparing 23 and 24
23<24, no swap required



5) Comparing, 34 and 32
34>32, Swap required



6) Swaping 34 and 32
as required



* Algorithm of Bubble sort

Step 1 : Compare the i^{th} and $(i+1)^{th}$ element, where i = first index to i+1 = second last index.

Step 2 : Compare the pair of adjacent element if i^{th} element is greater than the $(i+1)^{th}$ element, swap them.

Step 3 : Run steps 1st and 2nd a total of $(n-1)$ times to attain the final sorted array.

- **C program for Bubble Sort**

→

```
#include <Stdio.h>
int main ()
{
    int a[100], num, x, y, temp;
    printf ("Please Enter the number of element
             in array : ");
    scanf ("%d", &num);
    printf ("Please Enter the value of element :");

    for (x=0 ; x<num ; x++)
    {
        scanf ("%d", &a[x]);
    }

    for (x=0 ; x<num-1 ; x++)
    {
        for (y=0 ; y<num-x-1 ; y++)
        {
            if (a[y] > a[y+1])
            {
                temp = a[y];
                a[y] = a[y+1];
                a[y+1] = temp;
            }
```

```
        }
    }
printf("Array after implementing bubble sort:");
    for (x=0; x<num; x++)
    {
        printf("%d", a[x]);
    }
    return 0;
}
```

- **Output**

→ Please Enter the number of Element in array : 5
Please Enter the value of Element : 9 5 2 7 3
Array after implementing bubble sort : 2 3 5 7 9

- **Time Complexity of Bubble sort :)**

→ The complexity of sorting algorithm depends upon the number of comparison that are made. Total comparisons in bubble sort is : $n(n-1)/2 \approx n^2 - n$

⇒ Best Case : $O(n)$
⇒ Average Case : $O(n^2)$
⇒ Worst Case : $O(n^2)$

- **Space complexity of Bubble sort**

→ Space complexity is O(1) because an extra variable is used for swapping.

→ In the optimized bubble sort algorithm, two extra variables are used. Hence, the space complexity will be O(2).

- **Applications of Bubble sort**

→ Bubble sort is used if

⇒ Complexity does not matter
⇒ Slow execution speed does not matter
⇒ Short and easy to understand coding is perform.

——— x ———

• Experiment no - 03 •

• **Aim** : Develop a 'C' program to create and implement MERGE SORTING on two sorted list.

• **Theory**

* What is merge sort ?

→ Merge sort is one of the most popular sorting algorithm that is based on the principal of divide and conquer algorithm.

* **Divide and conquer algorithm**

→ Divide and conquer is a stategy based on the idea that a given hard problem can be solved by breaking it down into smaller sub-problems.

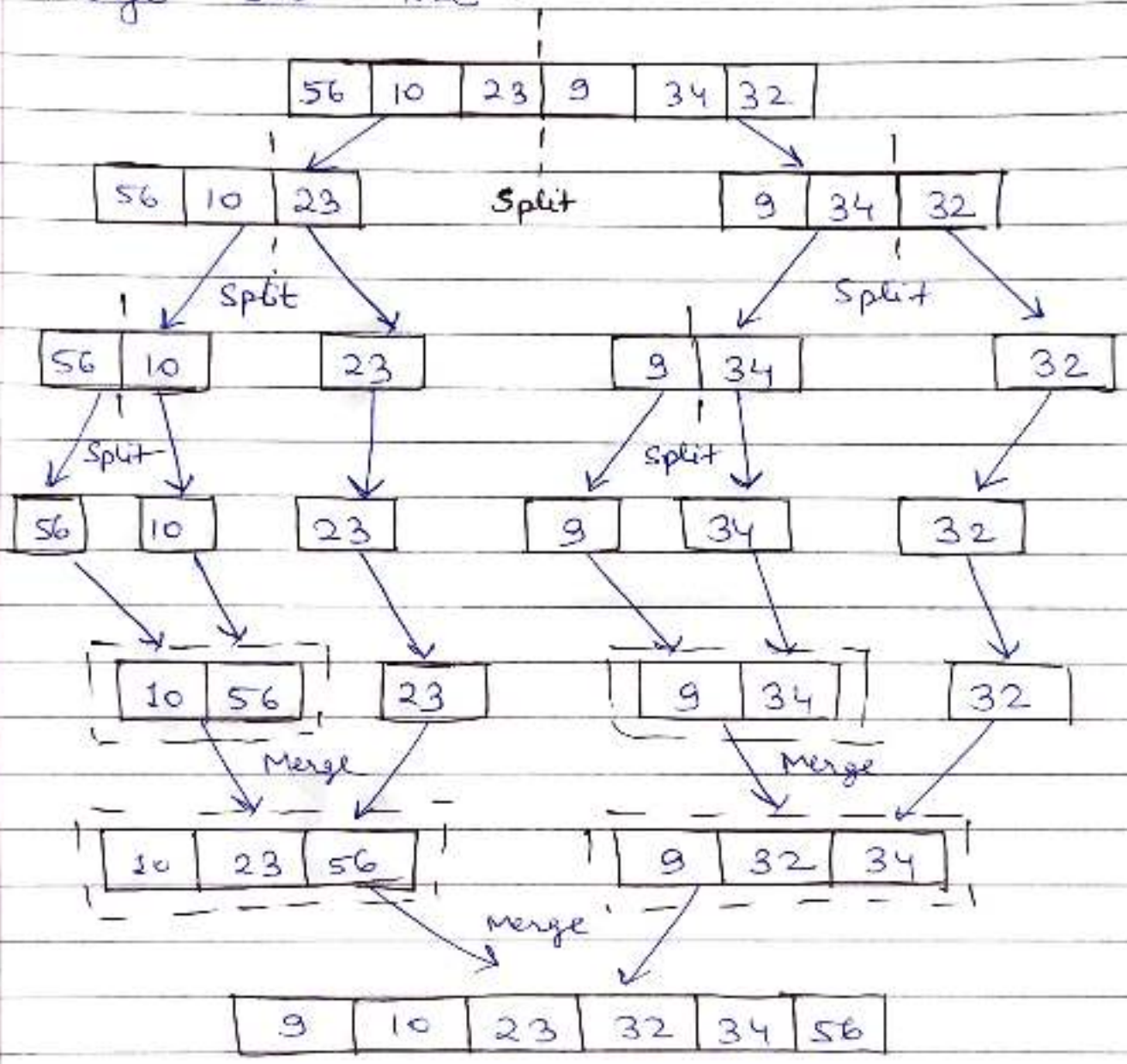
Divide : Break the given problem into sub-problems of same type.

Conquer : Recursively solve these sub-problems.

Combine : Appropriately combine the answers.

- **Concept**

→ Merge sort tree :



→ ~~But~~ Till the step 4 they are splitting and then after combining all of the array elements in the sorted form.
~~etc~~

* Algorithm of merge sorting on two sorted array

→ Step 1 : Put size and element in two array and store them separately in two array variable say size 1, arr 1, size 2, arr 2 ....

Step 2 : Create another array which will be store the merge array with size mergesize = size 1 + size 2, & say mergeArray [Merge size].

Step 3 : Initialize two variable end 1 = 0 and end 2 = 0. Both these variable will keep track of total merged elements from given two array individually.

Step 4 : Run a loop from 0 to mergesize. The loop structure must look like for (merged = 0; mergeEnd < mergesize; mergeEnd ++)

Step 5 : Inside loop check for smallest element in two array. which is if (array 1 [end 1] < array 2 [End 2] then assign element if first array to merge array i.e mergeArray [merge end] = arr 1 [end 1] and increment end 1. otherwise store mergeArray [mergeend] = arr 2 [end 2]; and increment end 2.

Step 6 : After loop merge the remaining array elements if any.

Program :

```
#include <stdio.h>
int main ()
{
    int i, j, n, K ;
    printf ("Enter the size of first array :");
    scanf ("%d", &n);
    int arr1[n] ;
    printf ("Enter the elements of first array :");

    for (i=0 ; i<n ; i++)
    {
        scanf ("%d", &arr1[i] ;
    }
    printf ("Enter the size of second array);
    scanf ("%d", &K);
    int arr2 [K];
    printf ("Enter the elements of second array);

    for (j =0 ; j<K ; j++)
    {
        scanf ("%d", &arr2 [j]);
    }

    int mergearr [n+K];
```

```
    i = j = 0 ;
    int end ;

for (end=0 , end < n+k ; end ++)
{
    if (i<n && j<k)
    { if (arr1[i] < arr2[j])
      {mergearr [end] = arr [i] ;
        i++ ;
      }
    else
     {
      mergearr [end] = arr2 [j] ;
     }
}
.
else if (i<n)
{
   mergearr [end] = arr1 [i] ;
    i++ ;
  }
 else
  {
    merge arr [end] = arr2 [j] ;
    j++ ;
  }
}

printf ("the merged array is :\n") ;
```

```
    for (end = 0 ; end < n + k ; end++)
    {
        printf("%d ", merge arr[end]);
    }
    printf("\n");

    return 0;
}
```

- **Time Complexity**

→ The time complexity of this algorithm is $O(n+k)$, where n is the size of first array and k is the size of second array. This simplifies to $O(n)$ as the loop is executed only once.

- **Space complexity**

→ The space complexity of this algorithm is $O(n+k)$, where

$n \longrightarrow$ size of first array
$k \longrightarrow$ size of second array.

- **Output**

→ The output of the above program is :

```
Enter the size of first array : 5
Enter the elements of first array :
2
4
6
8
10
Enter the size of second array : 3
Enter the elements of second array :
3
5
7
The merged array is :
2 3 4 5 6 7 8 10
```

—X—

• Experiment no - 04 •

• **Aim** : Develop a 'C' program to create and implement a STACK using arrays.

• **Theory**

* **Stack** :

→ A stack is a list in which all insertions and deletions are made at / from one end, called the top. It is collection of the contiguous cells, stacked of each other.

→ The last element to be inserted into the stack will be the first to be removed. Thus, stacks are sometimes reffered to as Last in first out (LIFO) lists.

→ The operations that can be performed on a stack is push and pop.

1) **Push** : Push is to insert an element at the top of the stack

pop : pop is to used for deleting an element that is at the topmost position in the stack.

- **Algorithm**

→ **Step 1** : Declare an array ahead of time called array.

Step 2 : Declare a structure called stack that contains the top of the stack and capacity to field.

Step 3 : The variable called top of the stack is initialized to -1.

Step 4 : To push an element of into the stack, increment top of stack and then set array [top of stack] = x.

Step 5 : To pop an element from the array, set the return value to the array [Top of stack], and then decrement Top of stack.

- if stack status is overflow we can't push the element into the stack.
- Otherwise, we can add the data to the stack.
- Move top to next position.

- **Program**

→
```
#include <stdio.h>
#define MAX 6
  void push();
  void pop();
  void display();

int stack[MAX], top=-1, item;
 int main()
{
   int ch;
   do
    {
      printf("\n\n\n 1. Push \n 2. pop\n 3: Display
          \n 4. Exit ");
     printf ("\n Enter your choice :");
     scanf ("%d", &ch);
     Switch(ch)
     {
        Case 1 : Push ();
        break;

        Case 2 : pop ();
        break;

        Case 3 : display ();
        break;
```

```
            Case 4 : default;
                printf ("\n Invalid entry!. please try again ");
        }
    }
    while (ch != 4);
}
void push (void)
    {
        if (top == MAX-1)
        printf ("\n Stack is full ");
        else
        {
            printf ("\n Enter Item. ");
            scanf ("%d", &item);
            top ++;
            Stack [TOP] = item;
            printf ("\n Item inserted = %d", item);
        }
}

void pop (void)
    {
        if (top == -1)
        printf ("\n Stack is Empty ");
        else
        {
            item = Stack [top];
            top --;
            printf (" item deted = %d ", item);
        }
}
```

```
void display (void)
{
    int i;
    if (top == -1)
     printf ("\n Stack is empty");

    else
     {
       for (i= top; i>=0; i--)
        {
          printf ("\n %d", stack [i]);
        }
     }
}
```

- **Output**

→ 1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1
Enter Item : 5
Item inserted : 5

1. Push
2. pop
3. Display

4. Exit

Enter your choice : 1
Enter Item : 8
Item inserted = 8

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3
8
5
1. push
2. pop
3. Display
4. Exit

Enter your choice : 2

Item deleted = 8
1. push
2. pop
3. display
4. Exit

Enter your choice : 3
5
1. push

2. pop
3. Display
4. Exit

Enter your choice : 4

Invalid entery . Please try again .

——— x ———

• Experiment no - 05 •

- **Aim** : Develop a 'C' program to create and implement a queue using array.

- **Theory**

→ Use three functions for three operations like insert, delete and display.

→ Use switch statement to access these functions.

Program :

→
```
#include <stdio.h>
#define MAX 50

void insert ();
void delete ();
void display ();
int queue_array [MAX];
int rear = -1;
int front = -1;

main ()
{
    int choice;
```

```
int item;
while (1)
{
printf ("1. Insert element to queue\n");
printf ("2. Delete element from queue\n");
printf ("4. Quit\n");
printf ("Enter your choice:")
Scanf ("%d", &choice")

Switch (choice)
    {
    Choice 1 :- insert ();
    break;

    Case 2 : delete ();
    break;

    Case 3 : display ();
     break;

    Case 4 : Exit

    Case 5 : Defalelt
        fmt ("Wrong choice \n");
```

```
void insert (int item)
{
    if (rear == MAX-1)
    {
        printf ("The Queue is full");
    }
    else if (front == -1 && rear == -1)
    {
        front = rear = 0;
        queue[rear] = item;
    }
    else
    {
        rear ++;
        queue[rear] = item;
    }
}

void delete ()
{
    if (front == -1 && rear == -1)
    {
        printf ("The queue is Empty");
    }
    else if (front == rear)
    {
        front = rear = -1;
    }
```

```
        else
          {
            printf ("The deleted element from
                the queue is %d ", queue[front]);
            front ++;
          }
        }

void display ();
    {
     int i;
     if (front == -1 && rear == -1)
       {
         printf ("The queue is Empty");
       }
     else
       {
         printf (" The elements of queue are :");

         for (i= front ; i< rear +1 , i++)
           {
             printf ("%d ", queue[i]);
           }

       }
    }
 return 0;
}
```

Divya 17/08/2023

—— × ——

```
        else
            {
            printf ("The deleted element from
                the queue is %d ", queue[front]);
            front ++;
            }
        }

void display ();
    {
    int i;
    if (front == -1 && rear == -1)
        {
        printf ("The queue is Empty");
        }
    else
        {
        printf ("The elements of queue are :");
        for (i = front ; i < rear + 1 ; i++)
            {
            printf ("%d ", queue[i]);
            }
        }
    }
return 0;
}
```

Adila 14/09/2023

—— x ——