Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours]

[Max. Marks : 70

Instructions to the candidates :

1) *Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.*

2) *Figures to the right indicate full marks.*

3) *Neat diagrams must be drawn wherever necessary.*

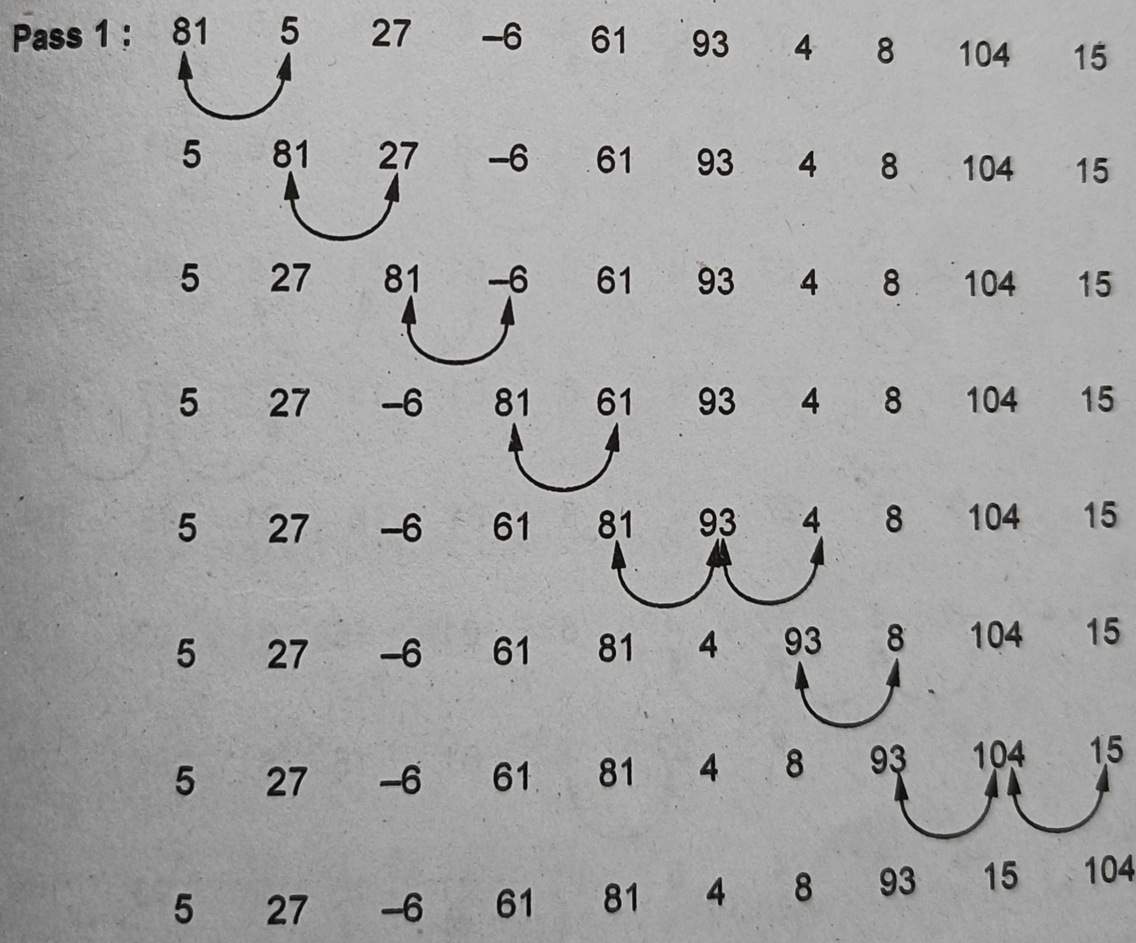
4) *Make suitable assumption whenever necessary.*

Q.1 a) Write an algorithm of bubble sort and sort the following numbers using bubble sort and show the contents of an array after every pass.

81, 5, 27, – 6, 61, 93, 4, 8, 104, 15

[9]

Ans. :



```
Pass 2 :  5   27  -6   61  81   4   8   93  15  104
          5   -6  27   61  81   4   8   93  15  104
          5   -6  27   61   4  81   8   93  15  104
          5   -6  27   61   4   8  81   93  15  104
          5   -6  27   61   4   8  81   15  93  104
          5   -6  27   61   4   8  81   15  93  104

Pass 3 :  5   -6  27   61   4   8  81   15  93  104
         -6    5  27   61   4   8  81   15  93  104
         -6    5  27    4  61   8  81   15  93  104
         -6    5  27    4   8  61  81   15  93  104
         -6    5  27    4   8  61  15   81  93  104
         -6    5  27    4   8  61  15   81  93  104

Pass 4 : -6    5  27    4   8  61  15   81  93  104
         -6    5   4   27   8  61  15   81  93  104
         -6    5   4    8  27  61  15   81  93  104
         -6    5   4    8  27  15  61   81  93  104
         -6    5   4    8  27  15  61   81  93  104
```

```
Pass 5 : -6    5   4    8  27  15  61   81  93  104
         -6    4   5    8  27  15  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104

Pass 6 : -6    4   5    8  15  27  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104

Pass 7 : -6    4   5    8  15  27  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104

Pass 8 : -6    4   5    8  15  27  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104

Pass 9 : -6    4   5    8  15  27  61   81  93  104
         -6    4   5    8  15  27  61   81  93  104
```

Thus we get a sorted list of elements as -

– 6 4 5 8 15 27 61 81 93 104

b) *Explain the radix sort. Sort the following numbers in ascending order.*
14, 1, 66, 74, 22, 36, 41, 59, 64, 54
Obtain the time and space complexity of your algorithm. [9]

Ans. : Consider, the unsorted array of 10 elements

14 1 66 74 22 36 41 59 64 54

Step 1 : Now, sort the elements according to last digit -

Last digit	0	1	2	3	4	5	6	7	8	9
Elements		1, 41	22		14, 54, 64,74		36,66			59

Step 2 : Now sort the elements based on second last digit.

Second last digit	0	1	2	3	4	5	6	7	8	9
Elements	01	14	22	36	41	54, 59	64, 66	74		

Since the list of elements is of two digits, that is why we stop comparing. Thus we get sorted list as -

01 14 22 36 41 54 59 64 66 74

Algorithm : Refer Q.18 of Chapter 3.

Time complexity : The time complexity of radix sort is O(nlogn)

OR

Q.2 a) *Explain internal and external sorting by taking suitable example of each type.* **(Refer Q.9 of Chapter 3)** **[9]**

b) *Write a short note on Fibonacci search with suitable example.* **(Refer Q.7 of Chapter 3)** **[9]**

Q.3 a) *Write pseudo C++ code for addition of two polynomials using singly linked list.* **(Refer Q.26 of Chapter 4)** **[9]**

b) *What is dynamic data structure ? Explain the circular linked list with its basic operations.* **[9]**

Ans. : Dynamic Data Structure : Refer Q.1 of chapter 4

Circular Linked List :

(1) Insertion of a node in circular linked list :

```
void sll::insert_head()
{
node *New,*temp;
New=new node;
New->next=NULL;
cout<<"\n Enter The element which you want to insert ";
cin>>New->data;
if(head==NULL)
   head=New;
else
{
      temp=head;
      while(temp->next!=head)
         temp=temp->next;
      temp->next=New;
      New->next=head;
      head=New;
      cout<<"\n The node is inserted!";
}
}
/*Insertion of node at last position*/
void sll::insert_last()
{
   node *New,*temp;
   New=new node;
   New->next=NULL;
   cout<<"\n Enter The element which you want to insert ";
   cin>>New->data;
   if(head==NULL)
      head=New;
   else
   {
   temp=head;
```

```
    while(temp->next!=head)
          temp=temp->next;
    temp->next=New;
    New->next=head;
    cout<<"\n The node is inserted!";
  }
}
void sll::insert_after()
{
 int key;
 node *New,*temp;
 New= new node;
 New->next=NULL;
 cout<<"\n Enter The element which you want to insert ";
 cin>>New->data;
 if(head==NULL)
 {
   head=New;
 }
else
{
  cout<<"\n Enter The element after which you want to insert the
  node ";
  cin>>key;
  temp=head;
  do
  {
    if(temp->data==key)
    {
       New->next=temp->next;
       temp->next=New;
       cout<<"\n The node is inserted";
       return;
  }
```

```
     else
        temp=temp->next;
   }while(temp!=head);
  }
}
```

(2) Deletion of a node in circular linked list :

```
void sll::Delete()
{
 int key;
 struct node *temp,*temp1;
 cout<<"\n Enter the element which is to be deleted";
 cin>>key;
 temp=head;
 if(temp->data==key)/*If header node is to be deleted*/

{
   temp1=temp->next;
   if(temp1==temp)
  /*if single node is present in circular linked list
  and we want to delete it*/
   {
     temp=NULL;
     head=temp;
     cout<<"\n The node is deleted";
   }
   else  /*otherwise*/
   {
    while(temp->next!=head)
     temp=temp->next;/*searching for the last node*/
     temp->next=temp1;
     head=temp1;/*new head*/
     cout<<"\n The node is deleted";
   }
}
```

If a single node is present in the list and we want to delete it

```
else
{
  while(temp->next!=head) /* if intermediate node is to
              be deleted*/
  {
  if((temp->next)->data==key)
  {
    temp1=temp->next;
    temp->next=temp1->next;
    temp1->next=NULL;
    delete temp1;
    cout<<"\n The node is deleted";
  }
  else
    temp=temp->next;
  }
 }
}
```

The previous node of the node to be deleted is searched. Here **temp 1** is the node to be deleted and **temp** is the previous node of **temp 1**

OR

Q.4 a) *Write a pseudo code for the addition of a node after the position 'P' in singly linked list.* **(Refer Q.7 of Chapter 4)** [9]

b) *Explain the doubly linked list with it's basic operations; list the advantages of doubly linked list over singly linked list.*
(Refer Q.16, Q.17 and Q.18 of Chapter 4) [9]

Q.5 a) *Write a pseudo code for basic operations of stack.*
(Refer Q.4 of Chapter 5) [8]

b) *What are the variants of recursion, explain with example.* **(Refer Q.19 of Chapter 5)** [9]

OR

Q.6 a) *Explain the linked implementation of stack with suitable example.* **(Refer Q.14 and Q.15 of Chapter 5)** [8]

b) *Write pseudo code for infix to postfix expression; Explain the need of conversion of expression.* **(Refer Q.9 of Chapter 5)** [9]

Q.7 a) *Define the following terms with example*
i) Linear queue **(Refer Q.1 of Chapter 6)**
ii) Circular queue **(Refer Q.6 of Chapter 6)**
iii) Priority queue. **(Refer Q.11 of Chapter 6)** [8]

b) *Write pseudo C++ code to implement linked queue.*
(Refer Q.9 of Chapter 6) [9]

OR

Q.8 a) *Explain array implementation of priority queue with all basic operation.* [8]

Ans. : The implementation of priority queue using arrays is as given below -

1. Insertion operation

While implementing the priority queue we will apply a simple logic. That is while inserting the element we will insert the element in the array at the proper position. For example if the elements are placed in the queue as -

9	12			
que[0]	que[1]	que[2]	que[3]	que[4]
front	rear			

And now if an element 8 is to be inserted in the queue then it will be at 0th location as -

8	9	12		
que[0]	que[1]	que[2]	que[3]	que[4]
front		rear		

If the next element comes as 11 then the queue will be -

8	9	11	12	
que[0]	que[1]	que[2]	que[3]	que[4]
front			rear	

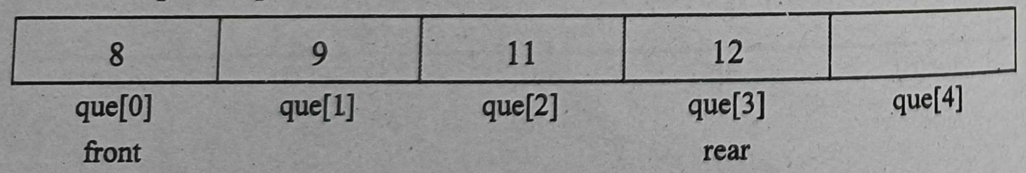
The C++ function for this operation is as given below -

```
int Pr_Q::insert(int rear,int front)
{
    int item,j;
    cout<<"\nEnter the element: ";
    cin>>item;
    if(front ==-1)
            front++;
    j=rear;
    while(j>=0 && item<que[j])
    {
    que[j+1]=que[j];
    j--;
    }
    que[j+1]=item;
    rear=rear+1;
    return rear;
}
```

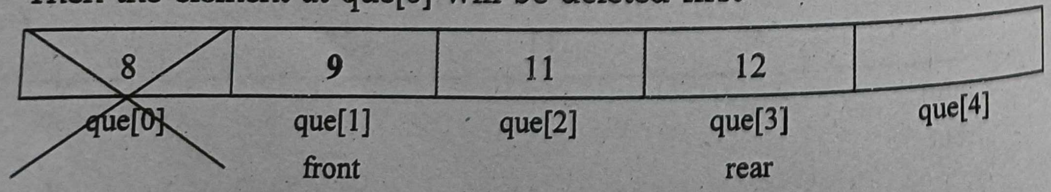
2. Deletion operation

In the deletion operation we are simply removing the element at the front.

For example if queue is created like this –



Then the element at que[0] will be deleted first



and then new front will be que[1].

The deletion operation in C++ is as given below -

```
int Pr_Q::delet(int front)
{
    int item;
    item=que[front];
    cout<<"\n The item deleted is "<<item;
    front++;
    return front;
}
```

b) *Write a pseudo C++ code to implement circular queue using array.* **(Refer Q.7 of Chapter 6)** [9]

END...