**DS- Exam 03 (Theory Exam 02 ) Answer Script**

| Question No. 01` |
| --- |
| | Index | 0 | 1 | 2 | 3 | 4 | 5 | | --- | --- | --- | --- | --- | --- | --- | | Value | 7 | 2 | 13 | 2 | 11 | 4 |   Write down all the steps of Bubble Sort on the Following Array. [10] |
| Answer: steps of bubble sort (without optimization) is shown below.  iteration no: 1  step 1 : **7 2** 13 2 11 4 >> 2 7 13 2 11 4  step 2 : 2 **7 13** 2 11 4 >> 2 7 13 2 11 4  step 3 : 2 7 **13 2** 11 4 >> 2 7 2 13 11 4  step 4 : 2 7 2 **13 11** 4 >> 2 7 2 11 13 4  step 5 : 2 7 2 11 **13 4 >>** 2 7 2 11 4 13  iteration no: 2  step 1 : **2 7** 2 11 4 13 >> 2 7 2 11 4 13  step 2 : 2 **7 2** 11 4 13 >> 2 2 7 11 4 13  step 3 : 2 2 **7 11** 4 13 >> 2 2 7 11 4 13  step 4 : 2 2 7 **11 4** 13 >> 2 2 7 11 4 13  step 5 : 2 2 7 4 **11 13 >>** 2 2 7 4 11 13  iteration no: 3  step 1 : **2 2** 7 4 11 13 >> 2 2 7 4 11 13  step 2 : 2 **2 7** 4 11 13 >> 2 2 7 4 11 13  step 3 : 2 2 **7 4** 11 13 >> 2 2 4 7 11 13  step 4 : 2 2 4 **7 11** 13 >> 2 2 4 7 11 13  step 5 : 2 2 4 7 **11 13** >> 2 2 4 7 11 13  iteration no: 4  step 1 : **2 2** 4 7 11 13 >> 2 2 4 7 11 13  step 2 : 2 **2 4** 7 11 13 >> 2 2 4 7 11 13  step 3 : 2 2 **4 7** 11 13 >> 2 2 4 7 11 13  step 4 : 2 2 4 **7 11** 13 >> 2 2 4 7 11 13  step 5 : 2 2 4 7 **11 13** >> 2 2 4 7 11 13  iteration no: 5  step 1 : **2 2** 4 7 11 13 >> 2 2 4 7 11 13  step 2 : 2 **2 4** 7 11 13 >> 2 2 4 7 11 13  step 3 : 2 2 **4 7** 11 13 >> 2 2 4 7 11 13  step 4 : 2 2 4 **7 11** 13 >> 2 2 4 7 11 13  step 5 : 2 2 4 7 **11 13** >> 2 2 4 7 11 13  iteration no: 6  step 1 : **2 2** 4 7 11 13 >> 2 2 4 7 11 13  step 2 : 2 **2 4** 7 11 13 >> 2 2 4 7 11 13  step 3 : 2 2 **4 7** 11 13 >> 2 2 4 7 11 13  step 4 : 2 2 4 **7 11** 13 >> 2 2 4 7 11 13  step 5 : 2 2 4 7 **11 13** >> 2 2 4 7 11 13 |

| Question No. 02 |
| --- |
| What are the limitations of the Bubble Sort approach? Explain them with an Example and write down what could be the solution.        [10] |
| Answer:  Let’s consider a pseudo code for the bubble sort algorithm-  Here n= size of the array.  for(int i=1; i<=n; i++){  for(int j=0; j<n-1; j++){  if(array[j]>array[j+1]){  swap(array[j],array[j+1]);  }  }  Limitations:   1. Redundant testing in a single iteration: in every iteration, bubble sort algorithm, compares every element with the next element until j=n-2. This is redundant. 2. Redundant iteration: if the array is already sorted or sorted after some iteration than, from next iteration all other iterations are unnecessary. If we don’t modify the above mentioned code, it will continue to iterate for n-1 times, which is fixed.   Example and solution: to solve this we can modify our previous code.  for(int i=1; i<n; i++){  int flag=0;  for(int j=0; j<n-i; j++){ //means runs for n-i-1 times, instead of n-1(constant time)  if(array[j]>array[j+1]){  swap(array[j],array[j+1]);  flag=1;  }  }  if(flag==0) break;  }   1. To solve first problem, we can modify the inner for loop, which will continue until n-i-1 times. For n=4, and i=1, j will continue till 4-1-i=4-1-1=2 times. 2. We can use a variable “flag”, which will monitor if swapping operation is happening or not in the inner loop. If there is no swapping in an iteration, outer loop will break and iterations stop.   Example:   | Bubble sort without optimization | After solving limitation 1 | After solving both limitations | | --- | --- | --- | | enter array size and elements  4  2 1 3 4  Before bubble sorting  2 1 3 4  iteration no: 1  step 1 : 2 1 3 4  step 2 : 1 2 3 4  step 3 : 1 2 3 4  iteration no: 2  step 1 : 1 2 3 4  step 2 : 1 2 3 4  step 3 : 1 2 3 4  iteration no: 3  step 1 : 1 2 3 4  step 2 : 1 2 3 4  step 3 : 1 2 3 4 | enter array size and elements  4  2 1 3 4  Before bubble sorting  2 1 3 4  iteration no: 1  1 2 3 4  1 2 3 4  1 2 3 4  iteration no: 2  1 2 3 4  1 2 3 4  iteration no: 3  1 2 3 4 | enter array size and elements  5  1 3 2 4 5  Before bubble sorting  1 3 2 4 5  iteration no: 1  1 3 2 4 5  1 2 3 4 5  1 2 3 4 5  1 2 3 4 5  iteration no: 2  1 2 3 4 5  1 2 3 4 5  1 2 3 4 5 | |

| Question No. 03 |
| --- |
| Write down all the steps of **Counting Sort**on the Following Array. Why do we start scanning from backward in the last step of Counting Sort (Use the given array to explain)?[10+2]   | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Value | 7 | 2 | 9 | 2 | 7 | 4 | 5 | 1 | |
| Answer: Frequency array:  max=9 , Freq[10]   | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Value | 0 | 1 | 2 | 0 | 1 | 1 | 0 | 2 | 0 | 1 |   Cumulative array C[10]:   | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Value | 0 | 1 | 3 | 3 | 4 | 5 | 5 | 7 | 7 | 8 |   Final array[8]:   | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Value | 1 | 2 | 2 | 4 | 5 | 7 | 7 | 9 |   Steps:   | Steps | i | a[i] | k | k=k-1 | Final[k]=a[i] | | --- | --- | --- | --- | --- | --- | | 1 | 7 | a[7]=1 | C[1]=1 | 0 | Final[0]=a[7]=1 | | 2 | 6 | a[6]=5 | C[5]=5 | 4 | Final[4]=a[6]=5 | | 3 | 5 | a[5]=4 | C[4]=4 | 3 | Final[3]=a[5]=4 | | 4 | 4 | a[4]=7 | C[7]=7 | 6 | Final[6]=a[4]=7 | | 5 | 3 | a[3]=2 | C[2]=3 | 2 | Final[2]=a[3]=2 | | 6 | 2 | a[2]=9 | C[9]=8 | 7 | Final[7]=a[2]=9 | | 7 | 1 | a[1]=2 | C[2]=2 | 1 | Final[1]=a[1]=2 | | 8 | 0 | a[0]=7 | C[7]=6 | 5 | Final[5]=a[0]=7 |   Reason for starting from the last index:  In general sense, an element which comes first, should be placed at the lowest index after sorting.  To ensure this “sanity check” we traverse counting sort algorithm backward.in this example,7 comes twice(0 and 4 index) in the unsorted array, first 7 placed at index 5 and second 7 placed at index 6 in the final array. |

| Question No. 04 |
| --- |
| Write down all the steps of Insertion Sort on the Following Array.[10]   | Index | 0 | 1 | 2 | 3 | 4 | 5 | | --- | --- | --- | --- | --- | --- | --- | | Value | 5 | 1 | 3 | 8 | 2 | 2 | |
| Answer: steps,  Before insertion sorting  5 1 3 8 2 2  iteration: 1, key: 1  steps 1: **5 5** 3 8 2 2 >> 5 is copied to index 01(as key<5 that means 1 < 5)  steps 1: **1** 5 3 8 2 2 >> 5 is replaced with 1 at index 00  iteration: 2, key: 3  steps 1: 1 **5 5** 8 2 2 >> 5 is copied to index 02 (as 3 < 5)  steps 2: 1 **3** 5 8 2 2 >> 5 is replaced with 3 at index 01  iteration: 3, key: 8  steps 1: 1 3 **5 8** 2 2 >> 8 is not modified (as 5 is already less than the key )  iteration: 4, key: 2  steps 1: 1 3 5 **8 8** 2 >> 8 is copied to index 04 (as 2 < 8)  steps 2: 1 3 **5 5** 8 2 >> 5 is copied to index 03 (as 2 < 5)  steps 3: 1 **3 3** 5 8 2 >> 3 is copied to index 02 (as 2 < 3)  steps 4: 1 **2** 3 5 8 2 >> 3 is replaced with 2 at index 01 (as 2 < 3 and the process ends because 1<2 at index 00)  iteration: 5, key: 2  steps 1: 1 2 3 5 **8 8** >> 8 is copied to index 05 (as 2 < 8)  steps 2: 1 2 3 **5 5** 8 >> 5 is copied to index 04 (as 2 < 5)  steps 3: 1 2 **3 3** 5 8 >> 3 is copied to index 03 (as 2 < 3)  steps 4: 1 2 **2** 3 5 8 >> 3 is replaced with 2 at index 02 (as 2 < 3 and the process ends because 2 is not less than 2 at index 01) |

| Question No. 05 |
| --- |
| Question: Find ‘6’ in the following array using Binary Search and show the steps.  Draw the Binary Search Tree for the given Array using the Binary Search technique.[10]   | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Value | 1 | 2 | 9 | 11 | 17 | 24 | 26 | 31 | 32 | 45 | |
| Answer: steps:   1. lb 0 ub 9 mid 4 array[mid] 17 2. lb 0 ub 3 mid 1 array[mid] 2 3. lb 2 ub 3 mid 2 array[mid] 9 4. not found   Binary search Tree is shown below, |

| Question No. 06 |
| --- |
| Assume a 2D array is declared as **int arr[50][45]**. The value of the base address of the array is **arr[0][0] = 760**. Find out the location of **arr[6][13]. (**An Integer is a word addressable (4 bytes) datatype) [6] |
| Answer:  For row-major ordering:  A[i][j]=Base +w[n(i-base)+(j-base)]……eq. 1  Here.  Base=760, w=4, n=45(elements per row), i=6, j=13, base=0  Using eq. 1,  A[6][13]=760+4 [45(6-0) + (13-0)]=760+1132= 1892 **ANS.** |

| Question No. 07 |
| --- |
| Write down Three Limitations of the array which can be solved by the use of Linked List and provide necessary figures and explanations.[10] |
| Answer: Three major limitations with array are as follows,   |  |  |  | | --- | --- | --- | | Fig A | Fig B | Fig C |  1. Fixed size: array size should be declared during initialization. Fan example, if we want to store marks of 100 students, we can declare an array size of int marks[100] , the problem is if any new student admits, we don’t have extra memory space in our array. On the other hand, if we declared the array as int marks[200], and there are 101 students, there are 99 redundant memory spaces and this is a waste. In Fig A, 200 bytes of memory is allocated, but 100 bytes is used, other 100 bytes are not used and is a waste. 2. Contiguous or sequential memory locations: array allocates memory locations in a sequential manner. An array int array[20] will allocate 20 memory locations sequentially. In Fig B, two different arrays, hold 70 bytes and 50 bytes separately. There are 30 bytes and 50 bytes (a total of 80 bytes) of unused memory, now if another array needs 80 bytes, it can’t use these 80 bytes, because these 80 bytes are not sequentially available. 3. Insertion and deletion of an element are cumbersome: if we want to insert or delete an element from an array index, we have to move every element after that index. In Fig C, an array has 4 elements initially. Then if we want to insert 4 at index 2, two elements (7 and 3) need to be shifted to one location also there a must be free memory location for this shifting. The same thing for deletion, if we want to delete an element from an index, elements also need to be shifted. This can increase the duration if the array is very large.   All these three problems can be solved using the Linked list because the Linked list does not allocate memory locations sequentially, so we can use randomly available memory locations and insertion or deletion from in between memory locations is faster than an array. |

| Question No. 08 |
| --- |
| Write down the code to insert an element at the head and Tail of the Linked List. Do you think it is possible to print a linear linked list in a reverse way (Make a function named reversedListPrint() if possible)?[10+10] |
| Answer:  Code to insert an element at the head or tail of the Linked list.  Code:  #include <bits/stdc++.h>  using namespace std;  class Node  {      public:          int value;          Node \*Next;      //constructor      Node(int val){          value=val;          Next=NULL;      }  };  //inert newnode at head function  void insertAtHead(Node\* &head, int val){      //s1: creat newNode      Node\* newNode=new Node(val);      //s2: replace newNode->next with head (head address)      newNode->Next=head;      //s3: update of head-make newNode as head      head=newNode;  }  //inert newnode at tail function  void insertAtTail(Node\* &head, int val){      Node\* newNode=new Node(val);      //there are 2 cases      //case 1: when linked list is empty      if(head==NULL){          head=newNode;          return;      }      //case 2: linked list is not empty-already have some elements      Node\* temp=head;      while(temp->Next!=NULL){          temp=temp->Next;      }      temp->Next=newNode;  }  //print function-print only values  void display(Node\* n){      while(n!= NULL){            cout<<n->value<<" ";          n=n->Next;      }  }  //print function-print with details  void displayDetails(Node\* n){      while(n!= NULL){            cout<<n<<"      "<<n->value<<"      "<<n->Next<<endl;          n=n->Next;      }  }  int main(){      Node\* head=NULL;      //input using loop      cout<<"enter for both head and tail,press 1 to add more"<<endl<<"enter choiche: ";      int choice;      cin>>choice;      while(choice==1){          cout<<"enter value: ";          int n;          cin>>n;          insertAtTail(head,n);          insertAtHead(head,n);          display(head);          cout<<endl<<"want to add more, enter 1: ";          cin>>choice;      }      //print using function      cout<<endl;      cout<<"node addreess "<<" value in node "<<" Next node address "<<endl;      displayDetails(head);      return 0;  }  **Part2: Yes Linear linked list can be printed in reverse.**  If ,head is NULL, the Linked list is empty and return from the reverse function, then recursively call the reverse function.  Code:  //print in reverse  #include <bits/stdc++.h>  using namespace std;  class Node  {      public:          int value;          Node \*Next;      //constructor      Node(int val){          value=val;          Next=NULL;      }  };  //inert newnode at head function  void insertAtHead(Node\* &head, int val){      //s1: creat newNode      Node\* newNode=new Node(val);      //s2: replace newNode->next with head (head address)      newNode->Next=head;      //s3: update of head-make newNode as head      head=newNode;  }  //inert newnode at tail function  void insertAtTail(Node\* &head, int val){      Node\* newNode=new Node(val);      //there are 2 cases      //case 1: when linked list is empty      if(head==NULL){          head=newNode;          return;      }      //case 2: linked list is not empty-already have some elements      Node\* temp=head;      while(temp->Next!=NULL){          temp=temp->Next;      }      temp->Next=newNode;  }  //print function-print only values  void display(Node\* n){      while(n!= NULL){            cout<<n->value<<" ";          n=n->Next;      }  }  //print function-print with details  void displayDetails(Node\* n){      while(n!= NULL){            cout<<n<<"      "<<n->value<<"      "<<n->Next<<endl;          n=n->Next;      }  }  //print in reverse  void reversedListPrint(Node\* n){      if(n==NULL) return; // empty list        reversedListPrint(n->Next);      cout<<n->value<<" ";  }  int main(){      Node\* head=NULL;      //input using loop      cout<<"enter for tail,press 1 to add more"<<endl<<"enter choiche: ";      int choice;      cin>>choice;      while(choice==1){          cout<<"enter value: ";          int n;          cin>>n;          insertAtTail(head,n);          //insertAtHead(head,n);          display(head);          cout<<endl<<"want to add more, enter 1: ";          cin>>choice;      }      //print using function      cout<<endl;      // cout<<"node addreess "<<" value in node "<<" Next node address "<<endl;      // displayDetails(head);      //cout<<endl;      cout<<"printing in reverse"<<endl;      reversedListPrint(head);      return 0;  } |

| Question No. 09 |
| --- |
| Answer the following questions using the Figure attached. [3 + 3 + 3 + 3]  https://lh3.googleusercontent.com/RJH7H-4ol2MOsy14H-tZY-yz_yOKoo_Ba7pAdO5TbxlkPQuIFH6tM6EST1P6fk7E_8FwxLz8fRe0TydQPNSxheTLZgx10mtcJWnQUFZ3RXiuK7B-G2vKrf6g5VGJz6_iBpvRc3K6KMAN-R2pBIaOx7Q   1. What is the value of Head? 2. What is the value of “?” marked address location? 3. What will be the value of Head->Next->Next->Value? 4. What will be the value of **Sum** following pseudocode snippets?   Sum = 0  Temp = Head  While ( Temp -> Next!= 1020){          Sum += Temp-> value          Temp = Temp -> Next  }  Sum -= Temp -> value; |
| Answer:   1. Value of Head = 5000 2. Value of “?” marked=1020, this is saved in 4th linked list. 3. 1 4. Sum=13-14=-1 |

Submitted by: Subrata Saha

Email:[Subratabaec@gmail.com](mailto:Subratabaec@gmail.com)

Date: 06.08.2022

THE END