**DS- Exam 05 (Theory- Mid-Term) Answer Script**

|  |
| --- |
| Question No. 01` |
| Calculate the time complexity of the following code snippets. |
| Answer:   1. O(log2(log2n)) 2. O(n) 3. O(n log2n) |

|  |
| --- |
| Question No. 02 |
| Write down all the steps of **Counting Sort** on the Following Array.   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Value | 3 | 3 | 1 | 7 | 7 | 4 | 4 | 5 | |
| Answer: Frequency array:  max=7 , Freq[8]   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | Value | 0 | 1 | 0 | 2 | 2 | 1 | 0 | 2 | 0 |   Cumulative array C[8]:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | Value | 0 | 1 | 1 | 3 | 5 | 6 | 6 | 8 | 8 |   Final array[8]:   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Value | 1 | 3 | 3 | 4 | 4 | 5 | 7 | 7 |   Steps:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Steps | i | a[i] | k | k=k-1 | Final[k]=a[i] | | 1 | 7 | a[7]=5 | C[5]=6 | 5 | Final[5]=a[7]=5 | | 2 | 6 | a[6]=4 | C[4]=5 | 4 | Final[4]=a[6]=4 | | 3 | 5 | a[5]=4 | C[4]=4 | 3 | Final[3]=a[5]=4 | | 4 | 4 | a[4]=7 | C[7]=8 | 7 | Final[7]=a[4]=7 | | 5 | 3 | a[3]=7 | C[7]=7 | 6 | Final[6]=a[3]=7 | | 6 | 2 | a[2]=1 | C[1]=1 | 0 | Final[0]=a[2]=1 | | 7 | 1 | a[1]=3 | C[3]=3 | 2 | Final[2]=a[1]=3 | | 8 | 0 | a[0]=3 | C[3]=2 | 1 | Final[1]=a[0]=3 | |

|  |
| --- |
| Question No. 03 |
| Find ‘4’ 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.   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | Value | 1 | 2 | 4 | 9 | 12 | 14 | 16 | 21 | 32 | 35 | |
| Answer: steps:   1. Let’s consider x=4. 2. lb index=0 , ub index=9, mid index=4, array[mid] =12 3. x<array[mid]-->go left 4. lb index=0, ub index= 3, mid index= 1 , array[mid] =2 5. x>array[mid]-->go right 6. lb index =2, ub index= 3, mid index = 2, array[mid]=4 7. array [mid]== x, x is found and return x.   Binary search Tree is shown below, |

|  |
| --- |
| Question No. 04 |
| Assume a 2D array is declared as int arr[70][65]. The value of the base address of the array is arr[0][0] = 1230. Find out the location of arr[3][18]. (An Integer is a word addressable (4 bytes) datatype) |
| Answer:  For row-major ordering:  A[i][j]=Base +w[n(i-base)+(j-base)]……eq. 1  Here,  i=3, j= 18, Base=1230, w=word size=4, n=items per row=65, base=0  so,  A[3][18]=1230+4[65(3-0)+(18-0)]=2082 |

|  |
| --- |
| Question No. 05 |
| Answer the following questions for the doubly linked list as shown below, where p = 12 , q = p+4, r = p+q, s = r-3, t = r+s.  a) head −> next −>next-> value = ?  b) last −> prev −> next->value = ?  c) temp −> prev −> prev −> prev= ?  d) temp −> next−> prev −>prev->value = ?  e) last −> prev −> prev −>next-> value = ? |
| Answer:  Here, p=12, q=16, r=28, s=25, t=53   1. 28 2. 53 3. NULL 4. 16 5. 25 |

|  |
| --- |
| Question No. 06 |
| Assume that you are given a single linked list as shown below. Write the statements to perform the following:  i) To insert 40 in between 33 and 47.  ii) To delete 14 from the list.  iii) To make a linear circular linked list from the current list. |
| 1. Answer:   1. First input the value and position (where to put the value). Here value is 40 and 40 should be placed at position 4. 2. Create a function with parameters as head pointer of the Linked List, position and value from the main (). Here, Value is 40, position is 4. 3. Create a new node with 40. 4. Traverse 2 times from head using temporary node. Now temp is at 33. 5. Give address of 47 to newNode->Next. 6. Finally, give new Node address to temp->Next.   code:  //insert at specific position  void insertAtspecificPosition(Node \*&head, int pos, int val)  { // for pos 1  if (pos == 1) // head  {  insertionAtHead(head, val);  return;  }  // works from pos 2  Node \*newNode = new Node(val);  Node \*temp = head;  for (int i = 0; i < pos - 2; i++)  {  temp = temp->Next;  }  newNode->Next = temp->Next;  temp->Next = newNode;  } |
| 2.Answer:   1. To delete 14, head need to be deleted. 2. Create a function with parameter as head pointer of the Linked List. 3. Check if head is empty, that means Linked list has any element or not. Here head contains 14. 4. Let’s assume, head as temporary node. 5. Make the second node as head by head->temp->next. 6. Delete the previous head node as delete temp.   Code:  // Deletion at head  void deletionAtHead(Node \*&head)  {  Node \*temp = head;  if (temp == NULL) // empty  {  cout << endl<< "nothing to delete" << endl;  return;  }  head = temp->Next;  delete temp;  } |
| 3.Answer:  Current list is 24 -> 33 -> 40 -> 47   1. Let’s consider head pointer as temp pointer. 2. Travers from head to last of the linked list using temp=temp->next. 3. Set the Next pointer of the last node(here 47) to the head pointer(here 24) using temp->Next=head.   Code:  //singly LL to circular LL  void makeCircular(Node \*&head)  {  if (head == NULL)  {  cout << "empty" << endl;  }  Node \*temp = head;  while (temp->Next != NULL)  {  temp = temp->Next;  }  //cout << temp->value;//last node  temp->Next=head;  } |

|  |
| --- |
| Question No. 07 |
| Write an algorithm to display the data stored in a doubly linked list in reverse order. Assume only the head pointer is given for the linked list. |
| Answer:  Steps to print doubly Linked List in reverse order:   1. Assume the head pointer as temp pointer 2. Initially check, if the temp or head pointer is NULL or not. If NULL, print ‘empty’ and return. 3. If temp or head is not NULL initially, then traverses from the head to the last node following the loop condition temp->next is not equal to NULL, and temp=temp->next; 4. Then traverse from the last node to the head node following the loop condition as temp is not equal to NULL and using temp=temp->prev.   Code snippet:  // reverse doubly LL  void reverse(doublyNode \*&head)  {  doublyNode \*temp = head;  if (temp == NULL)  {  cout << "empty doubly LL" << endl;  return;  }  // step 1: go to last node  while (temp->next != NULL)// dont go fot temp!=NULL  {  cout<<temp->value<<" ";  temp = temp->next;  }  cout<<temp->value<<endl;//to print the last value  cout<<endl;  // step 2:reverse from last node to head  while (temp != NULL)  {  cout << temp->value << " ";  temp = temp->prev;  }  cout << endl;  } |

|  |
| --- |
| Question No. 08 |
| Show the status of a STACK implemented by a linear linked list for the operations given below.  **Here, x= Last day of your birthday + 5, y=x+3, and z=y+x.**  **Push(x+y), Push(y+z), Pop(), Push(y\*z), Push(x\*y), Pop(), Pop()**  **1.4\*7=10** |
| Answer:  Here, let’s consider x=2+5=7, y=x+3=10, z=y+x=17  x=7, y= 10, z=17  Steps: (we know stack follows LIFO)   1. Push(x+y): Push (7+10=17)   Stack condition: 17; Head=Top=17   1. Push(y+z): Push (10+17=27)   Stack condition: 17, 27; Head=17, Top=27   1. Pop()   Stack condition: 17; Head=Top=17   1. Push(y\*z): Push (10\*17=170)   Stack condition: 17,170; Head=17, Top=170   1. Push(x\*y): Push (7\*10=70)   Stack condition: 17, 170, 70; Head=17, Top=70   1. Pop()   Stack condition: 17, 170; Head=17, Top=170   1. Pop()   Stack condition: 17; Head=Top=17 |

|  |
| --- |
| Question No. 09 |
| Show the effect of each of the statements given in the following code using a Stack. |
| Answer:   |  |  |  | | --- | --- | --- | | #include<stdio.h>  #include<string.h> | Required header files are imported. | | | int top=-1;  char Stack[4]={'\0'}; | Initially, variable top=-1 and array Stack of size 4 (character type) is declared. | | | From Push Function:  Stack[0]=C,  Stack[1]=S,  Stack[2]=E  // C <-- S <-- E | From Pop Function:  Stack[2]=E,  Stack[1]=S,  Stack[0]=C, | | int main() { | *m*ain() function starts | | | char Str1[4]={'\0'};  char Str2[4]={'\0'};  int i; | str1, str2 character type arrays of size 4 are declared, arrays are empty.  Integer type variable i is declared. | | | strcpy(Str1, "CSE"); | String "CSE" is copied to str1. | | | for(i=0; i<3; ++i){  Push(Str1[i]);  } | Push(str1[0])=push(C), Push(str1[1])=push(S), Push(str1[0])=push(E)  After this push operation top=2. | | | for(i=0; i<3; ++i){  Str2[i]=Pop();  } | str2[0] = E,  str2[1] = S,  str2[2] = C | | | printf("%s", Str2);  } | Print s2, ESC.  The end of *main* () function. | | |  | | | | void Push(char x){  Stack[++top]=x;  return;  } | Stack[0]=C,  Stack[1]=S,  Stack[2]=E | | | char Pop(void){  return Stack[top--];  } | Stack[2]=E is called,  Stack[1]=S is called,  Stack[0]=C is called, | | |  | | | //2 ,1 , 0 | |

|  |
| --- |
| Question No. 10 |
| What are the merits of implementing a QUEUE using Array in a circular fashion? How do you check the underflow and overflow in the QUEUE implemented circularly? |
| Answer:  Part 1:  Advantages of circular queue implementation using an array:  As an array has a limited memory, when the rear pointer reaches the last index, no more elements can be inserted. Although because of dequeue operations there may be vacant locations before the front pointer but they cannot be used if the queue is not circular.  In the case of a circular queue, elements can be inserted in vacant memory locations. After the last index, the rear pointer goes to index zero (if that is vacant) by using the modulo operator.  This increases the efficiency of memory uses.  Part 2:  Let’s consider, F=Front pointer, R=Rear pointer, array size=n  Overflow:  When there is no memory left to enqueue a new element overflow occurs.  Steps to detect overflow.   1. At first F=R=-1. 2. If (F=R=-1), after the first enqueue, F and R are incremented to 0 and the value is inserted into the array index 0.**(F=0, R=0)** 3. For the next enqueue operations, a new variable s is considered, where s= ((r+1) mod n) and then a new condition s! =F is checked. 4. This follows until s=n-1 and the value is inserted up to index n-1. 5. Next when r is n-1, s becomes equal to 0. 6. If there is no dequeue operation happens until now then still **F=0**. 7. The condition **s! =F is** False, which means s==F, so the value is going to be inserted at index 0, the same index where F is now located, this situation is called overflow.   In summary, it’s always checked whether s becomes equal to F or not, to solve the overflow problem.  Underflow:  When there is no element in the queue and the dequeue operation is called, there is no element to return, and underflow occurs.  Steps to detect underflow:   1. At first F=R=-1. 2. Let’s consider the queue is full (unto index n-1) and F=0 and R=n-1. 3. For every dequeue operation, two conditions array [F] ==0 and F==R is checked. 4. For every dequeue operation, F increments by 1, until F=n-1. 5. In this case F becomes equal to R (F==R), then array [F] is set to zero   (array [F] = 0).   1. Now, queue is totally empty. 2. For the next dequeue operation, (array [F] ==0 && F==R) is true and underflow occurs.   In summary, when F and R become equal and array[F] is zero simultaneously, underflow detects. |

|  |
| --- |
| Question No. 11 |
| Show the status of a QUEUE for the following operations, where the QUEUE is implemented by an array of size, m=3. Here, Enqueue and Dequeue mean insert and delete respectively, and x=9, y=x+3, z=x+y, and p=y+z.  Enqueue(z), Enqueue(p), Dequeue(), Enqueue(y), Enqueue(z) |
| Answer:  Here, x=9, y=9+3=12, z=y+x=9+12=21, p=y+z=12+21=33  Steps:  (we know the queue follows FIFO and Here F=Front pointer, R=Rear pointer)   |  |  |  | | --- | --- | --- | | Enqueue(z)=Enqueue(21) | F=0, R=0 | array[0]=21 | | Enqueue(p)=Enqueue(33) | F=0, R=1 | array[0]=21  array[1]=33 | | Dequeue() | F=1, R=1 | array[1]=33 | | Enqueue(y)=Enqueue(12) | F=1, R=2 | array[1]=33  array[2]=12 | | Enqueue(z)=Enqueue(21) | F=1, R=0 | array[0]=21  array[1]=33  array[2]=12 | |

|  |
| --- |
| Question No. 12 |
| Generate a pseudocode for solving the following problems within a time complexity of O(n^2)  Delete all of the consecutive elements from a Linear Linked List whose sum is equal to (Zero). **10**   |  |  |  | | --- | --- | --- | | Input | Output | Explanation | | 8  6 -6 8 4 -12 9 8 -8 | 9 | 6-6 = 0  8+4-12 =0  8-8 =0  Thus, all of these numbers from the list is eliminated | | 11  4 6 -10 8 9 10 -19 10 -18 20 25 | 20->25 | 4+6-10=0  8+10-18=0  9+10 = -19  Thus, all of these numbers from the list is eliminated | |
| Answer:  Psudocode:   1. Input values. 2. If the value is positive, pushed into a stack. 3. If the value is negative, sum with the top value and check whether the sum is zero or not. Pop values from the stack until the sum is zero or until the stack is empty. 4. After this operation, if there is any remaining value that is the answer.   Details steps:   1. At first initialize a Stack of integer type as “st”. 2. Initialize two int-type variables for the number of elements and value. 3. Enter n values using a loop. 4. Every time a value is being input, zeroSum function is called. 5. In the zeroSum function two things are happening 6. If the input value is positive, it is pushed into the stack. 7. If the value is negative, the value is considered as a new variable temp. 8. While temp is not zero, temp is summed with the top value and temp is updated. Then the stack is popped until stack is empty. 9. Each time temp is checked whether it is zero or not, if temp becomes zero then break and return. 10. This cycle repeats until n values are inserted. 11. Then if there is any remaining value in the stack, they are displayed and popped. But these values are in reverse, so they need to be reversed before popped. 12. For this, reverseStack function is called. 13. This function recursively popped every element from the stack and   Called another function named insertAtBottom.   1. In the insertAtBottom function, first check whether the stack is empty or not. 2. If empty, then, value from reverseStack is pushed into the stack. 3. Else, values in the stack are popped until stack is empty, and then pushed values from reverseStack function. 4. This way, stack elements are reversed (if any). 5. Then after reversing stack values are the answers.   Code:  // using STL  #include <bits/stdc++.h>  using namespace std;  stack<int> st;  void zeroSum(int val)  {  if (val > 0)  {  st.push(val);  }  else//if nagetive  {  int temp = val;  while (temp != 0)  {  // cout<<temp<<" "<<st.top()<<" \*\*1";  temp = st.top() + temp;  // cout<<" after minus: "<<temp<<endl;  // st.pop();  if (!st.empty())  {  st.pop();  }  if (temp == 0)  {  break;  }  }  }  }  void insertAtBottom(int x)  {  if (st.size() == 0)  {  st.push(x);  }  else  {  int y = st.top();  st.pop();  insertAtBottom(x);  st.push(y);  }  }  void reverseStack()  {  if (st.size() > 0)  {  int x = st.top();  st.pop();  reverseStack();  insertAtBottom(x);  }  return;  }  int main()  {  int n = 11, val;  for (int i = 0; i < n; i++)  {  cin >> val;  // st.push(val);  zeroSum(val);  }  cout << endl;  // reverse  reverseStack();  while (!st.empty())  {  cout << st.top() << " ";  st.pop();  }  } |

Submitted by: Subrata Saha

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

Date: 27.08.2022

THE END