Lab 9

|  |  |
| --- | --- |
| Function | Big O |
|  |  |
| struct student  {  char rollNo[20];  char emailId[35];  char lecture[3];  char tutorial[3];  char practical[3];  struct student\*left;  struct student\*right;  }; | O(1)[Declaration]  **O(1)** |
| struct student\*createNode(struct student buffer)  {  struct student\*ptr=(struct student\*)malloc(sizeof(struct student));  strcpy(ptr->rollNo,buffer.rollNo);  strcpy(ptr->emailId,buffer.emailId);  strcpy(ptr->lecture,buffer.lecture);  strcpy(ptr->tutorial,buffer.tutorial);  strcpy(ptr->practical,buffer.practical);  ptr->left=NULL;  ptr->right=NULL;  return ptr;  } | O(1)[memory allocation]    O(1)[strcpy function]  O(1)[making ptr next to null]  O(1)[return stmt]  **O(1)** |
| int insertNode(struct student\*\*startPtr,struct student\*temp)  {  struct student\* a = \*startPtr;  if(\*startPtr==NULL) {  \*startPtr = temp;  return 1;  }  struct student\* st = \*startPtr;  struct student\* pr;  while(st!=NULL) {  if(strcmp(st->rollNo,temp->rollNo)==0)  return 0;  if(strcmp(st->rollNo,temp->rollNo)>0) {  pr = st;  st=st->left;  }  else if(strcmp(st->rollNo,temp->rollNo)<0) {  pr = st;  st=st->right;  }  }  if(strcmp(pr->rollNo,temp->rollNo)>0)  pr->left=temp;  else  pr->right = temp;  return 1;  } | O(h)[while loop will traverse through the tree where h is height of BST, in worst case h=n]  **O(n)** |
| int deleteNode(struct student\*\*ptr,char rollNumber[20])  {  struct student\* head = \*ptr;  if(strcmp(head->rollNo,rollNumber)==0) {  struct student\* hr = head->right;  while(hr->left!=NULL) {  hr = hr->left;  }  hr->left = head->left;  struct student\* temp = \*ptr;  \*ptr = head->right;  free(temp);  return 1;  }  struct student\* st = \*ptr;  struct student\* pr;  while(st!=NULL) {  if(strcmp(st->rollNo,rollNumber)==0)  break;  if(strcmp(st->rollNo,rollNumber)>0) {  pr = st;  st=st->left;  }  else {  pr = st;  st=st->right;  }  }  if(st==NULL)  return 0;  //printf("%s %s %s",st->rollNo,st->right,st->left->rollNo);  if(st->right==NULL && st->left==NULL) {  if(pr->right==st) pr->right=NULL;  else if(pr->left==st) pr->left=NULL;  free(st);  return 1;  }  if(st->right==NULL && st->left!=NULL) {  if(pr->right==st) pr->right=st->left;  else if(pr->left==st) pr->left=st->left;  free(st);  return 1;  }  if(st->right!=NULL && st->left==NULL) {  if(pr->right==st) pr->right=st->right;  else if(pr->left==st) pr->left=st->right;  free(st);  return 1;  }  if(st->right!=NULL && st->left!=NULL) {  struct student\* pred = st->left;  struct student\* parent = st;  while(pred->right!=NULL) {  parent=pred;  pred=pred->right;  }  if(pred==parent->right)  parent->right=pred->left;  else  parent->left=pred->left;  if(pr->left==st)  pr->left = pred;  pred->left = st->left;  pred->right = st->right;  free(st);  return 1;  }  } | O(n)[ The worst case time complexity of delete operation is O(h) where h is the height of the Binary Search Tree. In worst case, we may have to travel from the root to the deepest leaf node. The height of a skewed tree may become n and the time complexity of delete operation may become O(n)]  **O(n)** |
| int searchNode(struct student\*ptr,char rollNumber[20])  {  int i=1;  if(ptr==NULL)  return 0;  while(ptr!=NULL) {  if(strcmp(ptr->rollNo,rollNumber)==0)  return i;  if(strcmp(ptr->rollNo,rollNumber)>0)  ptr = ptr->left;  else  ptr = ptr->right;  i++;  }  } | O(n)[while loop may traverse till the end of BST]  **O(n)** |
| int displayBST(struct student\*start)  {  if(start==NULL)  return 0;  displayBST(start->left);  printf("%s %s\n",start->rollNo,start->emailId);  displayBST(start->right);  return ++tot;  } | O(n)[as it visits every node]  **O(n)** |







