

Programme Name: \_\_\_\_\_\_\_\_**BCS HONS**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Course Code: \_\_**CSC 2516**\_\_\_\_\_\_\_\_

Course Name: \_\_\_\_\_\_\_**Data Structure and Algorithm**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Internal Examination**

Date of Submission: \_\_\_\_\_\_**8/24/2021**\_\_\_\_\_\_\_\_\_\_\_\_\_

**Submitted By: Submitted To:**

Student Name**: Dipesh Tha Shrestha** Faculty Name**: Prakash Chandra Sir**

IUKL ID: **041902900028** Department**: LMS**

Semester**: Fourth Semester**

Intake**: September 2019**

1. **Explain data structures and its importance. Also list at least 5 data structures.** Answer: A data structure is a collection of data elements that provides an efficient method for storing and organizing data in a computer so that it can be used effectively.

Data Structures are important because they allow programmers to work with data efficiently. Data structure provides the right way to organize information in the digital space.

List of 5 data structures are given below:

* + Array
  + List
  + Float
  + Integer
  + Pointers

1. **Explain the working of linear search algorithm and analyze its efficiency Answer:** A linear search algorithm is used to locate a target value within a list. It checks each element of the list for the target value in a sequential manner until a match is found or all elements have been searched.

In linear search, we search for an element or value in an array by traversing it from the beginning until the element or value we want is found. It compares the element to be searched with all of the elements in the array and returns the index of the element in the array if the element is successfully matched, otherwise it returns -1.

Best Case Scenario

It's possible that the element you're looking for is in the first position.

In this case, the search is successful after only one comparison.

As a result, the linear search algorithm takes O(1) operations in the best case scenario.

Worst Case Scenario

The element being searched may be at the very end of the array, or it may not exist at all.

In the first case, the search is successful after n comparisons.

With n comparisons, the search ends in failure in the latter case.

As a result, in the worst-case scenario, the linear search algorithm requires O(n) operations.

**Linear Search efficiency**

The efficiency of a method is determined by the amount of time it takes to search a record in a search table or the number of comparisons it makes. Only one comparison is made if the desired record is found in the first position of the search table. When the desired record is the most recent, n comparisons must be performed.

The average number of comparisons will be (n+1/2) if the record appears somewhere in the search table. The worst-case efficiency of this technique is O(n), where n is the execution order.

1. For doubly linked list:

a. Write C program for Insertion, Deletion and Search operation uses in real world applications.

#include <stdio.h>

#include <stdlib.h>

*struct* node

{

*struct* node \*prev;

*int* n;

*struct* node \*next;

}

\*h,\*temp,\*temp1,\*temp2,\*temp4;

*void* insertbeg();

*void* insertend();

*void* insertatpoint();

*void* display();

*void* sort();

*void* search();

*void* update();

*void* delete();

*int* count = 0;

*void* main()

{

*int* ch;

    h = NULL;

    temp = temp1 = NULL;

    printf("\n 1 - Insert at beginning");

    printf("\n 2 - Insert at end");

    printf("\n 3 - Insert at any position");

    printf("\n 4 - Delete at any position");

    printf("\n 5 - Display");

    printf("\n 6 - Search for element");

    printf("\n 7 - Sort the Elements");

    printf("\n 8 - Update an element");

    printf("\n 9 - Exit");

    while (1)

{

    printf("\n Enter choice : ");

    scanf("%d", &ch);

    switch (ch)

    {

        case 1:

        insertbeg();

        break;

        case 2:

        insertend();

        break;

        case 3:

        insertatpoint();

        break;

        case 4:

        delete();

        break;

        case 5:

        display();

        break;

        case 6:

        search();

        break;

        case 7:

        sort();

        break;

        case 8:

        update();

        break;

        case 9:

        exit(0);

        default:

        printf("\n Wrong choice menu");

        }

    }

}

*void* create()

{

*int* data;

    temp =(*struct* node \*)malloc(1\*sizeof(*struct* node));

    temp->prev = NULL;

    temp->next = NULL;

    printf("\n Enter value to node : ");

    scanf("%d", &data);

    temp->n = data;

    count++;

}

*void* insertbeg()

{

    if (h == NULL)

{

    create();

    h = temp;

    temp1 = h;

}

else

{

    create();

    temp->next = h;

    h->prev = temp;

    h = temp;

}

}

*void* insertend()

{

if (h == NULL)

{

    create();

    h = temp;

    temp1 = h;

}

else

{

    create();

    temp1->next = temp;

    temp->prev = temp1;

    temp1 = temp;

}

}

*void* insertatpoint()

{

*int* pos, i = 2;

    printf("\n Enter position to be inserted : ");

    scanf("%d", &pos);

    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))

{

printf("\n Position out of range to insert");

return;

}

if ((h == NULL) && (pos != 1))

{

printf("\n Empty list cannot insert other than 1st position");

return;

}

if ((h == NULL) && (pos == 1))

{

    create();

    h = temp;

    temp1 = h;

    return;

}

else

{

    while (i < pos)

{

    temp2 = temp2->next;

    i++;

}

create();

temp->prev = temp2;

temp->next = temp2->next;

temp2->next->prev = temp;

temp2->next = temp;

}

}

*void* delete()

{

*int* i = 1, pos;

    printf("\n Enter position to be deleted : ");

    scanf("%d", &pos);

    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))

{

    printf("\n Error : Position out of range to delete");

    return;

}

    if (h == NULL)

{

    printf("\n Error : Empty list no elements to delete");

    return;

}

else

{

    while (i < pos)

{

    temp2 = temp2->next;

    i++;

}

if (i == 1)

{

if (temp2->next == NULL)

{

    printf("Node deleted from list");

    free(temp2);

    temp2 = h = NULL;

    return;

}

}

if (temp2->next == NULL)

{

temp2->prev->next = NULL;

free(temp2);

printf("Node deleted from list");

return;

}

temp2->next->prev = temp2->prev;

if (i != 1)

temp2->prev->next = temp2->next; /\* Might not need this statement if i == 1 check \*/

if (i == 1)

h = temp2->next;

printf("\n Node deleted");

free(temp2);

}

count--;

}

*void* display()

{

temp2 = h;

if (temp2 == NULL)

{

printf("List empty to display \n");

return;

}

printf("\n Elements are: ");

while (temp2->next != NULL)

{

printf(" %d ", temp2->n);

temp2 = temp2->next;

}

printf(" %d ", temp2->n);

}

*void* search()

{

*int* data, count = 0;

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty to search for data");

return;

}

printf("\n Enter value to search : ");

scanf("%d", &data);

while (temp2 != NULL)

{

if (temp2->n == data)

{

printf("\n Data found in %d position",count + 1);

return;

}

else

temp2 = temp2->next;

count++;

}

printf("\n Error : %d not found in list", data);

}

*void* update()

{

*int* data, data1;

printf("\n Enter node data to be updated : ");

scanf("%d", &data);

printf("\n Enter new data : ");

scanf("%d", &data1);

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty no node to update");

return;

}

while (temp2 != NULL)

{

if (temp2->n == data)

{

temp2->n = data1;

display();

return;

}

else

temp2 = temp2->next;

}

printf("\n Error : %d not found in list to update", data);

}

*void* sort()

{

*int* i, j, x;

temp2 = h;

temp4 = h;

if (temp2 == NULL)

{

printf("\n List empty to sort");

return;

}

    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)

{

    for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)

{

    if (temp2->n > temp4->n)

{

    x = temp2->n;

    temp2->n = temp4->n;

    temp4->n = x;

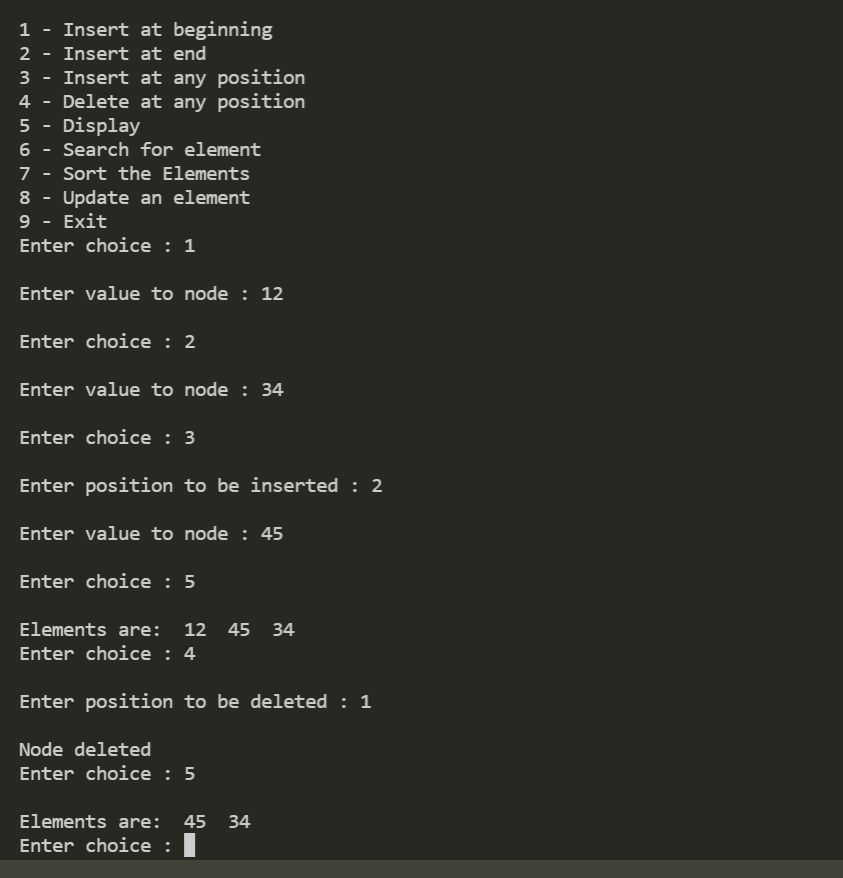
}

}

}

display();

}



1. **For the given list of numbers:**

**[ 15, 8 , 25 , 40 , 7 , 9 , 10 ]**

**Create binary search tree.**

