DS Experiment - 8

Harshit Saraswat | 500097101 | B42

**Question 1:** Store ‘n’ numbers in an array in an unsorted order. Using menu-driven logic perform a search (Linear and Binary Search algorithms) for a given number and report success or failure in the form of a suitable message. (Hint: Before performing a Binary search the array should be sorted using the Bubble sort algorithm.)

**Code:**

#include <stdio.h>

#include <stdlib.h>

int \* my\_array = NULL;

int count = 0;

int size = 0;

int main()

{

printf("Enter Size of Array : ");

scanf("%d", &size);

my\_array = (int \*)malloc(size\*sizeof(int));

while(1){

int choice = 0;

printf("1 - Append To Array\n2 - Linear Search\n3 - Binary Search\n4 - Print Array\n5 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : append\_to\_array();break;

case 2 : linear\_search();break;

case 3 : binary\_search();break;

case 4 : print\_array();break;

case 5 : exit(0);

}

printf("\n");

}

return 0;

}

void append\_to\_array(){

if (count == size){

printf("Array is Full..\n");

return;

}

int data = 0;

printf("Enter Number to Append : ");

scanf("%d",&data);

my\_array[count] = data;

count++;

}

void linear\_search(){

int element = 0;

printf("Enter Element To Find : ");

scanf("%d",&element);

int flag = 0;

for (int i = 0; i < count; i++){

if (my\_array[i] == element){

printf("%d is at Index %d",element,i);

flag = 1;

break;

}

}

if (flag == 0){

printf("Element not in Array... Exiting...\n");

}

}

void binary\_search(){

bubble\_sort();

int element;

printf("Enter Element To Find : ");

scanf("%d",&element);

int low = 0;

int high = count - 1;

if (my\_array[high] == element){

printf("Element %d is at Index %d",element, high);

return;

}

if (my\_array[low] == element){

printf("Element %d is at Index %d",element, low);

return;

}

int mid = (high + low)/2;

while (my\_array[mid] != element){

if (low == high - 1){

if (my\_array[high] == element){

printf("Element %d is at Index %d",element, high);

return;

}

if (my\_array[low] == element){

printf("Element %d is at Index %d",element, low);

return;

}

printf("Element NOT in Array...\n");

return;

}

if (element > my\_array[mid]){

low = mid;

mid = (high + low)/2;

continue;

}

if (element < my\_array[mid]){

high = mid;

mid = (high + low)/2;

continue;

}

}

printf("Element %d is at Index %d",element, mid);

}

void bubble\_sort(){

int flag = 1;

while(flag == 1){

flag = 0;

for (int i = 0 ; i+1 < count ;i++){

if (my\_array[i] > my\_array[i+1]){

flag = 1;

swap(i,i+1);

}

}

}

}

void print\_array(){

printf("\n");

for (int i = 0; i < count; i++){

printf("| %d |",my\_array[i]);

}

printf("\n");

}

void swap(int i, int j){

int temp = my\_array[i];

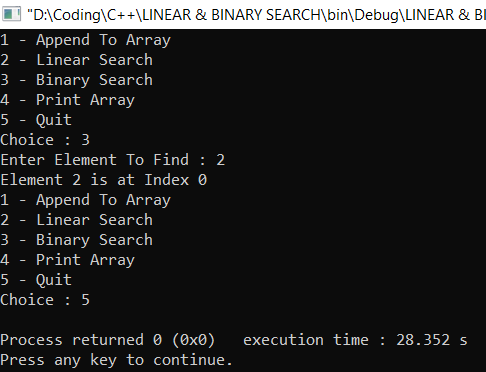
my\_array[i] = my\_array[j];

my\_array[j] = temp;

}

**Output:**



****

**Question 2:** Store ‘n’ numbers in a Linked list. Perform a Linear search for a given number and report success or failure in the form of a suitable message.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int value;

struct node \* next;

};

struct node \* first = NULL;

struct node \* ptr = NULL;

struct node \* new\_ = NULL;

void display\_list();

void insert\_node\_at\_end();

struct node \* new\_node();

void insert\_node\_at\_begin();

void linear\_search();

int main(){

int repeat = 1;

while (repeat == 1){

insert\_node\_at\_end();

printf("Press 1 To Enter Another Node : ");

scanf("%d",&repeat);

printf("\n");

}

display\_list();

linear\_search();

return 0;

}

void linear\_search(){

int element = 0;

printf("\nEnter Element TO Search : ");

scanf("%d",&element);

int flag = 0;

for(ptr = first; ptr!=NULL; ptr = ptr->next){

if (ptr->value == element){

flag = 1;

break;

}

}

if (flag == 0){

printf("Element not in List\n");

return;

}

printf("Address of Element %d is %x",element,ptr);

}

struct node \* new\_node(){

new\_ = (struct node \*)malloc(sizeof(struct node));

printf("Enter Value Of Node : ");

scanf("%d",&new\_->value);

new\_->next = NULL;

return new\_;

}

void insert\_node\_at\_end(){

if (first == NULL){

first = new\_node();

return;

}

for (ptr = first; ptr->next!=NULL; ptr = ptr->next);

ptr->next = new\_node();

}

void display\_list(){

for (ptr = first; ptr!= NULL; ptr = ptr->next){

printf("| %d |",ptr->value);

}

}

void insert\_node\_at\_begin(){

new\_ = new\_node();

new\_->next = first;

first = new\_;

}

void insert\_node\_at\_position(){

int position = 0;

printf("Position of Node : ");

scanf("%d",&position);

ptr = first;

for (int i = 0; i < position - 2; ptr = ptr->next,i++);

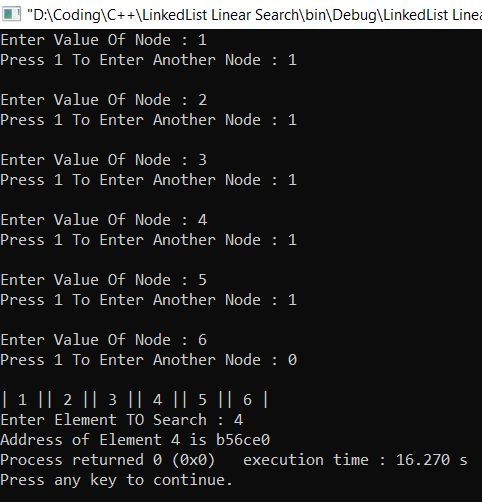
new\_ = new\_node();

new\_->next = ptr->next;

ptr->next = new\_;

}

**Output:**



**Question 3:** Store ‘n’ numbers in a Hash Table implemented with an Array of Linked Lists. Perform a search for a given number and report success or failure in the form of a suitable message.

**Code:**

#include <stdio.h>

#include <stdlib.h>

int \* hash\_array;

int count = 0;

int size;

int hash\_function(int hash\_key);

int add\_till\_death(int input);

int main()

{

printf("\t\t\t\t\tDSA Experiment - 8.3");

size = 10;

hash\_array = (int \*)malloc(size\*sizeof(int));

for(int i = 0;i < size;i++){

hash\_array[i] = NULL;

}

int choice = 0;

while (1 == 1){

printf("\n1 - To Enter Sap ID\n2 - To Delete SAP\_ID\n3 - Find Hash Key\n4 - Quit\nChoice : ");

scanf("%d",&choice);

switch(choice){

case 1 : insert\_key\_to\_table();break;

case 2 : delete\_key();break;

case 3 : search\_key();break;

case 4 : exit(0);

}

printf("\n");

}

return 0;

}

int hash\_function(int hash\_key){ //RETURN THE ARRAY INDEX FOR THE GIVEN KEY - IN THIS CASE SAP ID

hash\_key = hash\_key - 5000000;

return add\_till\_death(hash\_key);

}

void insert\_key\_to\_table(){ //FRONT END PRETTY FUNCTION

int data = 0;

printf("Enter SAP\_ID To insert : ");

scanf("%d",&data);

insert\_to\_array\_hash\_table(data,hash\_function(data));

}

int insert\_to\_array\_hash\_table(int data,int key){ //TECHNICAL FUNCTION BACKEND

if (hash\_array[key] == NULL){

hash\_array[key] = data;

return;

}

//LINEAR PROBING

int i = 0;

for(i = (key + 1) % size; hash\_array[i]!=NULL && i!=key ; i++,i = i % size); //KEEP GOING TO INDEX THAT IS NULL EVEN AFTER END OF ARRAY

if (i == key){

printf("\nNo Available Space Array is FULL...");

return;

}

hash\_array[i] = data;

}

int add\_till\_death(int input){

int total = 0;

while (input != 0){

total = total + input % 10;

input = input/10;

}

if (total > 9)

total = add\_till\_death(total);

return total;

}

void search\_key(){ //SEARCHING IN ARRAY OF LINKED LIST

int data;

int key\_index;

printf("Enter Entry to Find : ");

scanf("%d", &data);

key\_index = hash\_function(data);

if (hash\_array[key\_index] == data){ //TO MAKE THE FOR LOOP CONDITION USEABLE

printf("\n%d is Present at Index : %d",data,key\_index);

return;

}

int i = key\_index+1;

for (i ; hash\_array[i] != data && i!=key\_index ; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table");

return;

}

printf("\n%d is Present at Index : %d",data,i);

}

void delete\_key(){

int data;

printf("\nEnter Key to Delete : ");

scanf("%d",&data);

int key\_index = hash\_function(data);

int i = key\_index + 1;

if (hash\_array[key\_index] == data){

printf("\nDeleted Key : %d",hash\_array[key\_index]);

hash\_array[key\_index] = NULL;

return;

}

for (i ; hash\_array[i] != data && i!= key\_index; i++,i = i % size);

if (i == key\_index){

printf("\nKey not in Table\n");

return;

}

printf("\nDeleted Key : %d",hash\_array[i]);

hash\_array[i] = NULL;

}

**Output:**

