**Practical No. 6**

1. Heap Tree

Code :

#include<iostream>

using namespace std;

class heapTree{

int arr[100];

int length;

public:

heapTree(int l)

{

length=l;

for(int i=0;i<100;i++) // Initializing all elements to a random initial value

{

arr[i]=-2431;

}

}

bool sorted=false;

int parent;

int parent\_loc=0; // Initial parent location is 0

int parent\_loc\_dup=0; /\*Creating a parent duplicate node. This is used for setting parent back to original node when

\*Parents are changed while replacing the elements during insertion\*/

void input()

{

for(int i=0;i<length;i++) // Running till all elements are inserted

{

if(i==0)

{

cout<<"Enter value "; /\*If element is first element, then directly take it and move to next iteration

\*as first element has no parent

\*/

cin>>arr[i];

continue;

}

if(((2\*parent\_loc)+1<=length)&&((2\*parent\_loc+2)<=length) && (arr[(2\*parent\_loc)+1]!=-2431 && arr[(2\*parent\_loc)+2]!=-2431))

{

/\*This if condition is for changing parent.

\*Parents are changed when both the child locations are empty, and child location positions

\*dont exceed the maximum array length

\*/

parent=arr[i];

parent\_loc++; //new parent location will always be a +1 to the original location, i.e. next location always become parent

parent\_loc\_dup++;; //Parent dupe is made to copy parent, as it is a duplicate of parent

cout<<"Parent = "<<arr[parent\_loc]<<endl;

}

if((arr[(2\*parent\_loc)+1])==-2431) //If left child of a parent is empty, then this condition

{

cout<<"Enter value ";

cin>>arr[(2\*parent\_loc)+1]; //Taking into left child position, the new value

int x=(2\*parent\_loc)+1; //X variable stores the position of the child. Will be used later in the following while loop

while(true)

{

cout<<arr[x]<<" (>) "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;//A debugging statement.

if(arr[x]>arr[parent\_loc]) //If value stored in child node is greater than that of the parent node.

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x]; //Swap the locations of parent and child node.

arr[x]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl; //Simple debug statement

x=parent\_loc; /\*the new value of x is now the parent location, as the original child value is now on

\*parent's place and is the one that needs to be compared with the upper parent now

\*/

if(parent\_loc%2!=0)

{

/\* Finding the new parent position. New parnet position, in case the previous parent position is

\*an odd number is parent location-1/2. Since child position is, position=(2i+1), then i=(position-1)/2

\*An odd number always represents left child.

\*/

parent\_loc=(parent\_loc-1)/2;

}

else

{

/\* Finding the new parent position. New parnet position, in case the previous parent position is

\*an even number is parent location-2/2. Since child position is, position=(2i+2), then i=(position-1)/2

\*An even number always represents left child.

\*/

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

} //Loop is repeated till the child < parent (max heap)

else

{

break; //loop is broken if parent > child

}

}

parent\_loc=parent\_loc\_dup; //Parent is reset using duplicate variable

}

else if((arr[2\*(parent\_loc)+2])==-2431) //All comments are same as above. This is just a repeated part, but for the right child.

{

cout<<"Enter value ";

cin>>arr[(2\*parent\_loc)+2];

int x=(2\*parent\_loc)+2;

while(true)

{

cout<<arr[x]<<" > "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]>arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

}//end of for loop

cout<<"After insertion ";

for(int i=0;i<length;i++) //Just for displaying the final array values after insertion

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<endl;

}

void sort()

{

bool sorted=false;

int arr\_dup[length]; /\* A duplicate array is taken with same length as our original array. This is taken because this array

\*serves as input for original array in sorting. As sorting involves same procedure by repeatedly inserting elements

\*and after insertions are complete, replacing first and last element of array and taken out the last element, the entire

\*previous code is copied, with only parts that are removed are debug statements. Just additional step is

\*performed at the end where the first and last positions are swapped and last position is taken out

\*Also array duplicate is replaced again by the new array at the end.

\* see this below.

\*/

for(int i=0;i<length;i++)

{

arr\_dup[i]=arr[i]; //Initializing duplicate array with inserted array values

}

for(int i=0;i<100;i++)

{

arr[i]=-2431;//Resetting initial array.

}

while(!sorted) //loop is repeated till the sorted flag is set, which is set when length of new array reaches 0.

{

parent\_loc=0; //parent locations are reset.

parent\_loc\_dup=0; //Parent duplicate is reset as well.

for(int i=0;i<length;i++) // same conditions as above

{

if(i==0)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[i]=arr\_dup[i];

//cout<<endl;

continue;

}

if(((2\*parent\_loc)+1<=length)&&((2\*parent\_loc+2)<=length) && (arr[(2\*parent\_loc)+1]!=-2431 && arr[(2\*parent\_loc)+2]!=-2431))

{

parent=arr[i];

parent\_loc++;

parent\_loc\_dup++;;

//cout<<"Parent = "<<arr[parent\_loc]<<endl;

}

if((arr[(2\*parent\_loc)+1])==-2431)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[(2\*parent\_loc)+1]=arr\_dup[i];

//cout<<endl;

int x=(2\*parent\_loc)+1;

while(true)

{

//cout<<arr[x]<<" (>) "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]>arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

//cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

/\*if(arr[x]>arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[(2\*parent\_loc)+1];

arr[(2\*parent\_loc)+1]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

parent\_loc=((parent\_loc-1)/2);

}\*/

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

else if((arr[2\*(parent\_loc)+2])==-2431)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[(2\*parent\_loc)+2]=arr\_dup[i];

//cout<<endl;

int x=(2\*parent\_loc)+2;

while(true)

{

//cout<<arr[x]<<" > "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]>arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

//cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

} //end of for loop

int temp;

temp=arr[0];

arr[0]=arr[length-1]; //Swapping first and last element and then displaying the last element.

arr[length-1]=temp;

cout<<arr[length-1]<<" ";

length--; //REducing length by 1 as the last element is now removed from the array and from further processing.

for(int i=0;i<length;i++)

{

arr\_dup[i]=arr[i]; //Duplicate array is now the newly inserted array with one less location. it now again serves as input.

}

for(int i=0;i<100;i++)

{

arr[i]=-2431; //NEwly inserted array is now reset again.

}

if(length==0) //Sorted flag is set and entire loop is now broken once the length reaches 0.

{

sorted=true;

}

}

}

void mininput()

{

for(int i=0;i<length;i++) // Running till all elements are inserted

{

if(i==0)

{

cout<<"Enter value "; /\*If element is first element, then directly take it and move to next iteration

\*as first element has no parent

\*/

cin>>arr[i];

continue;

}

if(((2\*parent\_loc)+1<=length)&&((2\*parent\_loc+2)<=length) && (arr[(2\*parent\_loc)+1]!=-2431 && arr[(2\*parent\_loc)+2]!=-2431))

{

/\*This if condition is for changing parent.

\*Parents are changed when both the child locations are empty, and child location positions

\*dont exceed the maximum array length

\*/

parent=arr[i];

parent\_loc++; //new parent location will always be a +1 to the original location, i.e. next location always become parent

parent\_loc\_dup++;; //Parent dupe is made to copy parent, as it is a duplicate of parent

cout<<"Parent = "<<arr[parent\_loc]<<endl;

}

if((arr[(2\*parent\_loc)+1])==-2431) //If left child of a parent is empty, then this condition

{

cout<<"Enter value ";

cin>>arr[(2\*parent\_loc)+1]; //Taking into left child position, the new value

int x=(2\*parent\_loc)+1; //X variable stores the position of the child. Will be used later in the following while loop

while(true)

{

cout<<arr[x]<<" (>) "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;//A debugging statement.

if(arr[x]<arr[parent\_loc]) //If value stored in child node is less than that of the parent node.

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x]; //Swap the locations of parent and child node.

arr[x]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl; //Simple debug statement

x=parent\_loc; /\*the new value of x is now the parent location, as the original child value is now on

\*parent's place and is the one that needs to be compared with the upper parent now

\*/

if(parent\_loc%2!=0)

{

/\* Finding the new parent position. New parnet position, in case the previous parent position is

\*an odd number is parent location-1/2. Since child position is, position=(2i+1), then i=(position-1)/2

\*An odd number always represents left child.

\*/

parent\_loc=(parent\_loc-1)/2;

}

else

{

/\* Finding the new parent position. New parnet position, in case the previous parent position is

\*an even number is parent location-2/2. Since child position is, position=(2i+2), then i=(position-1)/2

\*An even number always represents left child.

\*/

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

} //Loop is repeated till the child < parent (max heap)

else

{

break; //loop is broken if parent > child

}

}

parent\_loc=parent\_loc\_dup; //Parent is reset using duplicate variable

}

else if((arr[2\*(parent\_loc)+2])==-2431) //All comments are same as above. This is just a repeated part, but for the right child.

{

cout<<"Enter value ";

cin>>arr[(2\*parent\_loc)+2];

int x=(2\*parent\_loc)+2;

while(true)

{

cout<<arr[x]<<" > "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]<arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

}//end of for loop

cout<<"After insertion ";

for(int i=0;i<length;i++) //Just for displaying the final array values after insertion

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<endl;

}

void minsort()

{

bool sorted=false;

int arr\_dup[length]; /\* A duplicate array is taken with same length as our original array. This is taken because this array

\*serves as input for original array in sorting. As sorting involves same procedure by repeatedly inserting elements

\*and after insertions are complete, replacing first and last element of array and taken out the last element, the entire

\*previous code is copied, with only parts that are removed are debug statements. Just additional step is

\*performed at the end where the first and last positions are swapped and last position is taken out

\*Also array duplicate is replaced again by the new array at the end.

\* see this below.

\*/

for(int i=0;i<length;i++)

{

arr\_dup[i]=arr[i]; //Initializing duplicate array with inserted array values

}

for(int i=0;i<100;i++)

{

arr[i]=-2431;//Resetting initial array.

}

while(!sorted) //loop is repeated till the sorted flag is set, which is set when length of new array reaches 0.

{

parent\_loc=0; //parent locations are reset.

parent\_loc\_dup=0; //Parent duplicate is reset as well.

for(int i=0;i<length;i++) // same conditions as above

{

if(i==0)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[i]=arr\_dup[i];

//cout<<endl;

continue;

}

if(((2\*parent\_loc)+1<=length)&&((2\*parent\_loc+2)<=length) && (arr[(2\*parent\_loc)+1]!=-2431 && arr[(2\*parent\_loc)+2]!=-2431))

{

parent=arr[i];

parent\_loc++;

parent\_loc\_dup++;;

//cout<<"Parent = "<<arr[parent\_loc]<<endl;

}

if((arr[(2\*parent\_loc)+1])==-2431)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[(2\*parent\_loc)+1]=arr\_dup[i];

//cout<<endl;

int x=(2\*parent\_loc)+1;

while(true)

{

//cout<<arr[x]<<" (>) "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]<arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

//cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

/\*if(arr[x]>arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[(2\*parent\_loc)+1];

arr[(2\*parent\_loc)+1]=temp;

cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

parent\_loc=((parent\_loc-1)/2);

}\*/

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

else if((arr[2\*(parent\_loc)+2])==-2431)

{

//cout<<"Enter value "<<arr\_dup[i];

arr[(2\*parent\_loc)+2]=arr\_dup[i];

//cout<<endl;

int x=(2\*parent\_loc)+2;

while(true)

{

//cout<<arr[x]<<" > "<<arr[parent\_loc]<<"parent\_loc = "<<parent\_loc<<" ? "<<endl;

if(arr[x]<arr[parent\_loc])

{

int temp;

temp=arr[parent\_loc];

arr[parent\_loc]=arr[x];

arr[x]=temp;

//cout<<"replaced "<<arr[parent\_loc]<<" with "<<temp<<endl;

x=parent\_loc;

if(parent\_loc%2!=0)

{

parent\_loc=(parent\_loc-1)/2;

}

else

{

parent\_loc=(parent\_loc-2)/2;

}

if(parent\_loc<0)

{

break;

}

}

else

{

break;

}

}

parent\_loc=parent\_loc\_dup;

}

} //end of for loop

int temp;

temp=arr[0];

arr[0]=arr[length-1]; //Swapping first and last element and then displaying the last element.

arr[length-1]=temp;

cout<<arr[length-1]<<" ";

length--; //REducing length by 1 as the last element is now removed from the array and from further processing.

for(int i=0;i<length;i++)

{

arr\_dup[i]=arr[i]; //Duplicate array is now the newly inserted array with one less location. it now again serves as input.

}

for(int i=0;i<100;i++)

{

arr[i]=-2431; //NEwly inserted array is now reset again.

}

if(length==0) //Sorted flag is set and entire loop is now broken once the length reaches 0.

{

sorted=true;

}

}

}

};

int main()

{

int size;

cout<<"Enter size ";

cin>>size;

int choice=0;

heapTree ht(size);

while(choice!=-1)

{

cout<<"Enter "<<endl<<"1.Input to max heap"<<endl<<"2.Input to min heap"<<endl<<"3. Max heap sort"<<endl<<"4.Min heap sort"<<endl<<"-1 to exit";

cin>>choice;

if(choice==1)

{

ht.input();

}

if(choice==2)

{

ht.mininput();

}

if(choice==3)

{

ht.sort();

}

if(choice==4)

{

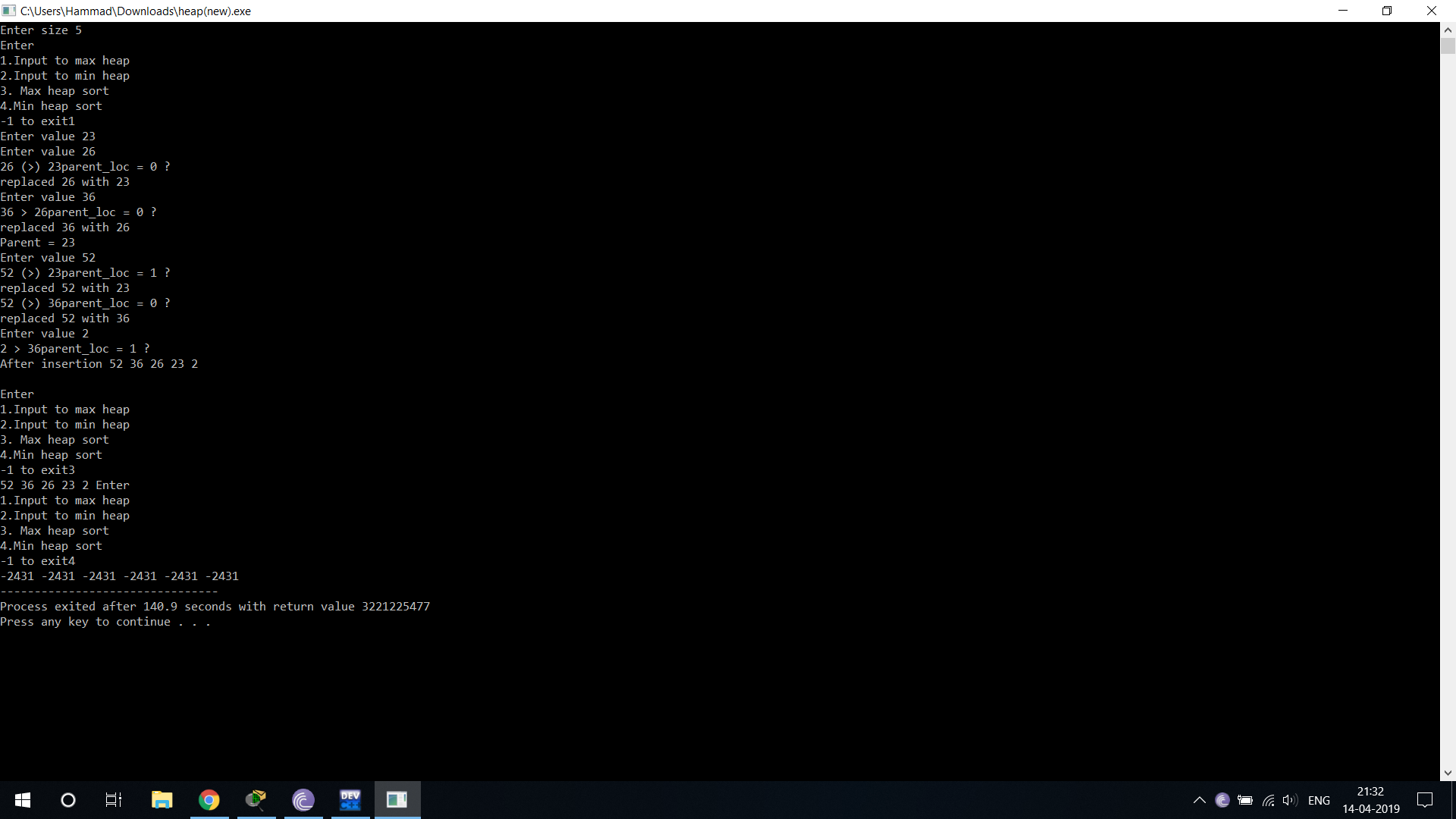
ht.minsort();

}

}

}

Screenshot :



1. B Tree

Code :

#include <stdio.h>

#include <stdlib.h>

#define M 4

struct node {

int n; /\* n < M No. of keys in node will always less than order of B tree \*/

int keys[M-1]; /\*array of keys\*/

struct node \*p[M]; /\* (n+1 pointers will be in use) \*/

}\*root=NULL;

enum KeyStatus { Duplicate,SearchFailure,Success,InsertIt,LessKeys };

void insert(int key);

void display(struct node \*root,int);

enum KeyStatus ins(struct node \*r, int x, int\* y, struct node\*\* u);

int searchPos(int x,int \*key\_arr, int n);

void eatline(void);

int main()

{

int key;

int choice;

printf("Creation of B tree for M=%d\n",M);

while(1)

{

printf("1.Insert\n");

printf("2.Display\n");

printf("3.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice); eatline();

switch(choice)

{

case 1:

printf("Enter the key : ");

scanf("%d",&key); eatline();

insert(key);

break;

case 2:

printf("Btree is :\n");

display(root,0);

break;

case 3:

exit(1);

default:

printf("Wrong choice\n");

break;

}/\*End of switch\*/

}/\*End of while\*/

return 0;

}/\*End of main()\*/

void insert(int key)

{

struct node \*newnode;

int upKey;

enum KeyStatus value;

value = ins(root, key, &upKey, &newnode);

if (value == Duplicate)

printf("Key already available\n");

if (value == InsertIt)

{

struct node \*uproot = root;

root=malloc(sizeof(struct node));

root->n = 1;

root->keys[0] = upKey;

root->p[0] = uproot;

root->p[1] = newnode;

}/\*End of if \*/

}/\*End of insert()\*/

enum KeyStatus ins(struct node \*ptr, int key, int \*upKey,struct node \*\*newnode)

{

struct node \*newPtr, \*lastPtr;

int pos, i, n,splitPos;

int newKey, lastKey;

enum KeyStatus value;

if (ptr == NULL)

{

\*newnode = NULL;

\*upKey = key;

return InsertIt;

}

n = ptr->n;

pos = searchPos(key, ptr->keys, n);

if (pos < n && key == ptr->keys[pos])

return Duplicate;

value = ins(ptr->p[pos], key, &newKey, &newPtr);

if (value != InsertIt)

return value;

/\*If keys in node is less than M-1 where M is order of B tree\*/

if (n < M - 1)

{

pos = searchPos(newKey, ptr->keys, n);

/\*Shifting the key and pointer right for inserting the new key\*/

for (i=n; i>pos; i--)

{

ptr->keys[i] = ptr->keys[i-1];

ptr->p[i+1] = ptr->p[i];

}

/\*Key is inserted at exact location\*/

ptr->keys[pos] = newKey;

ptr->p[pos+1] = newPtr;

++ptr->n; /\*incrementing the number of keys in node\*/

return Success;

}/\*End of if \*/

/\*If keys in nodes are maximum and position of node to be inserted is last\*/

if (pos == M - 1)

{

lastKey = newKey;

lastPtr = newPtr;

}

else /\*If keys in node are maximum and position of node to be inserted is not last\*/

{

lastKey = ptr->keys[M-2];

lastPtr = ptr->p[M-1];

for (i=M-2; i>pos; i--)

{

ptr->keys[i] = ptr->keys[i-1];

ptr->p[i+1] = ptr->p[i];

}

ptr->keys[pos] = newKey;

ptr->p[pos+1] = newPtr;

}

splitPos = (M - 1)/2;

(\*upKey) = ptr->keys[splitPos];

(\*newnode)=malloc(sizeof(struct node));/\*Right node after split\*/

ptr->n = splitPos; /\*No. of keys for left splitted node\*/

(\*newnode)->n = M-1-splitPos;/\*No. of keys for right splitted node\*/

for (i=0; i < (\*newnode)->n; i++)

{

(\*newnode)->p[i] = ptr->p[i + splitPos + 1];

if(i < (\*newnode)->n - 1)

(\*newnode)->keys[i] = ptr->keys[i + splitPos + 1];

else

(\*newnode)->keys[i] = lastKey;

}

(\*newnode)->p[(\*newnode)->n] = lastPtr;

return InsertIt;

}/\*End of ins()\*/

void display(struct node \*ptr, int blanks)

{

if (ptr)

{

int i;

for(i=1; i<=blanks; i++)

printf(" ");

for (i=0; i < ptr->n; i++)

printf("%d ",ptr->keys[i]);

printf("\n");

for (i=0; i <= ptr->n; i++)

display(ptr->p[i], blanks+10);

}/\*End of if\*/

}/\*End of display()\*/

int searchPos(int key, int \*key\_arr, int n)

{

int pos=0;

while (pos < n && key > key\_arr[pos])

pos++;

return pos;

}/\*End of searchPos()\*/

void eatline(void) {

char c;

printf("");

while (c=getchar()!='\n') ;

}

Screenshot :

