

P. S. Deshpande
O. G. Kakde



language course, so we do not explain them here.) Figure shows several representations of an integer number.

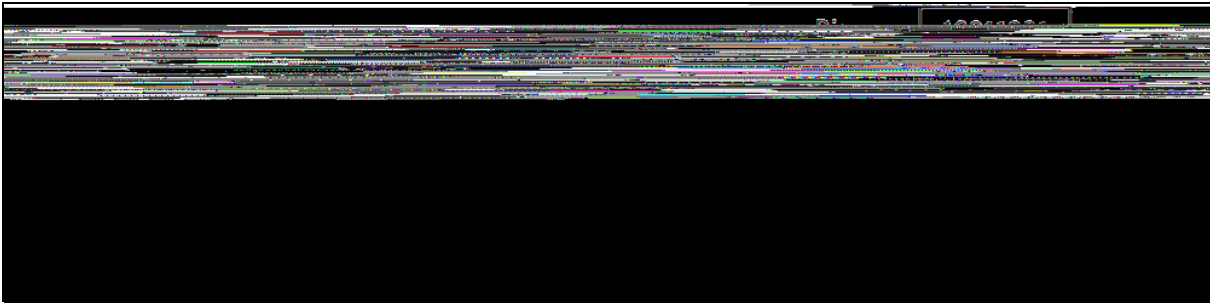
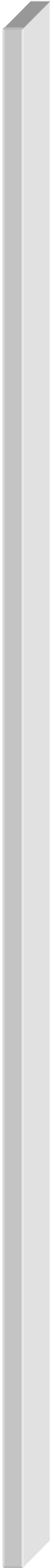


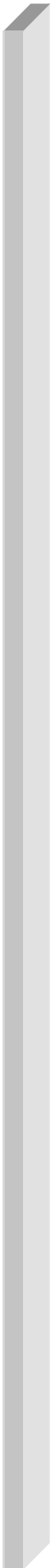
Figure: The decimal equivalents of an 8-bit binary number

The way that integers are physically represented determines how the computer manipulates them. As a C++ programmer, you rarely get involved at this level; instead, you simply use integers. All you need

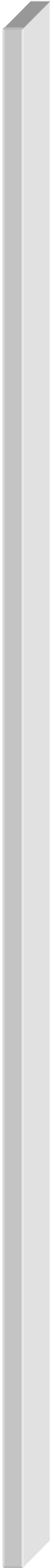
```
cin.gets(buf);  
cin.getf64(that_integ);
```


An observer is an operation that allows us to observe the state of one or more of the data values without changing them. Observers come in several forms:











```
void printarr_usingptr(int *a[])  
{
```

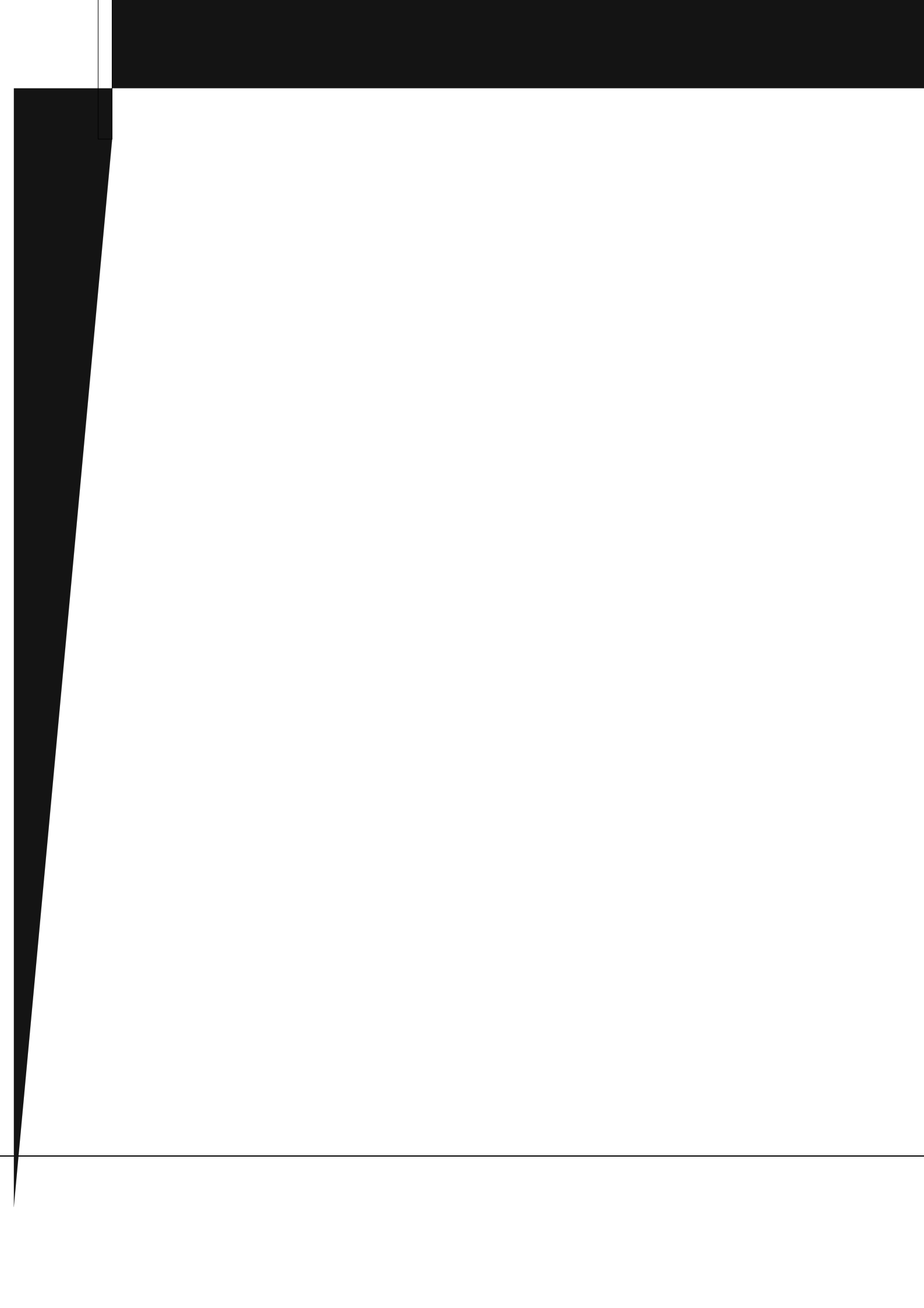


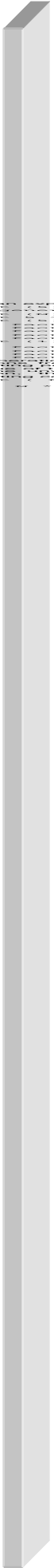

6. *RESOLVING VARIABLE REFERENCES***Introduction**

7. RECURSION

Introduction

You can express most of the problems in the following program by using





The mathematical combinations operation is a good example of a function that can quickly be implemented as a binary recursive function. The number of combinations, often represented as nCk where we are choosing n elements out of a set of k elements, can be implemented as follows:

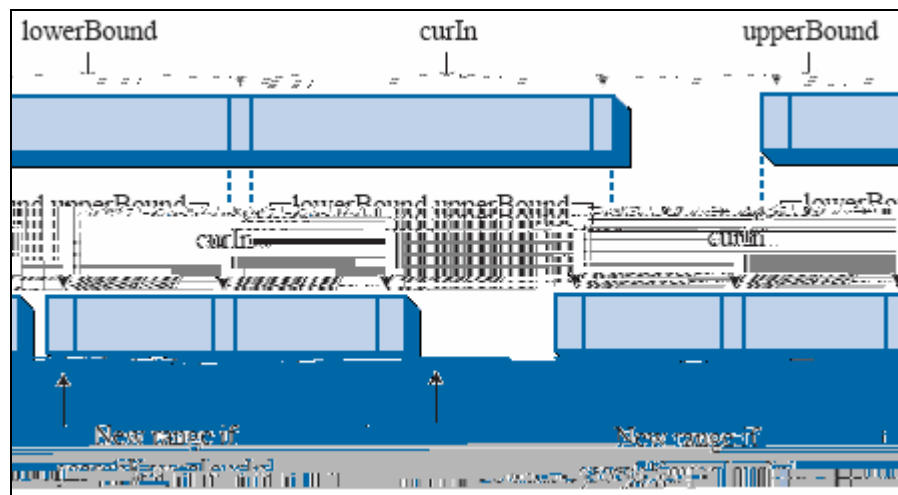
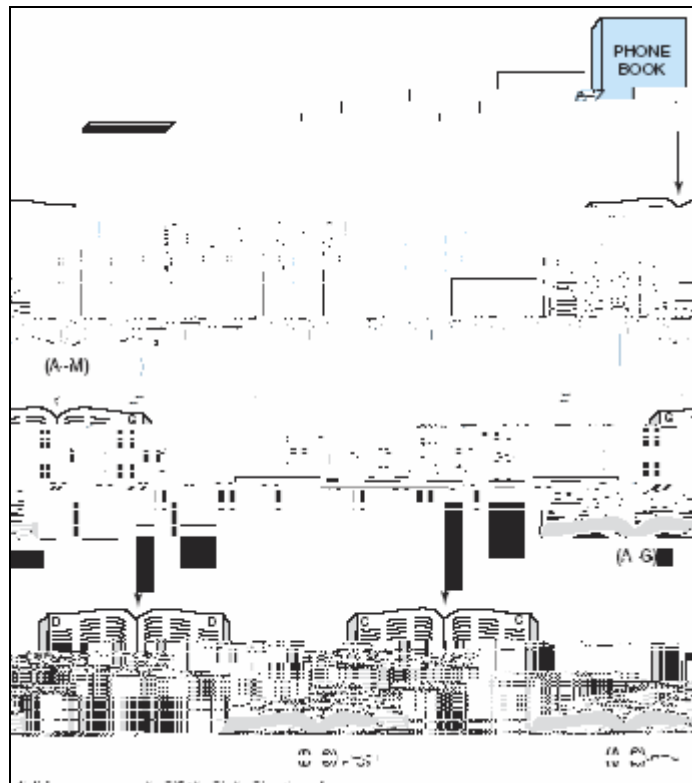
```
int choose(int n, int k)
{
    if (k == 0 || n == k) return(1);
    else return(choose(n-1,k) + choose(n-1,k-1));
}
```

EXPONENTIAL RECURSION

CHAPTER 3: SEARCHING TECHNIQUES

1. *SEARCHING TECHNIQUES: LINEAR OR SEQUENTIAL SEARCH*

than the element in the middle of the list, the search is continued in the upper half of the list. The procedure for the binary search is given in the following program.



Program

```
#include <stdio.h>
#define MAX 10

void bsearch(int list[],int n,int element)
{
    int l,u,m, flag = 0;
    l = 0;
    u = n-1;
    while(l <= u)
    {
```




```
bsearch(list,n,element);
}
```

Example

Input

Enter the number of elements in the list, max = 10

10

Enter the elements

34

2

1

789

99

45

66

33

22

11

Output

The elements of the list before sorting are:

34 2 1 789 99 45 66 33 22 11

1 2 3 4 5 6 7 8 9 10

Enter the element to be searched

99

The element whose value is 99 is present at position 6 of the list.

Enter the elements

34

Output

To compare the efficiency of algorithms, we don't rely on abstract measures such as the time difference in running speed, since it too heavily relies on the processor power and other tasks running.



4. Exercises

E1. Write a program that performs the following steps:

- * Input the integer array of N elements. (N is also an input data).
- * Input a value you want to search on the array.
- * Invoke the function `linearSearch` to find the element in the array which is equal to the specified E1. Write



Program

```
#include <stdio.h>
#define MAX 10
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void bsort(int arr[], int n)
{
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (arr[i] > arr[j])
                swap(&arr[i], &arr[j]);
}
int main()
{
    int arr[MAX];
    printf("Enter 10 numbers: ");
    for (int i = 0; i < MAX; i++)
        scanf("%d", &arr[i]);
    bsort(arr, MAX);
    printf("Sorted array: ");
    for (int i = 0; i < MAX; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```


3. SELECTION SORT

```
void selection_sort(int list[], int n)
{
    int i, j, min;

    for (i = 0; i < n - 1; i++)
    {
        min = i;
        for (j = i+1; j < n; j++)
        {
            if (list[j] < list[min])
            {
                min = j;
            }
        }
        swap(&list[i], &list[min]);
    }
}

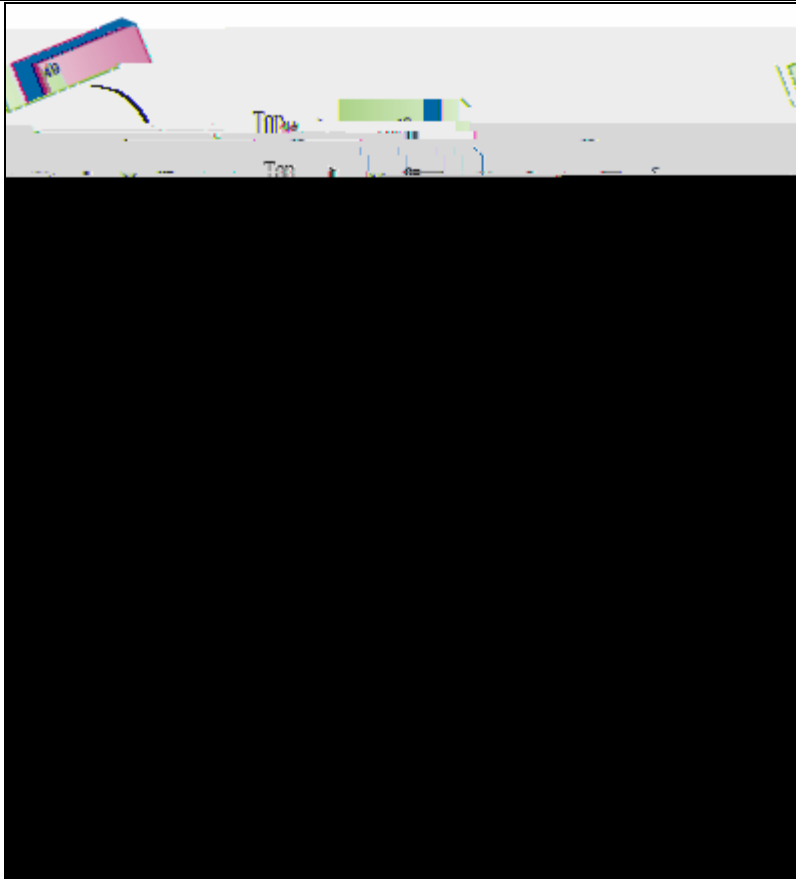
void printlist(int list[],int n)
{
    int i;
    for(i=0;i<n;i++)
```

}

$$\begin{aligned}T(n) &= 2 \cdot T(n-1) + c \\&= 2 \cdot [2 \cdot T(n-2) + c] + c \\&= 2^2 \cdot T(n-2) + 2c + c \\&= 2^2 \cdot [2 \cdot T(n-3) + c] + 2c + c \\&= 2^3 \cdot T(n-3) + 3c + c \\&\dots \\&\dots \\&= 2^k \cdot T(n-k) + k \cdot c + c \\&= 2^k \cdot T(n-k) + (k+1)c\end{aligned}$$

CHAPTER 5: STACKS AND QUEUES

1. *THE CONCEPT OF STACKS AND QUEUES*





Implementation of a Stack Using Linked Representation

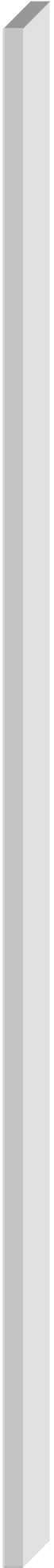
Initially the list is empty, so the top pointer is `NULL`. The push function takes a pointer to an existing list as the first parameter and a data value to be pushed as the second parameter, creates a new node by using the data value, and adds it to the top of the existing list. A pop function takes a pointer to an existing list as the first parameter, and a pointer to a data object in which the popped value is to be returned as a second parameter. Thus it retrieves the value of the node pointed to by the top pointer, takes the top point to the next node, and destroys the node that was pointed to by the top.

If this st.02 for cy th7 ry th2(eaxisti.02 a.9(s)-3.97t)1acky th7 9with roy









7. When the size of the stack/queue is known bef

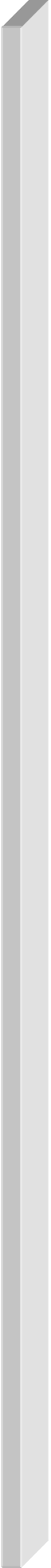


CHAPTER 6: LINKED LISTS

1. *THE CONCEPT OF THE LINKED LIST*

Introduction

When dealing with many problems we need a dynamic list, dynamic in the sense that the size requirement need not be known at compile time. Thus the list may grow or shrink during runtime. A *linked list* is a data structure that is used to model such a dynamic list of data items, so the study of the



```
    return (p);  
}
```

5. The main function reads the value of the number of nodes in the list. Calls iterate that many times by going in a



Figure 6.1: Sorting of a linked list.

To reverse a list, we maintain a pointer each to the previous and the next node, then we make the link field of the current node point to the previous, make the previous equal to the current, and the current equal to the next, as shown in Figure 6.2.



Figure 6.2: A linked list showing the previous, current, and next nodes at some point during reversal process.


```
scanf("%d",&x);  
start = newinsert(start,n,x);  
printf("The list after insertion is \n");  
printlist(start);  
}
```

Explanation


```
prev = NULL;
min = temp1 = p;
temp2 = p -> link;
while ( temp2 != NULL )
{
    if(min -> data > temp2 -> data)
    {
        min = temp2;
        prev = temp1;
    }
    temp1 = temp2;
    temp2 = temp2-> link;
}
if(prev == NULL)
    p = min -> link;
0 Tj0 -1.6108 TD(          e. NULL; )TjT-> link;
0 Tj0 link;
    prv == qL)
```




```

    struct dnode *temp, * temp1;
    int i;
    if ( node_no <= 0 || node_no > nodecount (p))
    {
        printf("Error! the specified node does not exist\n");
        exit(0);
    }
    if ( node_no == 0)
    {
        temp = ( struct dnode * )malloc ( sizeof ( struct dnode ));
        if ( temp == NULL )
        {
            printf( " Cannot allocate \n");
            exit (0);
        }
        temp -> data = value;
        temp -> right = p;
        temp->left = NULL
        p = temp ;
    }
    else
    {
        temp = p ;
        i = 1;
        while ( i < node_no )
        {
            i = i+1;
            temp = temp-> right ;
        }
        temp1 = ( struct dnode * )malloc ( sizeof(struct dnode));
        if ( temp == NULL )
        {
            printf("Cannot allocate \n");
            exit(0);
        }
        temp1 -> data = value ;
        temp1 -> right = temp -> right;
        temp1 -> left = temp;
        temp1->right->left = temp1;
        temp1->left->right = temp1
    }
    return (p);
}

void main()
{
    int n;

```




CHAPTER 7: TREES

1. *THE CONCEPT OF TREES*

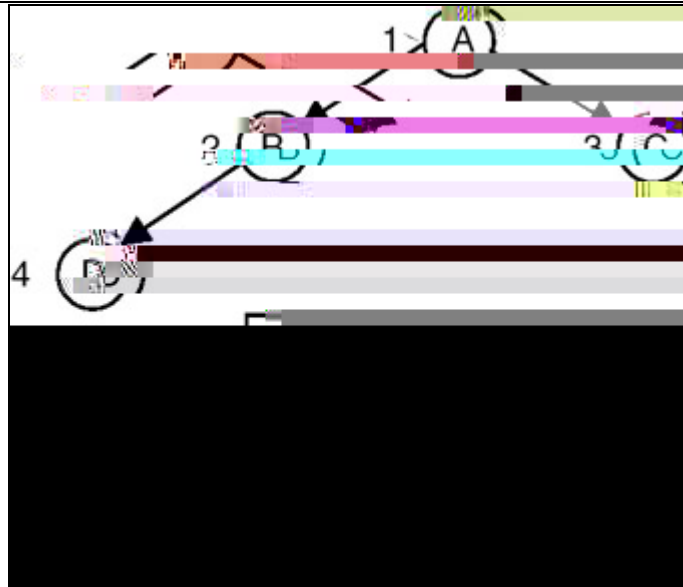
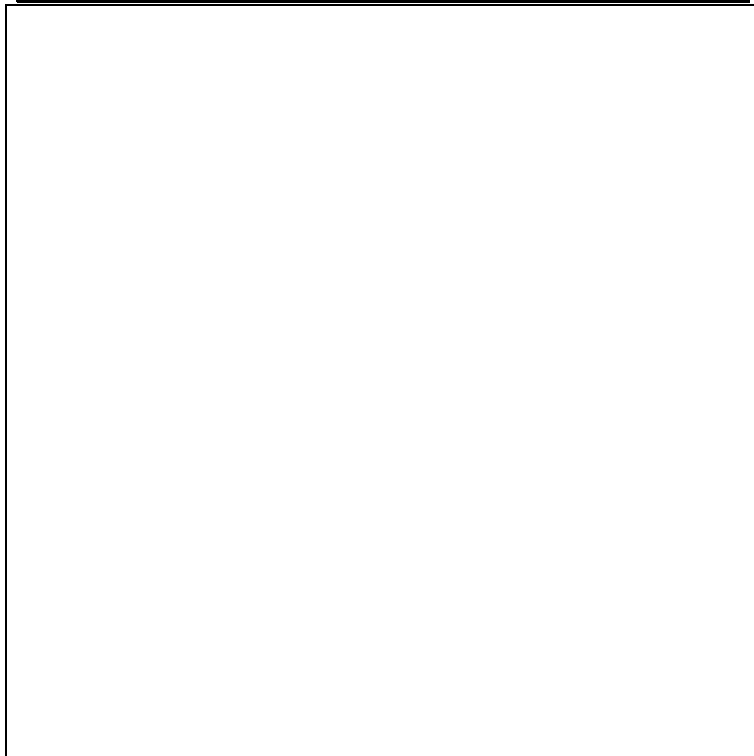
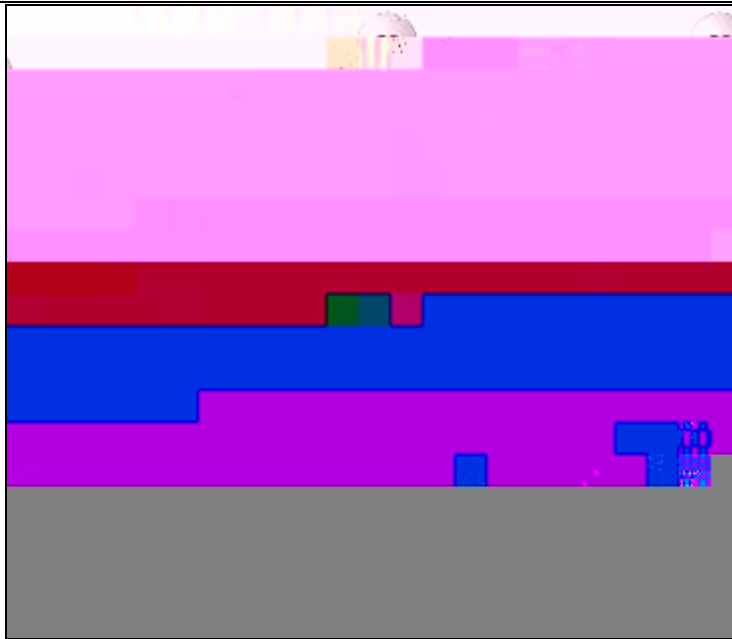
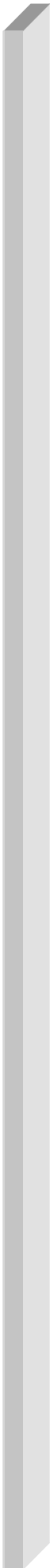


Figure 7.8: An array representation of a complete binary tree with 4 nodes and depth 3.

In general, any binary tree can be represented using







5. The degree of a tree is the maximum of the degree of the nodes of the tree.
6. The level of the root node is 1, and as we descend the tree, we increment the level of each node by 1.

AVL Tree Balance Requirements

The complexity of an AVL Tree comes from the balance requirements it enforces on each node. A

RL Rotation



- Find the sum of all node values in the tree.
- Find a node in the tree with input value. Replace the info number with new number.
- Adding a new node to the tree.
- Remove a node from the tree. The node is found from input value.
- Find the height of the tree.
-
-