Q.2 a. What is meant by an Abstract data type (ADT)? Explain generic ADT and how it is different from ADT. (4+4)

Answer:

Abstract Data Types:- A useful tool for specifying the logical properties of a data type is the abstract data type or ADT. The term “abstract data type” refers to the basic mathematical concept that defines the data type. In defining ADT as a mathematical concept, we are not concerned with space or time efficiency. An ADT consists of two parts:- a value definition and an operator definition. The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause. For example, the value consist definition for the ADT RATIONAL states that a RATIONAL value consists of two integers, the second of which does not equal 0. We use array notation to indicate the parts of an abstract type.

Refer section 1.6, page 10 of Text Book

b. What do you understand by the complexity of an algorithm? Explain relation between the time and space complexities of an algorithm. (4+4)

Answer:

Complexity of an Algorithm: An algorithm is a sequence of steps to solve a problem; there may be more than one algorithm to solve a problem. The choice of a particular algorithm depends upon following consideration:-
1) Time Complexity
2) Space Complexity
Time Complexity:- The time complexity of an algorithm is the amount of time it needs to run to completion. Some of the reasons for studying time complexity are:-
_ We may be interested to know in advance whether the program will provide a satisfactory real time response.
_ There may be several possible solutions with different time requirement.
Space Complexity:- The space complexity of an algorithm is the amount of memory it needs to run to completion. Some of the reasons to study space complexity are: -
_ There may be several possible solutions with in different space requirement.
_ To estimate the size of the largest problem that a program can solve.

Q.3 a. Describe the stack data structure. Also mention the limitations of using array for Stack implementation. (6)

Answer:
STACK: A stack is one of the most commonly used data structure. A stack is also called a Last-In-First-Out (LIFO) system, is a linear list in which insertion or deletion can take place only at one end called the top. This structure operates in the same way as the stack of trays If we want to place another tray, it can be placed only at the top. Similarly, if we want to remove a tray from stack of trays, it can only be removed from the top. The insertion and deletion operations in stack terminology are known as PUSH and POP operation.

limitations of Stack implementation using array:- To implement a stack we need a variable called top, that holds the index of the top element of stack and an array to hold the element of the stack.

For example:- #define MAX 100
```
Typedef struct
{ int top;
int elements [MAX]
} stack;
```

b. Using an appropriate example, explain the use of stack in Recursion. (4)

Answer: Refer page 152 of Text Book

c. Describe the Queue representation and implementation using suitable examples. (6)

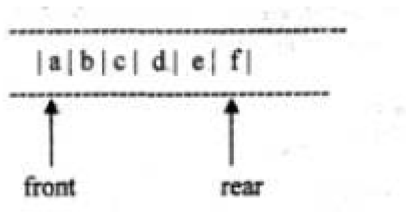
Answer:

Queue is a linear data structure used in various applications of computer science. Like people stand in a queue to get a particular service, various processes will wait in a queue for their turn to avail a service. In computer science, it is also called a FIFO (first in first out) list. In this chapter, we will study about various types of queues.

IMPLEMENTATION OF QUEUE

A physical analogy for a queue is a line at a booking counter. At a booking counter, customers go to the rear (end) of the line and customers are attendedto various services from the front of the line. Unlike stack, CustOmers are added at the rear end and deleted from the front end in a queue (FIFO).

An example of the queue in cornfiuter science is pfint jobs scheduled for printers. These jobs are maintained in a queue. The job fired forthe printer first gets printed first. Same is the scenario fôrjob scheduling in the CPU of computer. Like a stack, a queue also (usually) holds data elementsof the same type. We usually graphically display a queue horizontally.



Q.4 a. Write down a function to merge two linked lists containing same type of information in ascending order into a sorted single linked list. (6)

Answer:
```
merge(struct node *p, struct node *q, struct **s)
{
struct node *z;
z = NULL;
if((x= =NULL) && (y = =NULL))
```

```
return;
while(x!=NULL && y!=NULL)
{

if(*s= =NULL)

{
*s=(struct link *)malloc(sizeof(struct node *z));
z=*s;
}
else
{
z-->link=(struct link *)malloc(sizeof(struct node *));
z=z-->link;
}
if(x-->data < y-->data)
{
z-->data=x-->data;
x=x-->link;
}
else if(x-->exp > y-->exp)
{
z-->data=y-->data;
y=y-->link;
}
else if(x-->data= =y-->data)
{
z-->data=y-->data;
x=x-->link;
y=y-->link;
}
}
while(x!=NULL)
{
z_link = struct link *malloc(sizeof(struct node *));
z=z_link;
z-->data=x-->data;
x=x-->link;
}
while(y!=NULL)
{
z_link = struct link *malloc(sizeof(struct node *));
z=z_link;
z-->data=y-->data;
y=y-->link;
}
```

```
z-->link=NULL;
}
```

b. Write a C program to perform the creation of circular linked list. (4)

Answer:

```
#include<stdio.h>
#include<st4lib.h>
#define NULL 0
struct linked list
{
int data;
struct linked list next;
typedefstruct linked list clist;
clist hcad, s;
void main()
{
void create clist(clist 9;
int count(clisi 9;
void travcrse(clist 9;
head(clist ‘)malloc(sizeof(clist));
shead;
create clist( head);
printf(” n traversing the created clist and the starting address is %u \n”, head);
traverse(head);
printft(”\n number of elements in the clist O,/j \n”, count(head));
)
void create clist(clist start)
{
printf(”input the element -1111 for coming out of the loop\n’); scan f(O/odH, &start->data);
ifstart->data=-llll)
start-.>next=s;
else
I
start->next=(clist )malloc(sizeof(cl 1st));
create clist(start>next);
)
void traverse(clist start)
{
if(start->next!=s)
{
printf(”data is %d \t next element address is %u\n”, start->data, start >next); traverse(start-
>next);
}
if(start.>next = s)
printf(”data is %d \t next element address is %u\n’1start->data, start >next); int count(clist* start)
```

```
{
if(start->next s)
return 0;
else
return( 1 +count(start.>next));
```

c. Assume the linked list in the memory consisting of numerical values. Give a Program segment for the following:
(i) Find the maximum "MAX" of the values in the Linked list.
(ii) Find the average "MEAN" of the values in the Linked list. (6)

Answer:

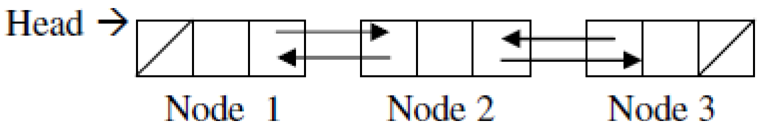
Linked List :-

A linked list is a linear collection of data elements called nodes. The linear order is given by pointer. Each node is divided into 2 or more parts. A linked list can be of the following types:-

_ Linear linked list or one way list
_ Doubly linked list or two way list.
_ Circular linked list
_ Header linked list

Representation of Linked list in Memory:-

Every node has an info part and a pointer to the next node also called as link. The number of pointers is two in case of doubly linked list. for example :



An external pointer points to the beginning of the list and last node in list has NULL. The space is allocated for nodes in the list using malloc or calloc functions. The nodes of the list are scattered in the memory with links to give linear order to the list.

(i)

```
float MAX_OF(node * start)
{
n=start;
max=n->k;
while(n!= NULL)
{
if(max<=n->k)
max=n->k;
}
return max;
```

```
(ii) float MEAN_OF(struct node * start)
{
int h=0; struct node *n;
n=start;
while (n!= NULL)
{
s+=n->k;h++;
n=n->next;
}
return s/h;
}
```

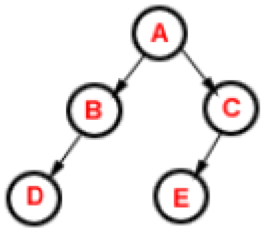
Q.5 a. What is threaded binary tree? How does Threaded binary tree represented in Data Structure? (3+4)

Answer:

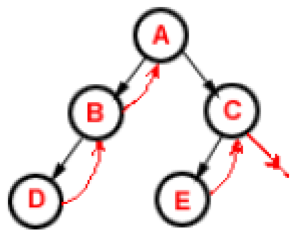
Threaded Binary Tree : If a node in a binary tree is not having left or right child or it is a leaf node then that absence of child node is represented by the null pointers. The space occupied by these null entries can be utilized to store some kind of valuable information. One possible way to utilize this space is to have special pointer that point to nodes higher in the tree that is ancestors. These special pointers are called threads and the binary tree having such pointers is called threaded binary tree. There are many ways to thread a binary tree each of these ways either correspond either in-order or pre-order traversal of T.A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time. The node structure for a threaded binary tree varies a bit and its like this

```
struct NODE
{
struct NODE *leftchild;
int node_value;
struct NODE *rightchild;
struct NODE *thread;
}
```

Let's make the Threaded Binary tree out of a normal binary tree...

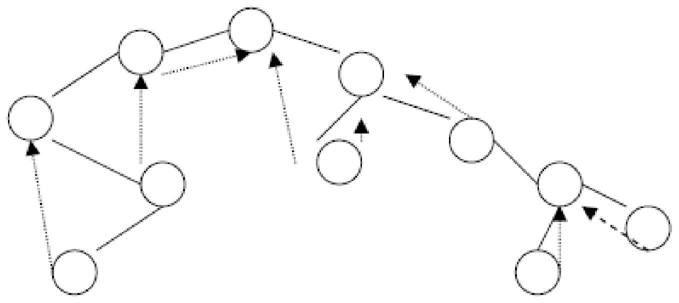


The INORDER traversal for the above tree is -- D B A E C. So, the respective threaded Binary tree will be –



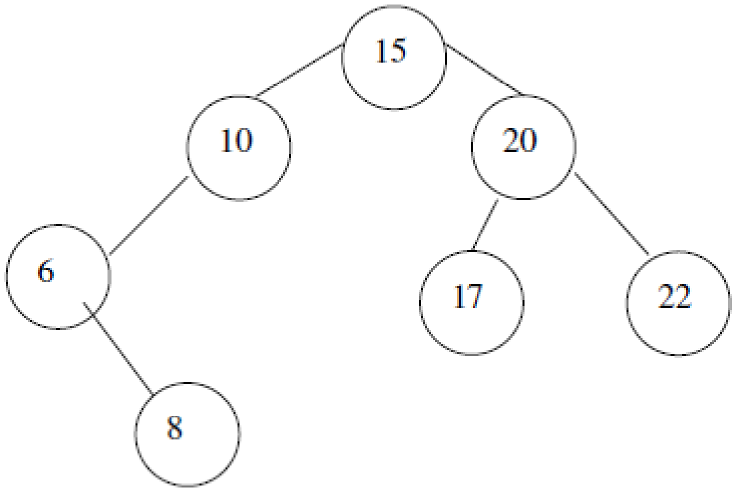
B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread. By changing the NULL lines in a binary tree to special links called threads, it is possible to perform traversal, insertion and deletion without using either a stack or recursion.

In **a right in threaded binary tree** each NULL link is replaced by a special lin to the successor of that node under inorder traversal called righ threaded. Using right threads we shall find it easy to do an inorder traversal of the tree, since we need only follow either an ordinary link or a threaded to find the next node to visit. If we replace each NULL left link by a special link to the predecessor of the node known as left threaded under inorder traversal the tree is known as **left in threaded binary tree**. If both the left and right threads are present in tree then it is known **as fully threaded binary tree** for example:

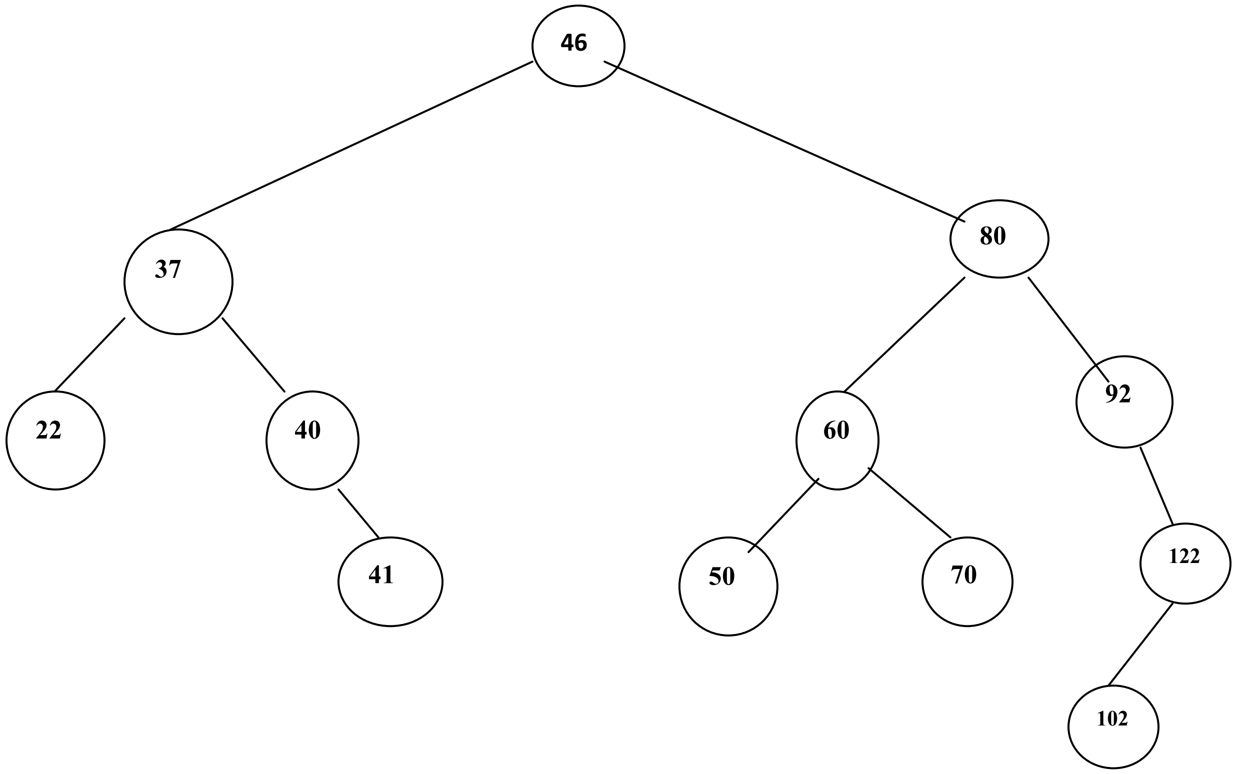


b. **Define Binary Search Tree (BST) with its essential properties. Generate a Binary Search Tree for the given series of numbers**
46, 37, 80, 22, 92, 122, 102, 40, 41, 60, 70, 50 **(4+5)**

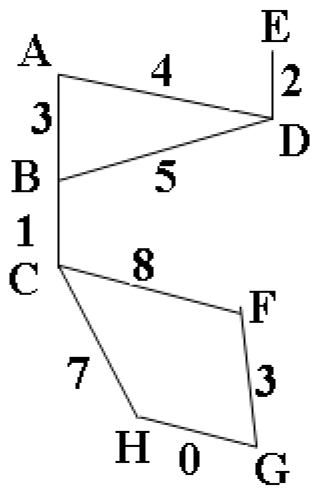
Answer:
Binary search tree. A binary search tree is a binary tree that is either empty or in which each node contains a key that satisfies the following properties: -
_ All keys (if any) in the left sub tree of the root precede the key in the root.
_ The key in the root precedes all keys (if any) in its right sub tree.
_ The left and right sub trees of the root are again search trees.



A Binary Search Tree for the given series of numbers: 46, 37, 80, 22, 92, 122, 102, 40, 41, 60, 70, 50



Q.6 a. What do you mean by Minimum Spanning Tree (MST) of a graph? Write the Kruskal's algorithm and apply it on the following graph to find the MST: (10)



Answer:

Minimum Spanning Tree (MST) :A Spanning Tree is any tree consisting of vertices of graph tree and some edges of graph is called a spanning tree. If graph has n vertices then spanning tree has exactly n-1 edges. A minimum spanning tree of an undirected graph *G* is a tree formed from graph edges that connects all the vertices of *G* at lowest total cost. A minimum spanning tree exists if and only if *G* is connected.

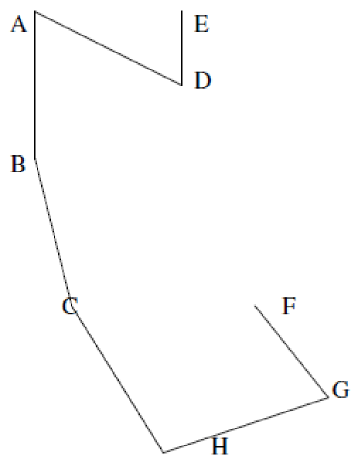
Kruskal's Algorithm:

```
MST-KRUSKAL($G, w$)
1  $A = \emptyset$
2  for each vertex $v \in G.V$
3      MAKE-SET($v$)
4  sort the edges of $G.E$ into nondecreasing order by weight $w$
5  for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6      if FIND-SET($u$) $\neq$ FIND-SET($v$)
7          $A = A \cup \{(u, v)\}$
8          UNION($u, v$)
9  return $A$
```

Solution of given graph using Kruskal's Algorithm:

Edge(u,v)	Weight	Included
(A,D)	4	√
(A,B)	3	√
(D,E)	2	√
(B,C)	1	√
(C,F)	8	X
(C,H)	7	√
(H,G)	0	√
(F,H)	3	√
(B,D)	5	X

The minimum spanning tree is:-

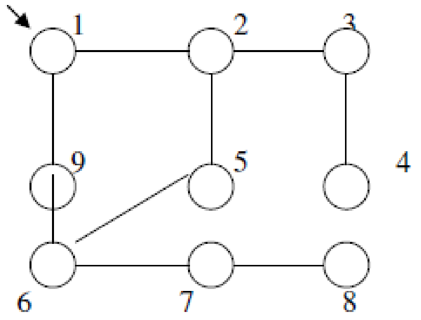


min cost = 2+4+3+1+7+0+3 = 20 units.

b. Discuss the different graph traversal techniques in detail. Also list their advantages & disadvantages. (6)

Answer:
Graph Traversal Scheme. In many problems we wish to investigate all the vertices in a graph in some systematic order. In graph we often do not have any one vertex singled out as special and therefore the traversal may start at an arbitrary vertex. The two
famous methods for traversing are:-
a) **Depth first traversal**: of a graph is roughly analogous to pre-order traversal of an ordered tree. Suppose that the traversal has just visited a vertex or, and let W0, W1,……Wk be the vertices adjacent

to V. Then we shall next visit W0 and keep W1…..Wk waiting. After visiting W0 we traverse all the vertices to which it is adjacent before returning to traverse W1, W2, ……..Wk.
b) **Breadth first traversal**: of a graph is roughly analogous to level by level traversal of ordered tree. If the traversal has just visited a vertex V, then it next visits all the vertices adjacent to V. Putting the vertices adjacent to these is a waiting list to be traversed after all the vertices adjacent to V have been visited. The figure below shows the order of visiting the vertices of one graph under both DFS and BFS.



DFT = 1 2 3 4 5 6 7 8 9
BFT= 1 2 9 3 5 6 4 7 8

Q.7 a. Describe Hashing and Hash function. What are problems in hashing? (4+2)

Answer:
Hashing provides the direct access of record from the file no matter where the record is in the file. This is possible with the help of a hashing function H which map the key with the corresponding key address or location. It provides the keyto- address transformation. Five popular hashing functions are as follows:
Division Method: An integer key x is divided by the table size m and the remainder is taken as the hash value. It can be defined as
H(x)=x%m+1 For example, x=42 and m=13, H(42)=45%13+1=3+1=4
Midsquare Method: A key is multiplied by itself and the hash value is obtained by selecting an appropriate number of digits from the middle of the square. The same positions in the square must be used for all keys. For example if the key is 12345, square of this key is value 152399025. If 2 digit addresses is required then position 4th and 5th can be chosen, giving address 39.
Folding Method: A key is broken into several parts. Each part has the same length as that of the required address except the last part. The parts are added together, ignoring the last carry, we obtain the hash address for key K.
Multiplicative method: In this method a real number c such that 0<c<1 is selected. For a nonnegative integral key x, the hash function is defined as H(x)=[m(cx%1)]+1
Here,cx%1 is the fractional part of cx and [] denotes the greatest integer less than or equal to its contents.
Digit Analysis: This method forms addresses by selecting and shifting digits of the original key. For a given key set, the same positions in the key and same rearrangement pattern must be used. For example, a key 7654321 is transformed to the address 1247 by selecting digits in position 1,2,4 and 7 then by reversing their order.

problems in hashing : Although hash table lookups use constant time on average, the time spent can be significant. Evaluating a good hash function can be a slow operation. In particular, if simple array indexing can be used instead, this is usually faster.

Hash tables in general exhibit poor locality of reference—that is, the data to be accessed is distributed seemingly at random in memory. Because hash tables cause access patterns that jump around, this can trigger microprocessor cache misses that cause long delays. Compact data structures such as arrays, searched with linear search, may be faster if the table is relatively small and keys are cheap to compare, such as with simple integer keys.

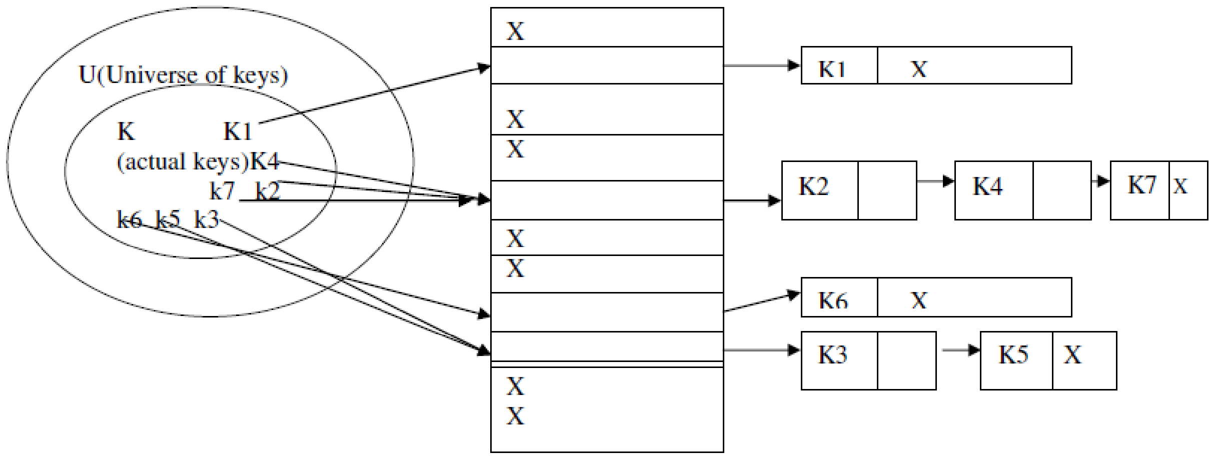
More significantly, hash tables are more difficult and error-prone to write and use. Hash tables require the design of an effective hash function for each key type, which in many situations is more difficult and time-consuming to design and debug than the mere comparison function required for a self-balancing binary search tree. In open-addressed hash tables it's even easier to create a poor hash function.

Additionally, in some applications, a black hat with knowledge of the hash function may be able to supply information to a hash which creates worst-case behavior by causing excessive collisions, resulting in very poor performance (i.e., a denial of service attack). In critical applications, either universal hashing can be used or a data structure with better worst-case guarantees may be preferable

b. Explain the different collision resolution techniques. (4)

Answer:

Methods of Collision Resolution: Two broad classes of collision resolution techniques are:
1) Collision Resolution by separate chaining
2) Collision Resolution by open addressing
1) Separate chaining: In this scheme all elements whose key hash to same hash table slot are put in a linked list. Thus the slot in the hash table contains a pointer to the head of the linked list of all the elements that hash to value i. If there is no such element that hash to value I , the slot I contains NULL value (pictures as X)



2) Open addressing: The simplest way to resolve a collision is to start with the hash address and do a sequential search through the table for an empty location. The idea is to place the record in the next available position in the array. This method is called linear probing. An empty record is indicated by a special value called null. The major drawback of the linear probe method is clustering.

c. **Write an algorithm for binary search. List out the conditions under which linear search of a list is preferred over binary search. (4+2)**

Answer:

Binary search: This technique is applied to an ordered list where elements are arranged either in ascending order or descending order. The array is divided into two parts and the item being searched for is compared with item at the middle of the array. If they are equal the search is successful. Otherwise the search is performed in either the first half or second half of the array. If the middle element is greater than the item being searched for the search process is repeated in the first half of the array otherwise the process is repeated in the second half of the array. Each time a comparison is made, the number of element yet to be searched will be cut in half. Because of this division of array into two equal parts, the method is called a binary search.

Algorithm for Binary Search

```
1. if (low> high)
2. return (-1)
3. Mid = (low + high)/2
4. if ( X = = a[mid])
5. return (mid);
6. if (X < a[mid])
7. search for X in a[low] to a[mid-1]
8. else
9. search for X in a[mid+1] to a[high]
```

In binary search, each iteration of the loop halves the size of the list to be searched. Thus the maximum number of key comparisons is approximately 2*log2n. This is quite an improvement over the Linear search, especially as the list gets larger. The binary search is, however, not guaranteed to be faster for searching for small lists. Even though the binary search requires less number of comparisons, each comparison requires more computation. When n is small, the constant of proportionality may dominate. Therefore, in case of less number of elements in the list, a Linear search is adequate. Linear Search is a preferred over binary search when the list is unordered and haphazardly constructed. When searching is to be performed on unsorted list then linear search is the only option.

Q.8 a. Consider a list which is already sorted, which sorting algorithm is performed best and why? Also State that algorithm. **(4)**

Answer:

Consider a list is which is already sorted Insertion sort as there is no movement of data if the list is already sorted and complexity is of the order O(N)

Insertion Sort: One of the simplest sorting algorithms is the *insertion sort.* Insertion sort consists of *n* - 1 *passes.* For pass *p* = 2 through *n*, insertion sort ensures that the elements in positions 1 through *p* are in sorted order. Insertion sort makes use of the fact those elements in positions 1 through *p* - 1 are already known to be in sorted order.

```
Void insertion_sort( input_type a[ ], unsigned int n )
{
unsigned int j, p;
input_type tmp;
a[0] = MIN_DATA; /* sentinel */
for( p=2; p <= n; p++ )
{
tmp = a[p];
for( j = p; tmp < a[j-1]; j-- )
a[j] = a[j-1];
a[j] = tmp;
}
}
```

Because of the nested loops, each of which can take *n* iterations, insertion sort is *O*(*n*2). Furthermore, this bound is tight, because input in reverse order can actually achieve this bound. A precise calculation shows that the test at line 4 can be executed at most *p* times for each value of *p*. Summing over all *p* gives a total of (n(n-1))/2 = *O*(*n*2).

b. Explain Heap Sort algorithm. Sort the following sequence of numbers using heap sort algorithm:
20, 12, 25 6, 10, 15, 13 (6+6)

Answer:

HeapSort:

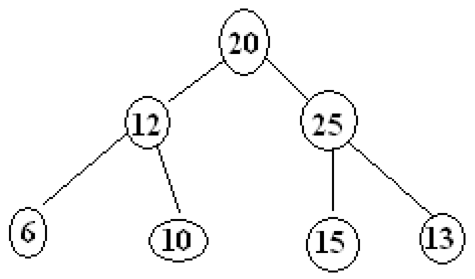
The idea is that we need to repeatedly extract the maximum item from the heap. As we mentioned earlier, this element is at the root of the heap. But once we remove it we are left with a hole in the tree. To fix this we will replace it with the last leaf in the tree (the one at position A [m]). But now the heap order will very likely be destroyed. So we will
just apply Heapify to the root to fix everything back up.

Heap sort algorithm:

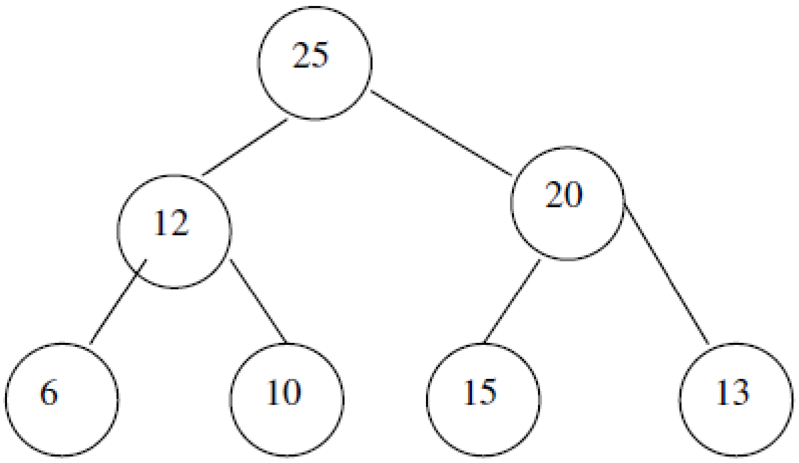
```
HeapSort(int n, array A[1..n]) { // sort A[1..n]
(
BuildHeap(n, A) // build the heap
m = n // initially heap contains all
while (m >= 2)
 {
swap A[1] with A[m] // extract the m-th largest
m = m-1 // unlink A[m] from heap
Heapify(A, 1, m) // fix things up
}

}
```

Let the following numbers be stored in array.
20, 12, 25, 6, 10, 15, 13



First we create the heap from the above binary tree:-

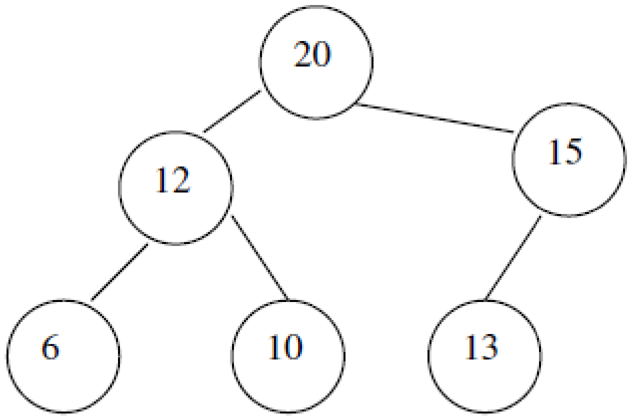


Now the above tree is a heap. Therefore, we store the above tree in an implicit representation of tree in an array.

25	12	20	6	10	15	13

Therefore 25 is target we place 25 in last of array.

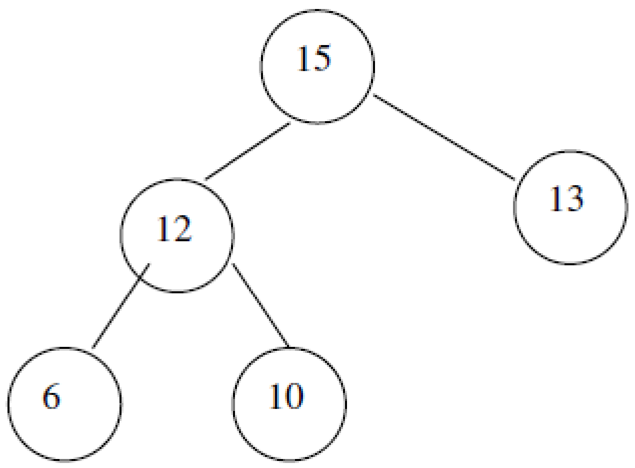
20	12	15	6	10	13	25



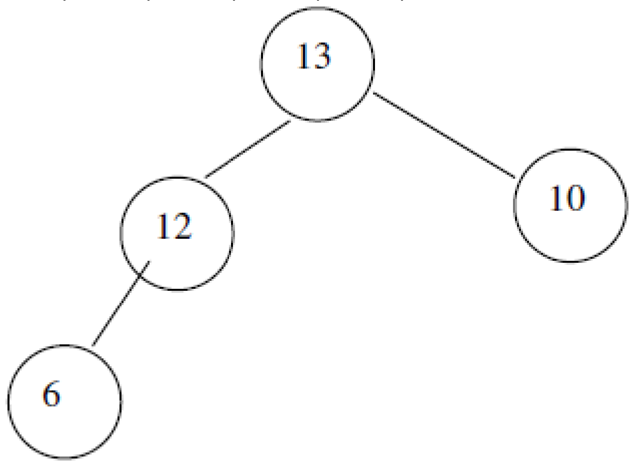
Now put 20 in its proper position by replacing 13

15	12	13	6	10	20	25

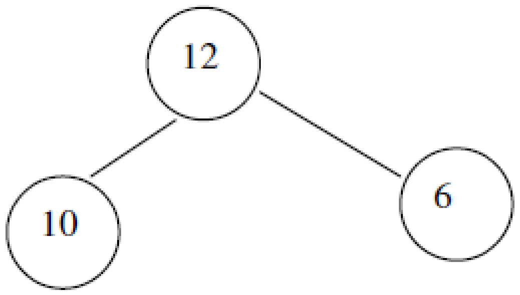
After reheapifying



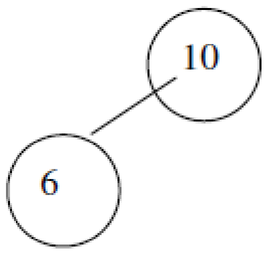
13	12	6	10	15	20	25



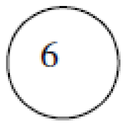
12	10	6	13	15	20	25



10	6	12	13	15	20	25



6	10	12	13	15	20	25



Q.9 a. Explain Direct File organization. State advantages and disadvantages of this file organization. (2+6)

Answer:

The methods available in storing sequential files are:

•Straight merging,
•Natural merging,
•Polyphase sort,
•Distribution of Initial runs.

Program :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main( )
{
FILE *fp ;
char str [ 67 ] ;
int i, noofr, j ;
clrscr( ) ;
printf ( "Enter file name: " ) ;
scanf ( "%s", str ) ;
printf ( "Enter number of records: " ) ;
scanf ( "%d", &noofr ) ;
fp = fopen ( str, "wb" ) ;
if ( fp == NULL )
{
printf ( "Unable to create file." ) ;
getch( ) ;
exit ( 0 ) ;
}
randomize( ) ;
for ( i = 0 ; i < noofr ; i++ )
{
j = random ( 1000 ) ;
fwrite ( &j, sizeof ( int ), 1, fp ) ;
printf ( "%d\t", j ) ;
}
fclose ( fp ) ;
printf ( "\nFile is created. \nPress any key to continue." ) ;
getch( ) ;
}
```

b. **Give a Program code in C that copies the contents of one file into another file using command line arguments. (8)**

Answer:

```
#include<stdio.h>
 #include<conio.h>
 #include<stdlib.h>
void main(int arg,char *arr[])
{
FILE *fs,*ft;
char ch;
clrscr();
if(arg!=3)
{
printf("Argument Missing ! Press key to exit.");
getch();
exit(0);
}
fs = fopen(arr[1],"r");
if(fs==NULL)
{
 printf("Cannot open source file ! Press key to exit.");
getch();
exit(0);
 }
ft = fopen(arr[2],"w");
 if(ft==NULL)
{
printf("Cannot copy file ! Press key to exit.");
fclose(fs);
 getch();
exit(0);
}
while(1)
{
ch = getc(fs);
 if(ch==EOF)
{
break;
}
else
 putc(ch,ft);
 }
printf("File copied succesfully!");
 fclose(fs);
fclose(ft);
 }
```

TEXT BOOK

I. Data Structures using C & C++, Rajesh K. Shukla, Wiley India. 2009