Q.2a. Why do we use a symptotic notation in the study of algorithm? Describe commonly used asymptotic notations and give their significance. (2+5)

Answer:

) An algorithm is a step by step process to solve a particular problem.

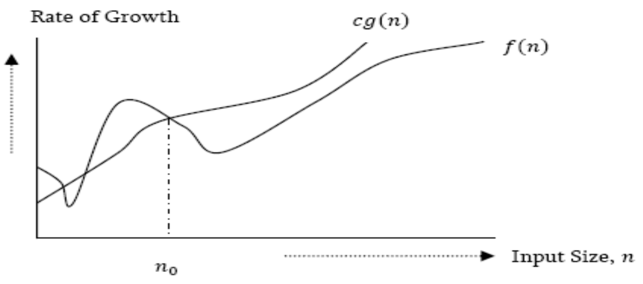
It is the process of determining how processing time increases as the size of the problem (input size) increases. Input size is number of elements in the input and depending on the problem type the input may be of different types. In general, the types of inputs are as follows.

Size of an array, Polynomial degree, Number of elements in a matrix,Number of bits in binary representation of the input,Vertices and edges in a graph

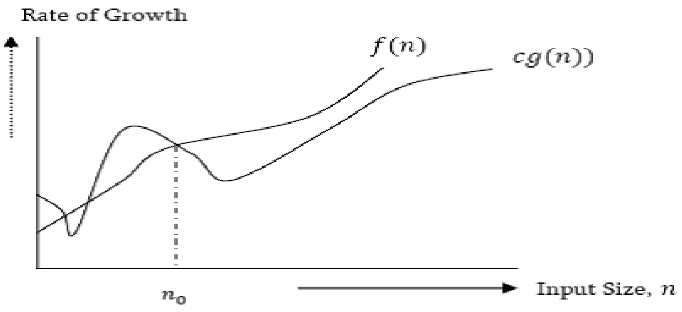
A particular problem can solve by different algorithms, but we have to use the best algorithm among them. So, that can be decided by comparing the running time of all the algorithms. To measure running time complexity for a given algorithm, asymptotic Notation is the best method , because this method is not depend on machine time, programming style, etc..

Different types of asymptotic notations are as follows.

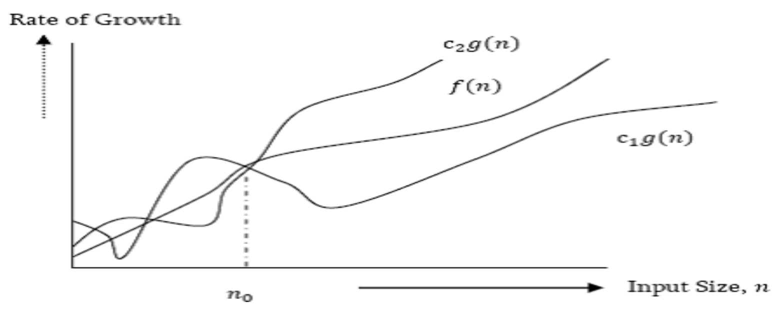
Big oh notation(O): for non-negative functions, *f(n)* and *g(n),* if there exists an integer n_0 nd a constant *c* >0 such that for all integers $n>n_0$,*f(n)≤cg(n),* then *f(n)*is Big O of *g(n)*



Big Omega notation(Ω):For non-negative functions,*f(n)* and *g(n),* if there exists an integer n_0 and a constant *c* >0 such that for all integers $n>n_0$, *f(n) ≥ cg(n),* then *f(n)* is omega of *g(n).*



Big Theata notation(θ): For non-negative functions,*f(n)* and *g(n),* if there exists an integer n_0 and c1 & c2 two constants *c1* >0 & c2>0 such that for all integers $n>n_0$,then c1g(n) ≤*f(n)* ≥ *c2g(n),* then *f(n)* is omega of *g(n)*



b. A linear array A is given with lower bound as 1. If address of A[25] is 375 and A[30] is 390, then find address of A[16]. (4)

Answer:

Loc (a[k]) = base (a) + w (k-lb)
375 = base (a) + w(25 – 1)
390 = base (a) + w(30 – 1)
375 = x + 24w
390 = x + 29w
15 = 5w
∴ w = 3
∴ x = 375 – 24 *3
x = 375 – 72
x = 303
∴base address is 303 and w = 3.
∴Address of A[16] is
loc = 303+ 3(16-1) = 348.

c. Write a program using Pointers to compute the sum of all elements stored in an array. (5)

Answer:

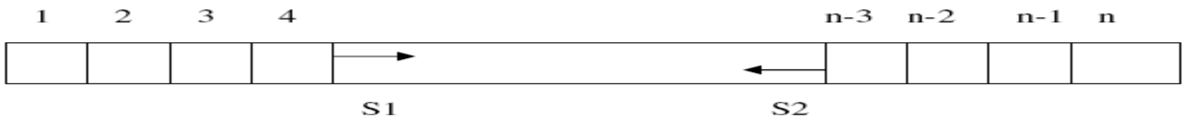
```
) #include<stdio.h>
main()
{
int *p,sum,i;
int x[5]={2,7,8,9,6};
i=0;
p=x;        /* initializing with base address*/
printf("Element value address\n");
while(i<5)
{
printf("x[%d] %d %u\n",i,*p,p);
sum=sum+*p;
i++;
p++;
}
printf("sum=%d\n",sum);
printf("\n&x[0]=%u\n",&x[0]);
```

printf("\n p=%u\n",p);

Q.3 a. Explain how to implement two stacks in one array A[1..n] in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in O(1) time. (6)

Answer:

a) Two stacks s1 and s2 can be implemented in one array A[1,2,...,N] as shown in the following figure



We define A[1] as the bottom of stack S1 and let S1 "grow" to the right and we define A[n] as the bottom of the stack S2 and S2 "grow" to the left. In this case, overflow will occur only S1 and S2 together have more than n elements. This technique will usually decrease the number of times overflow occurs. There will separate push1, push2, pop1 and pop2 functions to be defined separately for two stacks S1 and S2.

b. How can stacks be used to check whether an expression is correctly parenthized or not. For example (()) is well formed but (() or) () (is not. (6)

Answer:

a) To determine whether an expression is well parenthized or not:- two conditions should be fulfilled while pushing an expression into a stack. Firstly, whenever an opening bracket is pushed inside a stack, there must be an occurrence a closing bracket before the last symbol is reached. Whenever a closing bracket is encountered, the top of the stack is popped until the opening bracket is popped out and discarded. If no such opening bracket is found and stack is emptied, then this means that the expression is not well parenthized.

An algorithm to check whether an expression is correctly parenthized is as follows:

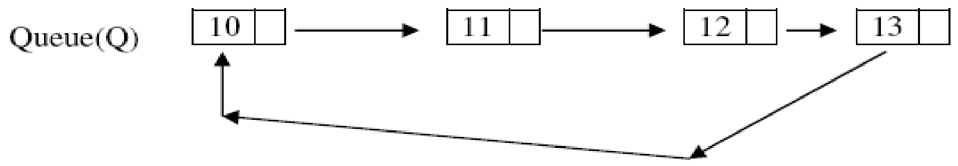
```
flag=TRUE;
clear the stack;
Read a symbol from input string;
while not end of input string and flag do
{
if(symbol= '( ' or symbol= '[' or symbol = '{' )
push(symbol,stack);
else if(symbol= ') ' or symbol= '[' or symbol ='{' )
if stack is empty
flag=false;
printf("More right parenthesis than left
parenthises");
else
c=pop(stack);
match c and the input symbol;
If not matched
{
flag=false;
printf("Mismatched
parenthesis”);
}
Read the next input symbol;
}
if stack is empty then
printf(“parentheses are balanced properly”);
else
printf(“ More left parentheses than right parentheses”);
```

c. Can a Queue be represented by circular linked list with only one pointer pointing to the tail of the queue? Substantiate your answer using an example.(4)

Answer:

) Yes a Queue can be represented by a circular linked list with only one pointer pointing to the tail of the queue.

for example:

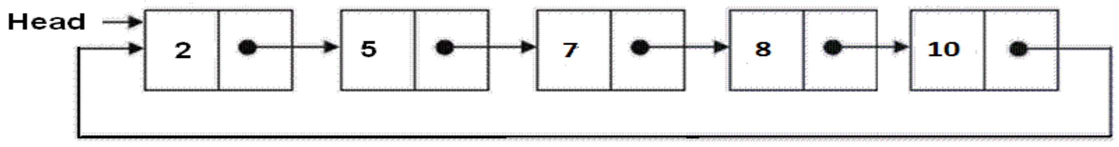


Q has a pointer pointing to the tail of ‘Q’.This pointer represents real of the Q and ptr->next is the front of Q.

Q.4 a. What is a circular link list? Write the properties of circular link list. (2+2)

Answer:

a) A circular linked list is a linked list in which last element or node of the list points to first node.



For non-empty circular linked list, there are no NULL pointers. The memory declarations for representing the circular linked lists are the same as for linear linked lists. All operations performed on linear linked lists can be easily extended to circular linked lists with following exceptions:

• While inserting new node at the end of the list, its next pointer field is made to point to the first node.

• While testing for end of list, we compare the next pointer field with address of the first node Circular linked list is usually implemented using header linked list. Header linked list is a linked list which always contains a special node called the header node, at the beginning of the list. This header node usually contains vital information about the linked list such as number of nodes in lists, whether list is sorted or not etc. Circular header lists are frequently used instead of ordinary linked lists as many operations are much easier to state and implement using header listThis comes from the following two properties of circular header linked lists:

i.The null pointer is not used, and hence all pointers contain valid addresses
ii.Every (ordinary) node has a predecessor, so the first node may not require a special case.

b. Write the algorithm to insert and delete a node in a circular link list. (8)

Answer:

Insertion in a circular header linked list

Algorithm: INSRT(INFO,LINK,START,AVAIL,ITEM,LOC)
(This algorithm inserts item in a circular header linked list)

```
after the location LOC
Step 1:If AVAIL=NULL, then
Write: 'OVERFLOW'
Exit
Step 2: Set NEW:=AVAIL and AVAIL:=LINK[AVAIL]
Step 3: Set INFO[NEW]:=ITEM
Step 4: Set LINK[NEW]:=LINK[LOC]
Set LINK[LOC]:=NEW
Step 5: Return
```

Deletion from a circular header linked list

Algorithm: DELLOCHL(INFO,LINK,START,AVAIL,ITEM)
(This algorithm deletes an item from a circular headerlinked list)

```
Step 1: CALL FINDBHL(INFO,LINK,START,ITEM,LOC,LOCP)
Step 2: If LOC=NULL, then:
Write: 'item not in the list'
Exit
Step 3: Set LINK[LOCP]:=LINK[LOC] [Node deleted]
Step 4: Set LINK[LOC]:=AVAIL and AVAIL:=LOC
[Memory retuned to Avail list]
Step 5: Return
```

c. Write the comparison in between array and link list. (4)

Answer:

Comparison inbetween array and linklist

(1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.
(2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.
For example, suppose we maintain a sorted list of IDs in an array id[].
id[] = [1000, 1010, 1050, 2000, 2040, …..].
And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

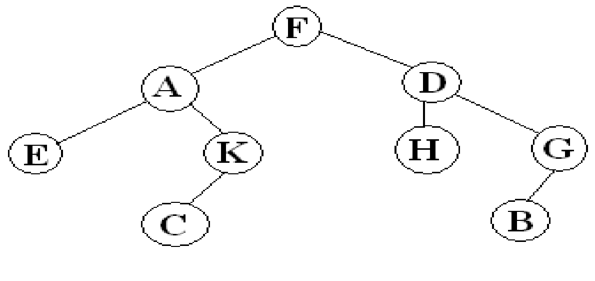
So Linked list provides following two advantages over arrays
1) Dynamic size
2) Ease of insertion/deletion

Linked lists have following drawbacks compare to an array:
1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
2) Extra memory space for a pointer is required with each element of the list.
3) Arrays have better cache locality that can make a pretty big difference in performance.

Q.5 a. A Binary tree has 9 nodes. The inorder and preorder traversals of the tree yields the following sequence of nodes: (6)
Inorder : E A C K F H D B G
Preorder: F A E K C D H G B
Draw the tree. Explain your algorithm.

Answer:

Inorder: E A C K F H D B G
Preorder: F A E K C D H G B
The tree T is drawn from its root downward as follows.
i) The root T is obtained by choosing the first node in its preorder. Thus F is the root of T.
ii) The left child of the node F is obtained as follows. First use the inorder of T to find the nodes the in the left subtree T 1 of F. Thus T 1 consists of the nodes E, A, C and K. Then the left child of F is obtained by choosing the first node in the preorder of T 1 (which appears in the preorder of T). Thus A is the left son of F.
iii) Similarly, the right subtree T 2 of F consists of the nodes H, D, B and G, and D is the root of T 2 , that is, D is the right child of F.
Repeating the above process with each new node, we finally obtain the required tree.



b. Explain the following terms with respect to Binary trees (6)
(i) Strictly Binary Tree
(ii) Complete Binary Tree
(iii) Almost Complete Binary Tree

Answer:

(i) Strictly Binary Tree:- If every non leaf node in binary tree has non empty left and right sub-trees , then the tree is called a strictly binary tree.
(ii) Complete Binary Tree:- A complete binary tree of depth d is that strictly binary tree all of whose leaves are at level D.

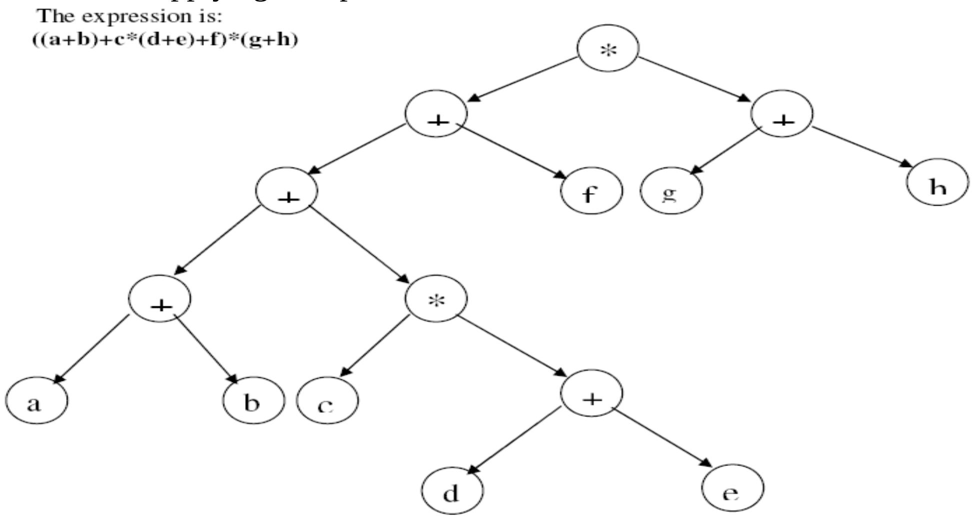
(iii) Almost Complete Binary Tree:- A binary tree of depth d is an almost complete binary tree if: 1.Any node nd at level less than d-1 has two children. 2. for any node nd in the tree with a right descendant at level d, nd must have a left child and every descendant of nd is either a leaf at level d or has two children.

c. What is an expression binary tree? Draw the expression tree of the following infix expression. (4)
((a + b) + c * (d + e) + f) * (g + h)

Answer:
It is a special binary tree which have the following properties

i. Each leaf node contains a single operand

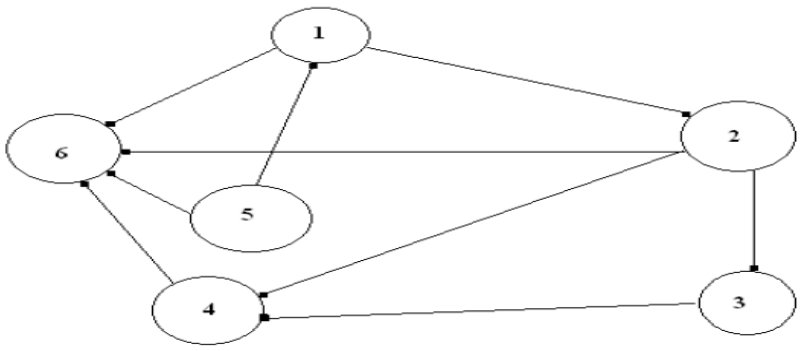
ii. Each nonleaf node contains a single binary operator
iii. The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.



Q.6 a. Draw a picture of the directed graph specified below: (6)
G = (V, E)
V(G) = {1, 2, 3, 4, 5, 6}
E(G) = {(1,2), (2, 3), (3, 4), (5,1), (5, 6), (2, 6), (1, 6), (4, 6), (2, 4)}

Obtain the following for the above graph:
(i) Adjacency matrix.
(ii) Path matrix

Answer:



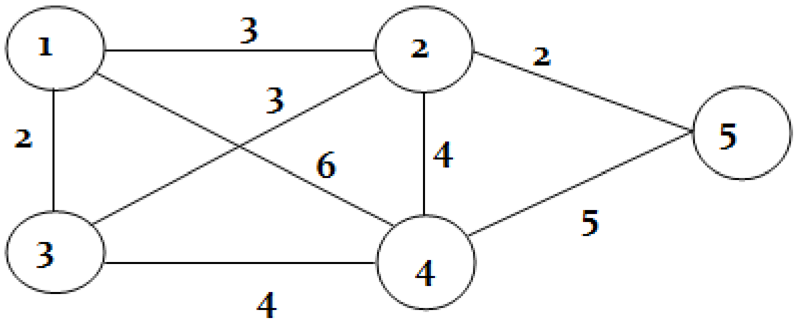
(i) Adjacency matrix

	1	2	3	4	5	6
1	0	1	0	0	0	1
2	0	0	1	1	0	1
3	0	0	0	1	0	0
4	0	0	0	0	0	1
5	1	0	0	0	0	1
6	0	0	0	0	0	0

(ii) Reachability Matrix (Path Matrix)

	1	2	3	4	5	6
1	0	1	1	1	0	1
2	0	0	1	1	0	1
3	0	0	0	1	0	1
4	0	0	0	0	0	1
5	1	1	1	1	0	1
6	0	0	0	0	0	0

b. What is the difference between Prims algorithm and Kruskals algorithm for finding the minimum-spanning tree of a graph? Execute the Prims algorithm on the following graph. **(4+6)**



Answer:

a) Prim’s algorithm initializes with a node, whereas Kruskal’s algorithm initiates with an edge.

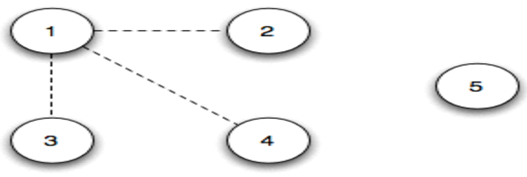
•Prim’s algorithms span from one node to another while Kruskal’s algorithm select the edges in a way that the position of the edge is not based on the last step.

•In prim’s algorithm, graph must be a connected graph while the Kruskal’s can function on disconnected graphs too.

•Prim’s algorithm has a time complexity of O(V2), and Kruskal’s time complexity is O(logV).

MST with Prim's algorithm

Step 1: Dequeue vertex 1 and update *Q* (and reprioritizing) by changing *u*3.*key* = 2 (edge (*u*1,*u*3)),*u*2.*key* = 3 (edge (*u*1,*u*2)), *u*4.*key* = 6 (edge (*u*1,*u*4))

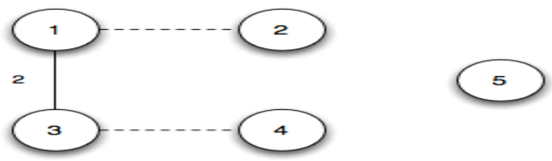


Q

key	u	π
2	3	1
3	2	1
6	4	1
∞	5	/

w(T) = 0

Step 2: Dequeue vertex 3 (adding edge (*u*1,*u*3) to *T*) and update *Q* (and reprioritizing) by changing *u*4.*key* = 4 (edge (*u*3,*u*4))

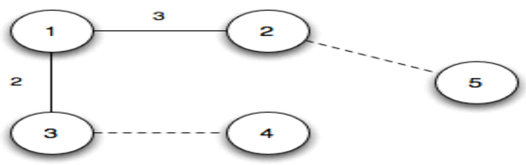


Q

key	u	π
3	2	1
4	4	3
∞	5	/

w(T) = 2

Step 3: Dequeue vertex 2 (adding edge (*u*1,*u*2) to *T*) and update *Q* (and reprioritizing) by changing *u*5.*key* = 2 (edge (*u*2,*u*5))

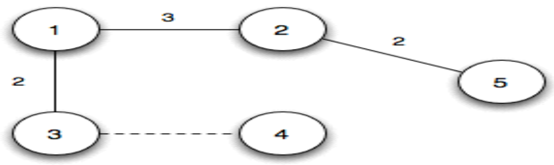


Q

key	u	π
2	5	2
4	4	3

w(T) = 5

Step 4: Dequeue vertex 5 (adding edge (*u*2,*u*5) to *T*) with no updates to *Q*

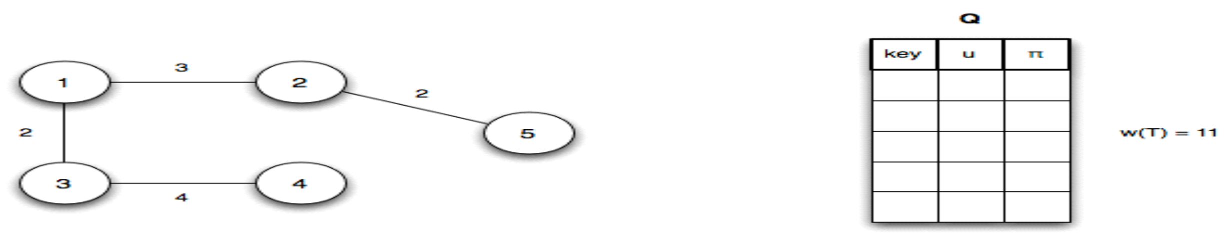


Q

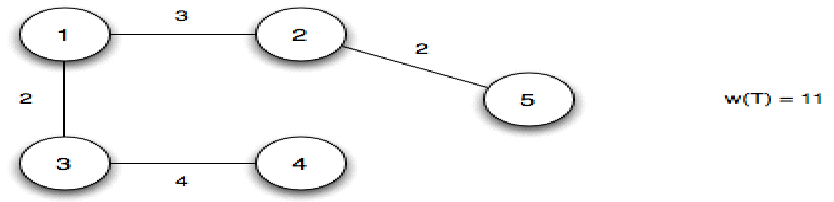
key	u	π
5	4	3

w(T) = 7

Step 5: Dequeue vertex 4 (adding edge (*u*3,*u*4) to *T*) with no updates to *Q*



At this point $Q = \emptyset$ giving the final MST



Total weight of the MST is 11.

Q.7 a. How do we pick a good hash function? Write the procedure to Implement a Simple Hash Table. (4+4)

Answer:

What we mean by “good” is that the function must be easy to compute and avoid collisions as much as possible. If the function is hard to compute, then we lose the advantage gained for lookups in O(1). Even if we pick a very good hash function, we still will have to deal with “some” collisions.

It is difficult to find a “perfect” hash function, that is a function that has no collisions. But we can do “better” by using hash functions as follows. Suppose we need to store a dictionary in a hash table. A dictionary is a set of Strings and we can define a hash function as follows. Assume that S is a string of length n and $S = S_1 S_2 \dots S_n$

$$\mathbf{H(S) = H(“S_1\, S_2 \dots S_n”) = S_1 + p\, S_2 + p^2\, S_3 + \dots + p^{n-1}\, S_n}$$

where p is a prime number. Obviously, each string will lead to a unique number, but when we take the number Mod TableSize, it is still possible that we may have collisions but may be fewer collisions than when using a naïve hash function like the sum of the characters.

Although the above function minimizes the collisions, we still have to deal with the fact that function must be easy to compute. Rather than directly computing the above functions, we can reduce the number of computations by rearranging the terms as follows.

$$\mathbf{H(S) = S_1 + p\,(\,S_2 + p(S_3 \;+ \dots P\,(S_{n-1} + p\,S_n\,))))}$$

This rearrangement of terms allows us to compute a good hash value quickly.

Implementation

A hash table is stored in an array that can be used to store data of any type. In this case, we will define a generic table that can store nodes of any type. That is, an array of void*’s can be defined as follows.

void* A[n];

The array needs to be initialized using

```
for (i = 0; i < n ; i++)
A[i] = NULL;
```

Suppose we like to store strings in this table and be able to find them quickly. In order to find out where to store the strings, we need to find a value using a hash function. One possible hash function is Given a string $S = S_1 S_2 \ldots S_n$

Define a hash function as

$H(S) = H(\text{“}S_1S_2 \ldots S_n\text{”}) = S_1 + p\,S_2 + p_2\,S_3 + \ldots + p_{n-1}S_n$ ----------------(1)

where each character is multiplied by a power of p, a prime number. The above equation can be factored to make the computation more effective.

Using the factored form, we can define a function hashcode that computes the hash value for a string s as follows.

```
 int hashcode(char* s){
int sum = s[strlen(s)-1], p = 101;
 int i;
for (i=1; i<strlen(s);i++)
sum = s[strlen(s)-i-1] + p*sum;
return sum;
}
```

This allows any string to be placed in the table as follows. We assume a table of size 101.

A[hashcode(s)%101] = s; // we assume that memory for s is already being allocated.

One problem with above method is that if any collisions occur, that is two strings with the same hashcode, and then we will lose one of the strings. Therefore we need to find a way to handle collisions in the table.

b. Write a C program to search an element in an array using binary search. (6)

Answer:

a)
```
#include<stdio.h>
int main(){
inta[10],i,n,m,c=0,l,u,mid;
printf("Enter the size of an array: ");
scanf("%d",&n);
printf("Enter the elements in ascending order: ");
for(i=0;i<n;i++){
scanf("%d",&a[i]);
 }
printf("Enter the number to be search: ");
scanf("%d",&m);
l=0,u=n-1;
while(l<=u){
 mid=(l+u)/2;
if(m==a[mid]){
 c=1;
break;
 }
elseif(m<a[mid]){
 u=mid-1;
 }
else
 l=mid+1;
 }
if(c==0)
 printf("The number is not found.");
else
printf("The number is found.");
return0;
}
```

c. What is the complexity of binary search in best-case, worst case and average case? (2)

Answer:

Complexity in best-case: O (1)
Complexity in worst case: O (log n)
Complexity in average case: O (log n)

Q.8 a. Sort the following sequence of keys using merge sort. (6)
66, 77, 11, 88, 99, 22, 33, 44, 55

Answer:

Sorting using Merge sort:-
Original File : [66] [77] [11] [88] [99] [22] [33] [44] [55]

Pass1 : [66 77] [11 88] [22 99] [33 44] [55]
Pass2 : [11 66 77 88] [22 33 44 99] [55]
Pass3 : [11 22 33 44 66 77 88 99] [55]
Pass4 : [11 22 33 44 55 66 77 88 99]

b. Describe insertion sort with a proper algorithm. What is the complexity of insertion sort in the worst case? (4+2)

Answer:

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.

```
INSERTION-SORT (A)
1  for j <- 2 to length[A]
2      do key <- A[j]
3        Insert A[j] into the sorted sequence A[1 . . j - 1].
4       i <- j - 1
5       while i > 0 and A[i] > key
6          do A[i + 1] <- A[i]
7             i <- i - 1
8       A[i + 1] <- key
```

Because of the nested loops the complexity of insertion sort is O (n 2).

c. Which sorting algorithm is easily adaptable to singly linked lists? Explain your answer. (4)

Answer:

Simple Insertion sort is easily adabtable to singly linked list. In this method there is an array link of pointers, one for each of the original array elements. Initially link[i] = i + 1 for 0 < = i < n-1 and link[n-1] = -1. Thus the array can be thought of as a linear link list pointed to by an external pointer first initialized to 0. To insert the kth element the linked list is traversed until the proper position for x[k] is found, or until the end of the list is reached. At that point x[k] can be inserted into the list by merely adjusting the list pointers without shifting any elements in the array. This reduces the time required for insertion but not the time required for searching for the proper position. The number of replacements in the link array is O(n).

Q.9 a. What is a file? Write different file modes which are used in C. (4)

Answer:

A file is a collection of related data stored in auxiliary storage device. This definition includes two additionally and essential attribute of file that is data in file being related and storage device. The mere representation of file in machine is in form of 0's and 1's .

Any file has end-of-file (EOF) marker. A file can be opened in read state, write state or in error state. The error state occurs due to illegal operation done either in read or in write modes of file opening. In read state, i.e., file is in read mode, if write operation is performed leads to error state. Similarly, In write state, if read operation is performed it leads to error state. Apart from this, we have append mode where data is written at the end of file, so data is appended. Speaking with respect to our binary file, the modes are rb, wb and ab. Apart from these modes discussed we also have update mode, where we can open the file to perform any operation. There are three update modes r+b, w+b, a+b . In the r+b mode file is initially meant for reading and updating and later we can move to write state by positioning the file. Similarly the other modes.

b. Write a C program to append the content of one file at the end of another file (6)

Answer:

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp1,*fp2;
char ch,fname1[20],fname2[20];
printf("\n enter sourse file name");
gets(fname1);
printf("\n enter sourse file name");
gets(fname2);
fp1=fopen(fname1,"r");
fp2=fopen(fname2,"a");
if(fp1==NULL||fp2==NULL)
{
printf("unable to open");
exit(0);
}
do
{
ch=fgetc(fp1);
fputc(ch,fp2);
}
while(ch!=EOF);
fcloseall();
}
```

c. What is file organization? Discuss various file organization methods, its advantages and disadvantages. (6)

Answer:

) File organization refers to the relationship of the key of the record to the physical location of that record in the computer file.File organization may be either physical file or a logical file. A physical file is a physical unit, such as magnetic tape or a disk.
The various file organization methods are:
Sequential access.
Direct or random access.
Index sequential access.

Sequential access: Here the records are arranged in the ascending or descending order or chronological order of a key field which may be numeric or both. Since the records are ordered by a key field, there is no storage location identification. It is used in applications like payroll management where the file is to be processed in entirety, i.e each record is processed. Here, to have an access to a particular record, each record must be examined until we get the desired record. Sequential files are normally created and stored on magnetic tape using batch processing method.

Advantages:

- Simple to understand.
- Easy to maintain and organize
- Loading a record requires only the record key.
- Relatively inexpensive I/O media and devices can be used.
- Easy to reconstruct the files.
- The proportion of file records to be processed is high.

Disadvantages:

- Entire file must be processed, to get specific information.
- Very low activity rate stored.
- Transactions must be stored and placed in sequence prior to processing.

- Data redundancy is high, as same data can be stored at different places with different keys.
- Impossible to handle random enquiries.

Direct access files organization:(Random or relative organization). Files in his type are stored in direct access storage devices such as magnetic disk, using an identifying key. The identifying key relates to its actual storage position in the file. The computer can directly locate the key to find the desired record without having to search through any other record first. Here the records are stored randomly, hence the name random file. It uses online system where the response and updation are fast.

Advantages:

- Records can be immediately accessed for updation.
- Several files can be simultaneously updated during transaction processing.
- Transaction need not be sorted.
- Existing records can be amended or modified.
- Very easy to handle random enquiries.
- Most suitable for interactive online applications.

Disadvantages:

- Data may be accidentally erased or over written unless special precautions are taken.
- Risk of loss of accuracy and breach of security. Special backup and reconstruction procedures must be established.
- Less efficient use of storage space.
- Expensive hardware and software are required.
- High complexity in programming.
- File updation is more difficult when compared to that of sequential method.

Indexed sequential access organization: Here the records are stored sequentially on a direct access device i.e. magnetic disk and the data is accessible randomly and sequentially. It covers the positive aspects of both sequential and direct access files.

The type of file organization is suitable for both batch processing and online processing.

Here, the records are organized in sequence for efficient processing of large batch jobs but an index is also used to speed up access to the records.

Indexing permit access to selected records without searching the entire file.

Advantages:

- Permits efficient and economic use of sequential processing technique when the activity rate is high.

- Permits quick access to records, in a relatively efficient way when this activity is a fraction of the work load.

Disadvantages:

- Slow retrieval, when compared to other methods.
- Does not use the storage space efficiently.

Hardware and software used are relatively expensive.

<u>Text Book</u>

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