**AlGORITHMS**

**1.ANALYSIS OF ALGORITHAMS**

Types of time function:

O(1) --- constant function -->f(n)=1, f(n)=5

O(logn) -- logerethemic

O(n)--- linear -->f(n)=2n+5,f(n)=n/2 +4

O(n2)--quadratic -->f(n)=2n2+3

O(n3)-- cubical

O(3n)--exponential

asending order condition:

1<logn<root(n)<n<nlogn<n2<n3<.……<2n<3n…..

ASYMPOTIC NOTATIONS:

O(big-oh) --> Upper Bound

Ɵ (Theta) --> Average bound

Ω (Omega) --> lower bound

Big-oh :

The function f(n)=O(g(n)) iff all (Revers E) +ve Constants c and n0

Such that f(n)<=c\*g(n) for all n>=n0

exp: f(n)=2n+3 <= 2n+4 or 3n+3 for all n>=n0 so f(n)=O(n)

Omega:

The function f(n)=Ω(g(n)) iff all (Revers E) +ve Constants c and n0

Such that f(n)>= c\*g(n) for all n>=n0

exp: f(n)=2n+3 <= n or nlogn for all n>=n0 so f(n)=Ω(n) or Ω(logn)

Theta-oh :

The function f(n)=O(g(n)) iff all (Revers E) +ve Constants c1 c2 and n0

Such that c1\*g(n) <= f(n)<=c2\*g(n) for all n>=n0

exp: f(n)=2n+3

n<=f(n)<=5n

**Properties of Asymptotic Notations:**

1.Gene renal properties:

If f(N) is O(g(n)) then a\*f(n) is O(g(n));

Exp: f(n)= 2n2+5 is O(n2) then 7f(n) ->7(2n2+5) also O(n2)

2.Reflexive:

If f(n) is given then f(n) is O(f(n))

Exp: f(n)=n2 O(n2)

3.Transitive:

If f(n) is O(g(n)) and g(n) id O(h(n))

Then f(n)=O(h(n))

Exp: f(n)=n g(n)=n2  h(n)=n34

4.Symmetric:

If f(n) is Ɵ(g(n)) then g(n) is Ɵ(f(n))

Exp:f(n)=n2 g(n)=n2

f(n)=Ɵ(n2) and g(n)=Ɵn2)

5.Transpose Symmetric:

If f(n)=O(g(n)) then g(n) is Ω(f(n))

Exp:f(n)=n g(n)=n2

Then n is O(n2) and n2 is Ω(n)

6.If f(n)=O(g(n)) and f(n)=Ω(g(n))

Then g(n)<=f(N)<=g(n) then f(n)=Ɵ(g(n))

1. if f(n) =O(g(n)) and d(n)=O(e(n)) then f(n)+d(n)=O(max(g(n),e(n)))

Exp:f(n)=n ->O(n) and d(n)=n2 ->O(n2)

Then f(n)+d(n)=n+n2=O(n2)

1. if f(n)=O(g(n) and d(n)=O(e(n))

Then f(n)\*d(n)=O(g(n)\*e(n))

**COMPARISON OF FUNCTION:**

Log Formula:

log ab= log a+ log b

Log a/b = log a -log b

Log ab = b log a

alog(c)^b = blog(c)^a

ab = n Then b=log an

a+ar+ar2+ar3+ar4+……ark =a(rk+1-1)/(r-1)

1. to find the smaller or big function the can apply log to bot side.if we applied,don’t ignore the scaler value.
2. If direct function,we can ignore the scaler value.

**Best,Worst and Average case Analysis:**

Linear Search: checking the key element in list of all elements from left to right

Best case: key element in first index:O(1)->B(n)=O(1)

Worst case:Key at last index ->n W(n)=O(n)

Average case: all possible case time/No.of Case

->1+2+3+..n/n ->(n(n+1)/2)/n ->(n+1)/2 so Avg is A(n) = (n+1)/2

Binary Search Tree:

Hight of tree is :log n or n

Best case: Searching element key is head ->O(1)

Worst case: search for leaf element ->w(n)=Hight of the tree

Min w(n)=logn

Max w(n)=n

**DISJOINT SETS:**

No axis are common to both the set

FIND and UNION.

Criscal algoritham to find the loops or cycles

Colapsing rule.

**DIVIDE AND CONQUER**

Divide the big problem into small problems and get the solution after add all solutions

1. Binary Search
2. Finding maximum and minimum
3. merge sort
4. quick sort
5. Strassen’s matrix multiplication

Recurrence Relation:

Exp:Test(int n)

{

If(n>0)

{ T(n)= {1 n=0

Printf(“%d”,n); {T(n-1)+1 n>0

Test(n-1)

}

}

Time is T(n)=1+n ->Ɵ (n) because T(n)=T(n-n)+n=>T(n)=1+n

Exp2:

Test(int n)

{

If(n>0) =>1 T(n)=T(n-1)+2n+2 ==>T(n)=T(n-1)+n

{

For(I=0;I<n:I++)=>n+1 times T(n)= {1 n=0

{ {T(n-1)+n n>0

Printf(“%d”,n);=>n times

}

Test(n-1)

}

}

Time is T(n)=n(n+1)/2 ->Ɵ (n2) because T(n)=T(n-2)+(n-1)+n=>T(n)=T(n-3)+(n-2)+(n-1)+n..

In those Above examples:

T(n)=T(n-1)+1 ==>O(n)

T(n)=T(n-1)+n ==>O(n2)

T(n)=T(n-1)+logn ==>O(nlogn)

T(n)=T(n-1)+n2 ==>O(n3)

T(n)=T(n-2)+1-n/2 ==>O(n)

T(n)=T(n-infinity)+n ==>O(n2)

T(n)=2T(n-1)+1 ==>O(2n) =>2[2T(n-2)+1]+1=>22[T(n-2)]+2+1

If n=k=>2kT[n-k)+2k-1+2k-2+.…+22+2+1=>2n+2k -1=>2n+1-1

Master Theorem for Decreasing function

General form of recurence relation: T(n)=aT(n-b)+f(n)

where a>0 b>0 and f(n)=O(nk) where k>=0

then case1:a<0 O(n2) or O(f(n))

case2:a=1 O(nk+1) or O(n\*f(n))

case3:a>1 O(nkan/b) or O(f(n)\*an/b)

Dividing function:

Test(int n)

{

if(n>1) T(n)={1 n=1

{ {T(n/2)+1 n>1

p.f(“%d”,n); Steps:max n/2k  means n=2k then k =logn

Test(n/2); O(logn)

}

}

Master Theorem for dividing function:

General fun: T(n)=aT(n/b)+f(n)

a>=1 and b>1 f(n)= Ɵ(nk logpn)

1.log ba

2.k case1: if lo log ba >k then Ɵ(nlog ba )

case2: if log ba  =K

if p>-1 Ɵ(nklogp+1n)

if p=-1 Ɵ(nk loglogn)

if p<-1 Ɵ(nk)

case3: if log ba <k if p>=0 Ɵ(nk logpn)

if p<0 O(nk)

Root Function:

Test(int n)

{

if(n>2)

{

stmt; T(n)={1 n=2

Test(root n) {T(root n) n>2

} assume n=2m  Ɵ(loglog2n)

}

Binary Search:

1. elements should in sorted order.(took low L and high H indexes)

2.calculated the mid index value and check the is greater or less

3.if greater,increase the mid +1 value assigned to L and again calculate the mid index value.

4.again check the value is greater or less.

5.if less, assign the mid-1 value to H. this process is going on until to find

Binary search(A,n,Key)

{

L=1,H=n

while(L<=H)

{

mid=(L+H)/2

if(Key=A[mid])

{

return found(mid)

}

if(Key<A[mid])

{

H=mid-1;

}

else

{

L=mid+1;

}

}

}

order of binary search is min→O(1)

max→O(logn) avarage→logn

BinarySearchRecursice(l,h,Key)

{

if(l==h)

{

if(A[l]==key)

return l;

else

return 0;

}

else

{

mid=(l+h)/2;

if(key == A[mid])

return mid;

if(key < A[mid])

return BinarySearchRecursice(l,mid,key);

else

return BinarySearchRecursice(mid+1,h,key);

}

}

Time T(n)= {1 n=1

{T(n/2)+1 n>1

**HEAP**

If a node is at index ->I

It’s left child is at ->2\*I

It’s right child is at ->2\*I+1

It’s parent is at ->(I/2)

Full binary tree:don’t have missing elements in an array,

Complete binary tree: should have the elements 2h+1-1 where h is high of the tree

HeapSort:

1. Create the heap from the given elements
2. Delete the elements from the heap and add at end of given location.

Creating the heap:O(nlogn)

Heapify:O(n)

PriorityQueue

Smaller number is high priority in min heap

Larger number is high priority in max heap

**MERGE**

List may array or linked list

Algorithm Merge(A,B,m,n)

{

I=1,j=1,k=1;

While(I<=m &&j<=n)

{

If(A[I]<B[j])

C[k++] = A[I++];

Else

C[k++] = B[I++];

}

For(;I<=m;I++)

C[k++] = A[I++];

For(;j<=n;j++)

C[k++]=b[j];

}

2 way merge sort ->iterative(loop) process

Merge sort->recursive process(divide and conquer algorithm.

2 way merge sort:

1. > 9 3 7 5 6 4 8 2 ->each list having one element. Sort 2 elements with next value

1Iterat-> 3 9 5 7 4 6 2 8

2ite-> 3 5 7 9 2 4 6 8

3rd iter-> 2 3 4 5 6 7 8 9 No of passes are logn where n=8

Merge sort:

Mergesort(L,H)

{

If(L<H)

{

Mid=(L+H)/2

Mergesort(L,mid)

Mergesort(mid+1,h);

Merge(L,mid,H)

}

}

Time taken T(n)= O(nlogn) and Theta(nlogn)

PROS:

1. Large size of list
2. Linked list
3. External sorting
4. Stable(arrangement of values-> 8 8 9)(bubble,insertion sorts also stable)

CONS:

1. Extra space
2. No small problem (if small problem, it slow process so internally use insertion sort)
3. Recursive.

**QUICK SORT**

Divide and conquer algorithm

Pivot : is a value

Form left side search greater than pivot and right side less than Pivot and inter change the values.

Partition(L,H)

{

Pivot =A[L];

i=L;j=H

While(I<j)

{

Do

{

i++

}while(A[I]<=pivot);

Do

{

j--

}while(A[j>pivot);

If(I<j)

Swap(A[I],A[j]);

}

Swap(A[L],A[j])

Return j;

}

QuickSort(L,H)

{

If(l<h)

{

j=Partition(l,h);

QuickSort(l,j);

QuickSort(j+1,h);

}

}

Best case:O(nlogn)

Pivot in the middle

Worst case:O(n2)

Already sorted one trying to sort.

Suggetions:

1. select the middle element as pivot
2. Select random element as pivot

**Strassens Matrix multiplication**

Algorithm MM(A,B,n)

{

If(n<=2)

{

Use the 2\*2 matrix multiplication

}

Else

{

Mid --- n/2

MM(A11,B11,n/2)+(A12,B21,n/2)

MM(A11,B12,n/2)+(A12,B22,n/2)

MM(A21,B11,n/2)+(A22,B21,n/2)

MM(A21,B12,n/2)+(A22,B22,n/2)

}

}

T(n)={1 n<=2

{8T(n/2)+n2 n>2

Master theorm:a=8 ->logab=log28=3

B=2

F(n)=n2 =>nk=n2=>k=2

So theta(n3)

Final Formula for strassens:

P=(A11+A22)(B11+B22)

Q=(A21+A22)B11

R=A11(B12-B22)

S=A22(B21-B11)

T=(A11+A12)B22

U=(A21-A!!)(B11+B12)

V=(A12-A22)(B21+B22)

C11=P+S-T+V

C12=R+T

C21=Q+S

C22=P+R-Q+U

By using this 7 multiplications

T(n) ={1 n<=2

{7T(n/2)+n2 n>2

Log27=2.81 k=2

O(nlog27) =O(n2.81)

**GREADY METHOD**

Just like divide and coquer or other like to to fine the solution

Exp: if I want to travel from A->B

Sol: walk,cars,bike,bus,train,flight

Constrain a problem like need to complete the journey in 12hrs

So trains or flights these solution is called feasible solution.

One more is minimum cost is called minimisation is called optimal solution

Train is solution is called optimum solution

Optimization problem - >A problem which requires minimum or maximum solution

Algorithm Gready(a,n)

{

For(I=0;I<n;I++)

{

X=Select(a)

If feasible(a) then

Solution=solution+X

}

}

Knapsack Problem:

Some objects are there to fill in a 15kg bag. Each object having some weight with profit.

So need to calculate the profit per KG and which one is high profit select that one first.

Job Sequencing with Deadlines:High profit job first allocate space ,if space is filled check the left side space for job is available. If free use this to space process

Optimal Merge Pattern: Merge the small size list and merge first.

Huffman Coding: FIXED SIZE CODE:used for reducing the size for transmitting or storing the file.

Exp: message:BCCABBDDAECCBBAEDDCC

Character count code

A 3 000

B 5 001

C 6 010

D 4 011

E 2 100

For decoding purpose need to send the table so each char is 8 bits so 5\*8 and each byte has the code with 3 bits 5\*3 so total length is=5\*8+5\*3+20\*3 = 115 bits

VARIABLE SIZE CODE:

Character count code Left side is mark as 0 and right side mark as 1

A 3 001 count order

B 5 10 E A D B C

C 6 11 2 3 4 5 6

D 4 01 5 11

E 2 000 9

20

Minimum cost spanning tree:it won’t have a cycle

Graph G=(Vertices V, Edges E)

For spanning tree, need to take all vertices but Edges may vary E=V-1;

If edges are E = 6 and vertices V= 5 then spanning tress possible are |E|C|V|-1- no.of cycles.

Prim’s algorithm:

First select the smallest one, then after onwards select the smallest connected one.

It won’t work on non connected graphs

Kruskal’s

Always select smallest one from set of edges,if it creating a cycle.Ignore the edge.

It may work on non connected graphs

For spanning tree Edges: V-1

Theta(VE)

Theta(n\*e)==>Theta(n2) if n=e

For min heap time is T(n)=Theta(nlogn) so it is faster.

Dijkstra Algorithm:Single source shortest path algorithm

This algorithm gives the minimal solution

Always select the shortest path and select the other vertices In indirect way to find the shortest path if possible.This is called Relation.

Relaxation:if(d(u)+c(u,v)<d[v])

D[v]=d[u]+c(u,v)

If each vertices is connected to each one then worst case is n2 so theta(n2)

This algorithm is work on non connected graph also.

This will may work on Negative weighted edges also.

**DYNAMIC PROGRAMMING**

Follow the principle of optimality

Every stage we take a decision

For fibanaci series we will use the recursive function so O(2n) because it took 15 calls

Algorithm for finaci series

Int fib(int n)

{

If(n<=1) //top-down approach

Return n;

Return fib(n-2)+fib(n-1)

}

If we got the ans in calls store in a global variable. And if not called the function previously call the function and update the global buffer so no of call n+1 calls so O(n)

Int fib(int n)

{

If(n<=1) //bottom-up approach

Return n; //tabulation method

F[0]=0,F[1]=1;

For(int I=2;I<=n;I++)

{

F[I]=F[I-2]+F[I-1];

}

Return F[n]

}

Multi stage Graph:

Cost(Stage I,Vertex No. J) = min{C(j,l)+Cost(I+1,l) j,l belogs to E and l belogs to Vi+1

Main()

{

Int stages=4,min

Int cost[9],d[9],path[9]

Int c[9][9]={{0,0,0,0,0,0,0,0,0},

……………

Cost[n]=0;

For(int I=n-1;I>=1;I--)

{

Min=32767;

For(k=I+1;k<=n;k++)

{

If(c[I][k]!=0 &&c[I][k]+c[k]<min)

{

Min=c[I][k]+c[k];

D[I]=k;

}

}

Cost[I]=min;

}

P[I[=1;p[stages]=n;

For(I=2,I<stages;I++)

P[I]=p[d[I-1]];

}

O(n2)

All pairs shortest path:

Formula:Ak[I,j]=min(Ak-1[I,j],Ak-1[I,k]+Ak-1[k,j]);

For(k=1;k<=n;k++) //k is intermediate vertex

{

For(I=1;I<=n;I++)

{

For(j=1;j<=n;j++)

{

A[I,j]=min(A[I,j],A[I,k]+A[k,j]);

}

}

}

O(n3)

Matrix Chain Multiplication:

If we have 5by4 and 4by 3 matrix multiplication, it will give 5by3 so 15 elements will come and for each element we are doing 4 so 15\*4=>60 multiplications are doing

node we can do T(n)=2(n-1)Cn-1/(n) possible trees or parenthesise =>A1.A2.A3.A4=> (((A1.A2).A3).A4) or => (A1.A2).(A3.A4)…

Formula:

M[I,j]=min{m[I,k]+m[k+1,j]+di-1\*dk\*dj}

I<=k<j

O(n3)

A1 \* A2 \* A3

2\*3 3\*4 4\*2

D0 d1 d1 d2 d2 d3

O(n3)

Code:

Main()

{

Int n=5;

Int p[]={5,4,6,2,7};

Int m[5][5]={0};

Int [5][5]={0};

Int j,min,q; m 0 1 2 3

For(int d=1;d<n-1;d++) //d means difference 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 120 | 88 | 153 |
| 0 |  | 0 | 18 | 104 |
| 0 |  |  | 0 | 89 |
| 0 |  |  |  | 0 |

{ I 1

For(int I=1;I<n-d;I++)// row 2

3

{

J=I+d;

Min=32767; S 0 1 2 3 4

For(int k=1;k<j-1;k++) 1 0 1 1 3

{ 2 0 2 3

Q=m[I][k]+m[k+1][j]+p[I-1]\*p[k]\*p[j]; 3 0 3

If(q<min) 4 0

{

Min=q;

S[I][j]=k;

}

}

M[I][j]=min;

}

Cout<<m[1][n-1]

}

}

**Single Source Shortest path:**

**Bellman-ford algorithm:** it will work on -ve values,dijsra algorithm wont work on -ve.

Relaxation to be done up to V-1 times.

O(|E||V|-1) =>O(n2) minimum

If the graph is complete graph:O(n3).

Drawback:

Fails,if the graph weight is negative.

**0/1 Knapsack Problem:** Ǝ

V[I,w]=Max{V[I-1,w],V[I-1,w-w[I]]+p[I]}

Main()

{

Int p[5]={0,1,2,5,6}

Int wt[5]={0,2,3,4,5}

Int m=8,n=4;

Int k[5][9];

For(int I=0;I<=n;I++)

{

For(int w=0;w<=m;w++)

{

If(I==0 ||w==0)

K[I][w]=0;

Else if(wt[I]<=w)

K[I][w]=max(p[I]+k[I-1][w-wt[I]],k[I-1][w])

Else

K[I][w]=k[I-1][w];

}

}

Cout<<k[n][w];

While(I>0 &&j>0)

{

If(k[I][j] == k[I-1][j])

{

Cout <<I<<”=0”<<endl;

I--;

}

Else

{

Cout <<I<<”=1”<<endl;

1. -;
2. J=j-wt[I];

}

}

}

**Optimal Binary Search tree:**

For n node binary trees are possible 2nCn /(n+1).

In those binary trees, one will give the best result

C[I,j]=min{c[I,k-1]+c[k,j]}+w(I,j)

I<K<=J

Cost of Binary Tree:C[0,n]=ƐPi\*level(ai)+ƐQi\*(level(Ei)-1)

1<=I<=n 0<=I<=n

C[I,j]=min{c[I,k-1]+c[k,j]}+w[I,j]

I<k<=j

W[I,j]=w[I,j-1]+pi+qj

**Traveling Salesman problem :**

g(i,s)=min{cik+g(k,s-{k})}

Kbelongs to s

**Reliability Design:**

Ui=(c**-**Ɛci)/ci

**Longest Common subsequence(LCS):**

Int LCS(I,j)

{

If((A[I]==’\0’ ||B[j] == ‘\0’)

Return 0;

Else if(A[I] == B[I])

Return 1+LCS(I+1,j+1)

Else

Return max(LCS(I+1,j),LCS(I ,j+1));

}

Tabulation:

If(A[I]==b[j])

LCS[I,j]=1+LCS[I-1,j-1]

Else

LCS[I,j]=max(LCS[I-1,j],LCS[I,j-1])

**Breth for search and Depth for search(BFS and DFS) :**

Visiting vertex

Exploration of vertex

BFS:level order

DFS:Pre order

**Articulation point:**

Bi connected components:

If a point is removed from the network/Graph.The network may split into 2 parts.That node is called articulation point.

To find

1. First do DFS

**Back Tracking**

Used when to use all the solution instead of optimized solution.

Brute force approach:

Bounding function.

1. Queens:

1+Sigma[Pi(N-j)]

i=0 j=0

HASHING:

1. open hashing

Chaining

1. closed Hashing

Open Addressing

1. Linear Probing
2. Quadratic Probing.